

# Organização de Computadores

## Professores:

Lúcia Maria de A. Drummond

Simone de Lima Martins

# Organização de Computadores

## Livro Texto:

"Introdução à Organização de Computadores"

Mário A. Monteiro

LTC editora

# Organização de Computadores

## Objetivo:

Proporcionar ao aluno o conhecimento funcional dos diversos blocos e partes que compõem a arquitetura de um computador.

## Ementa:

- Organização lógica e funcional do Modelo de Von-Neumann: conceito, arquitetura lógica e funcional
- Unidades Funcionais: UCP, memória, cache, dispositivos de E/S e barramentos
- Hierarquia de memória
- Arquiteturas micro e nanoprogramadas
- Mecanismos de interrupção e de exceção
- Arquiteturas avançadas: pipeline, múltiplas unidades funcionais e máquinas paralelas
- Processadores RISC e CISC

# Aula 1

## Professores:

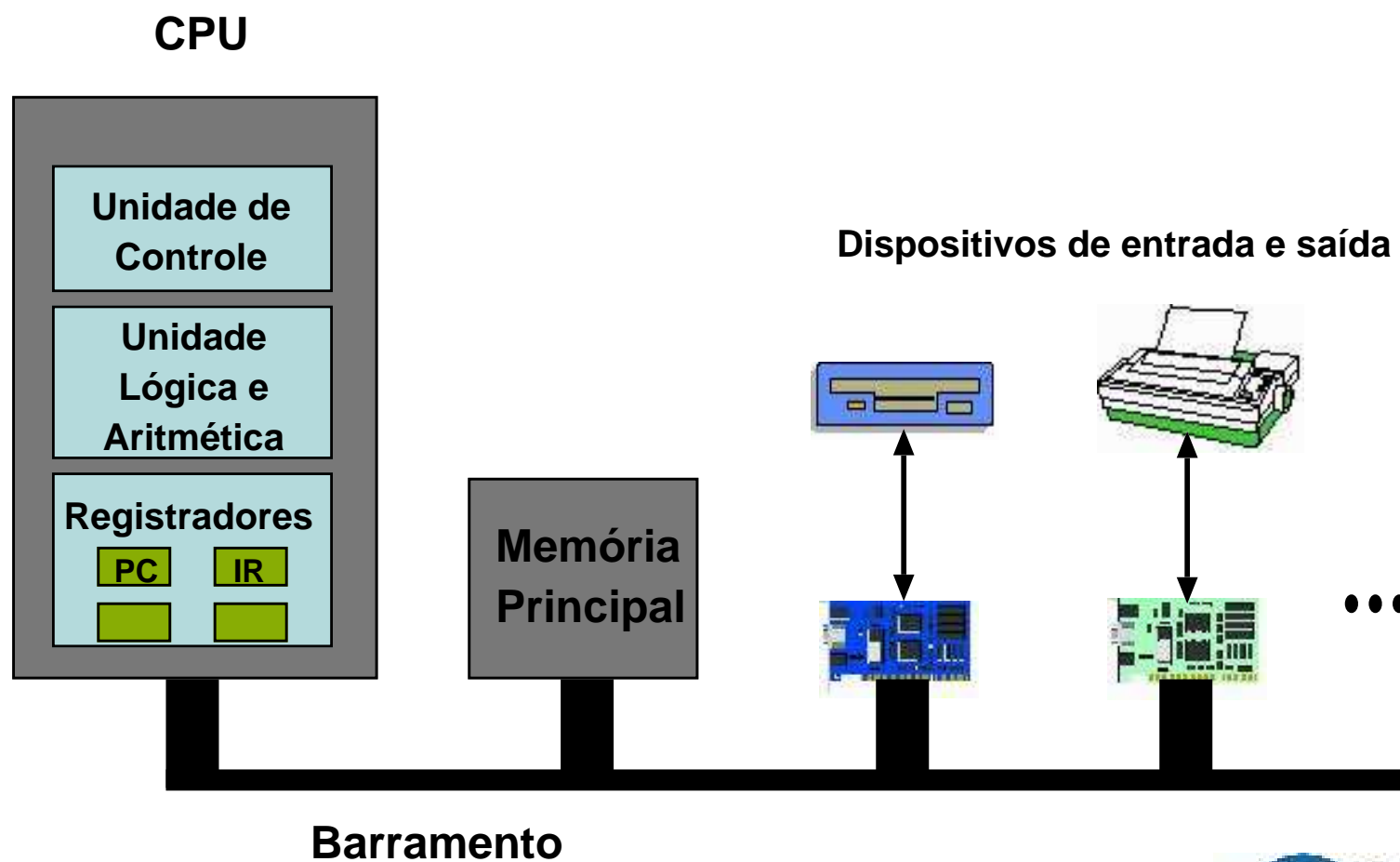
Lúcia M. A. Drummond  
Simone de Lima Martins

## Conteúdo:

### Subsistemas de memória

- Introdução
- Hierarquia de Memória
- Memória Principal
- Erros

# Organização de um Computador



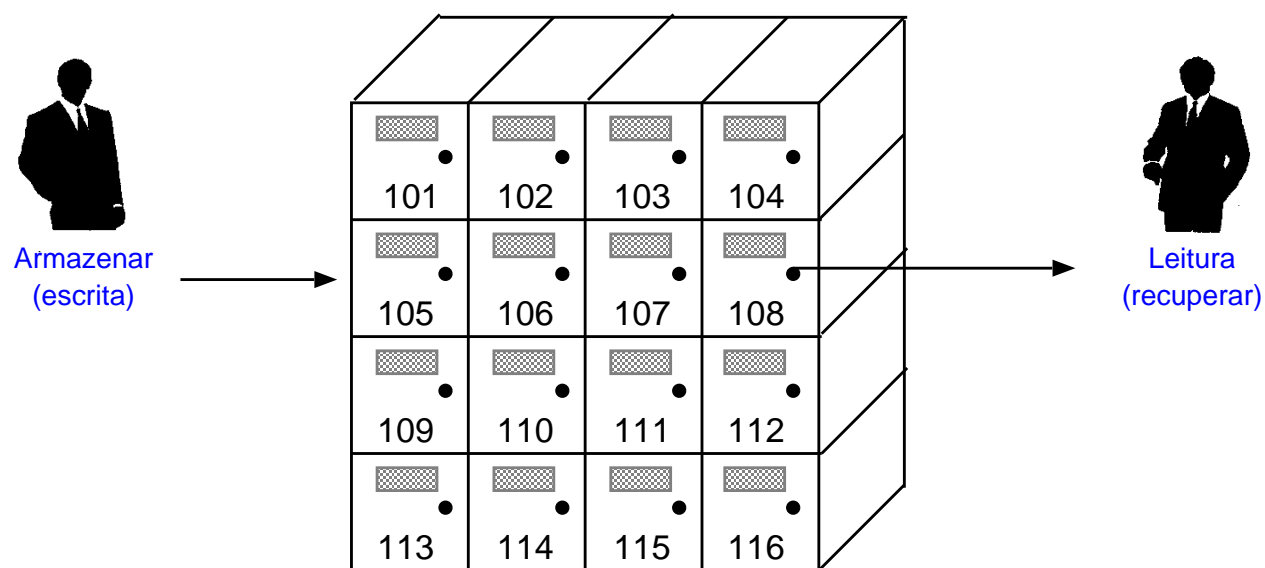
# Introdução

## Memória

"Componente de um sistema de computação cuja função é armazenar as informações que são (ou serão) manipuladas por esse sistema, para que elas (as informações) possam ser prontamente recuperadas, quando necessário."

# Introdução

## Memória: Depósito



(Fig. 5.1 do livro texto)

### Ações:

1. Armazenamento - Escrita ou gravação (*write*);
2. Recuperação - Leitura (*read*)



# Introdução

## Exemplo de Depósito: Biblioteca

1. Elemento: livro
2. Identificação: nome do livro
3. Código de localização: número da estante, da prateleira, etc...

**Armazenamento:** guardar o livro em uma estante previamente identificada

**Recuperação:** através do conhecimento da localização do livro, emprestá-lo

# Introdução

## Representação da Informação na Memória

**Bit:** Elemento básico de armazenamento físico, pode indicar dois valores distintos - 0 ou 1

Para representar:

- 26 letras maiúsculas
- 26 letras minúsculas
- 4 símbolos matemáticos
- 8 sinais de pontuação

*64 tipos de representação - 6 bits*

# Introdução

## Representação da Informação na Memória

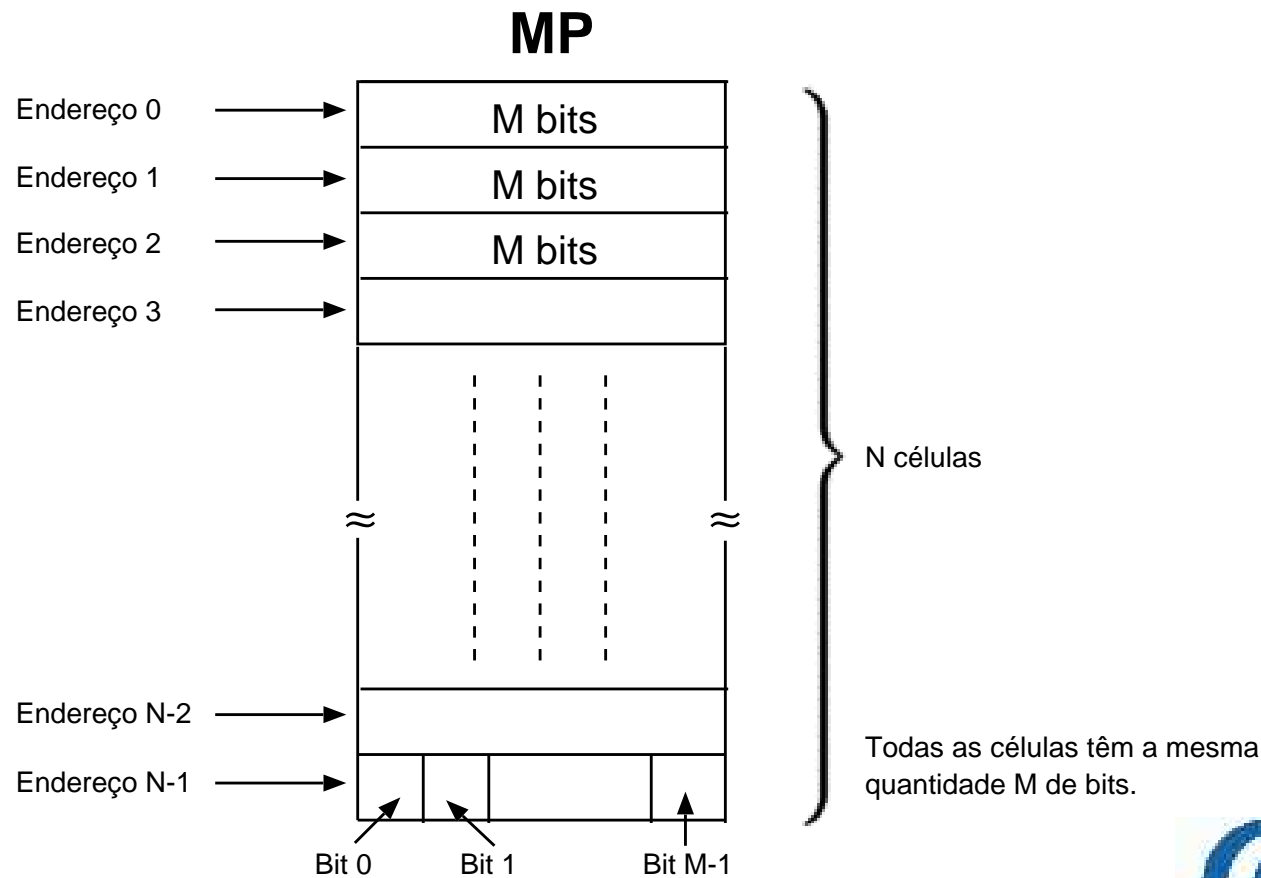
Célula: grupo de bits tratado em conjunto pelo sistema.

*A célula é tratada como uma unidade para efeito de armazenamento e transferência.*

# Introdução

## Localização da Informação na Memória

Cada célula é identificada por um número denominado endereço.

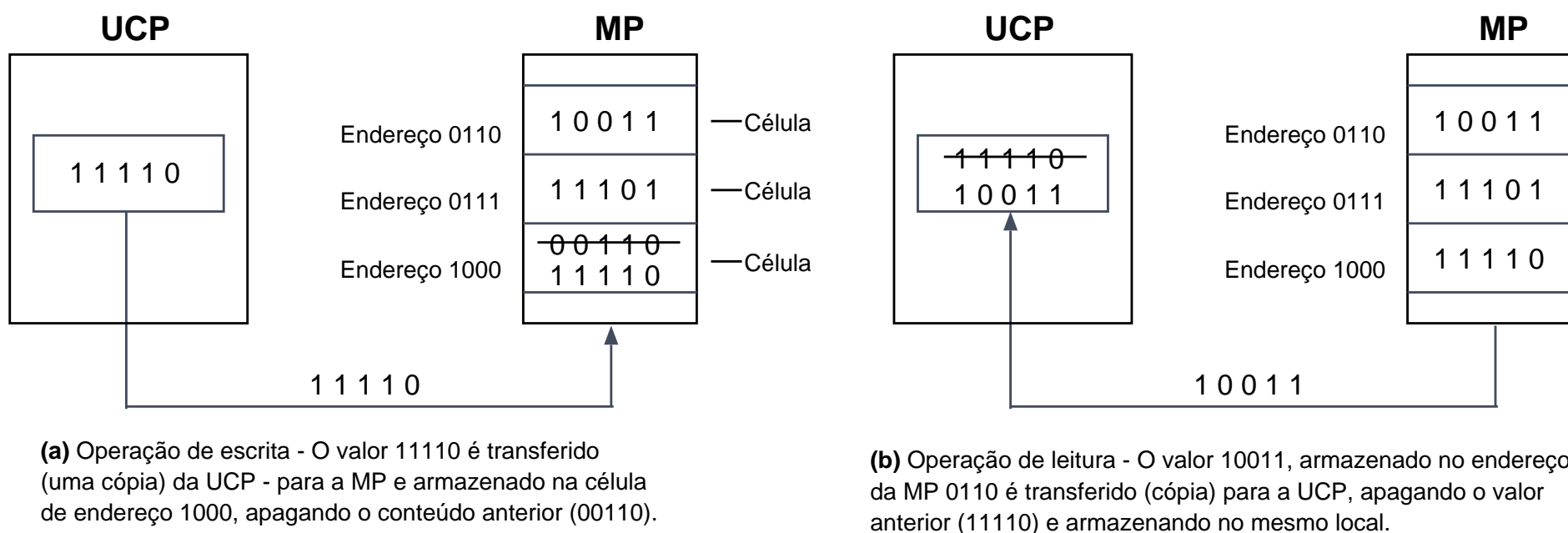


(Fig. 5.8 do livro texto)

# Introdução

## Operações realizadas em uma memória

Escrita e Leitura:

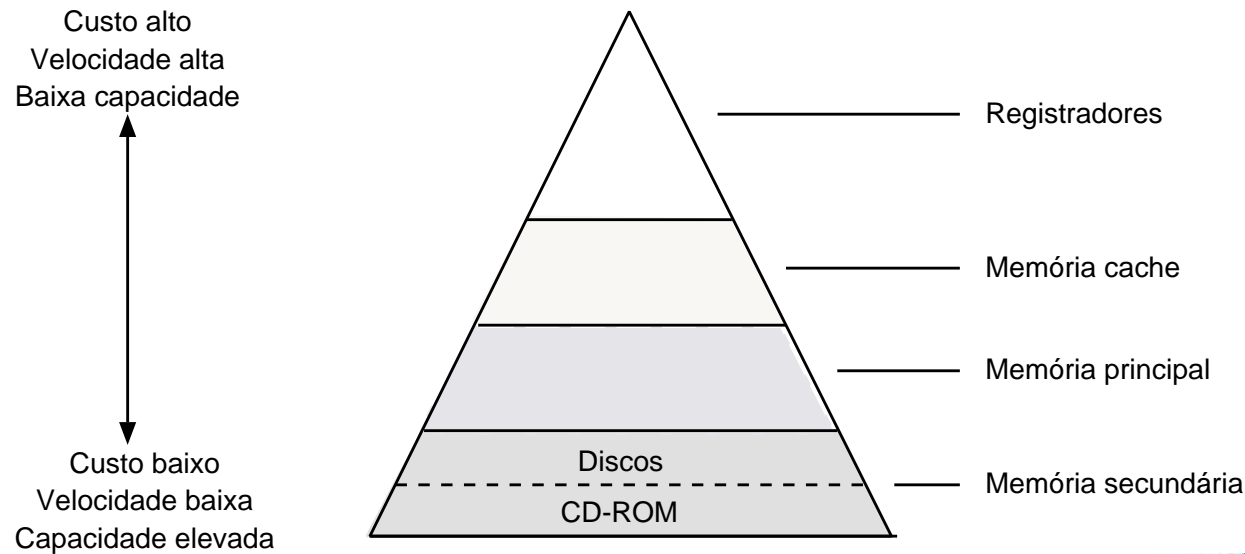


(Fig. 5.3 do livro texto)

# Hierarquia de Memória

Há muitas memórias no computador:

- interligadas de forma bem estruturada
- organizadas hierarquicamente
- constituem o subsistema de memória



(Fig. 5.4 do livro texto)

# Hierarquia de Memória

Parâmetros para análise de cada tipo de memória:

- *Tempo de acesso*: tempo de leitura, ou seja, transferência da memória para a Unidade Central de Processamento (UCP)
- *Capacidade*: quantidade de informação que pode ser armazenada em uma memória

(continua...)

# Hierarquia de Memória

(...cont) Parâmetros para análise de cada tipo de memória:

- *Tecnologias de fabricação:*
  1. Memórias de semicondutores : fabricadas com circuitos eletrônicos, rápidas e caras. Ex: registradores, memória principal (MP) e cache.
  2. Memórias de meio magnético : armazenam as informações sob a forma de campos magnéticos, baratas e de grandes capacidades. Ex: disquetes e discos rígidos.
  3. Memórias de meio ótico : utilizam um feixe de luz para marcar o valor 0 ou 1 de cada dado. Ex: CD-ROM e CD-RW.



# Hierarquia de Memória

(...cont) Parâmetros para análise de cada tipo de memória:

- *Temporalidade*: tempo de permanência da informação na memória. Ex: programas e dados em um disco - memória permanente, programas e dados em memória principal - memória transitória
- *Custo*: varia em função da tecnologia de fabricação. Unidade de medida de custo - preço por byte armazenado, ao invés de custo total da memória, já que há variações nas capacidades.

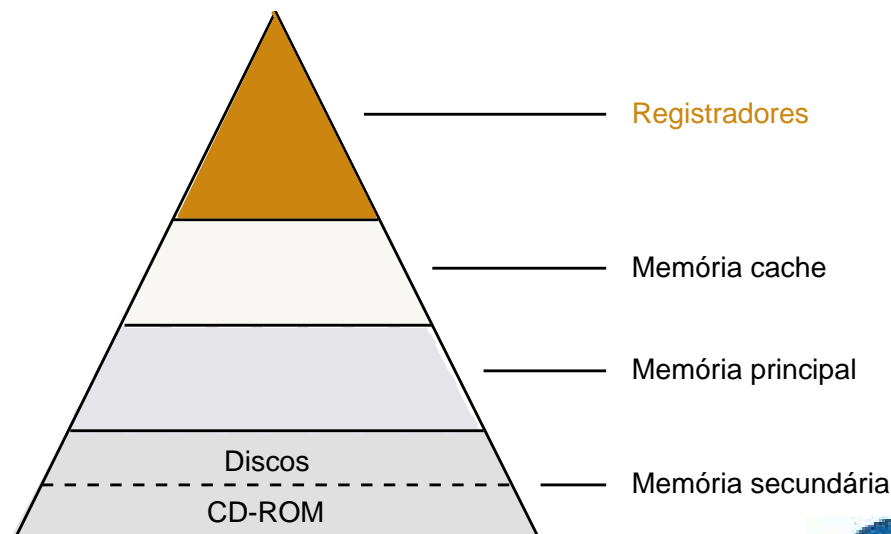
# Hierarquia de Memória

## Registradores

- Pequenas unidades de memória que armazenam dados na UCP.
- Topo da pirâmide: maior velocidade de transferência, menor capacidade de armazenamento e maior custo.

Custo alto  
Velocidade alta  
Baixa capacidade

↑  
↓  
Custo baixo  
Velocidade baixa  
Capacidade elevada



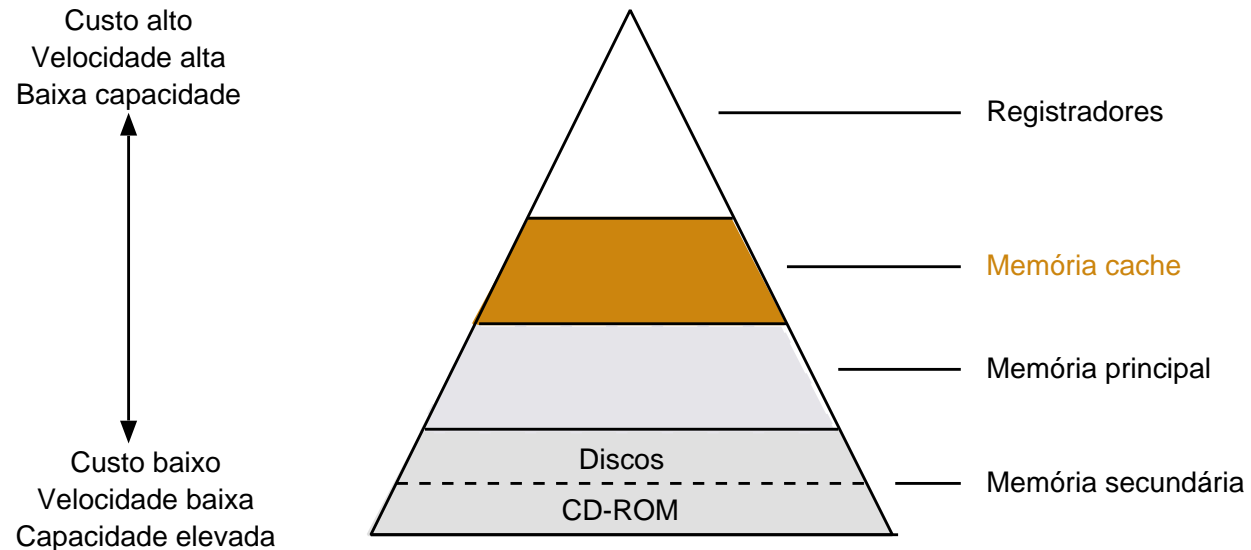
# Hierarquia de Memória

## Registradores - Parâmetros:

- *Tempo de acesso:* 1 ciclo de memória
- *Capacidade:* poucos bits, suficientes para armazenar um dado, uma instrução ou um endereço
- *Volatilidade:* memórias voláteis, precisam de energia elétrica
- *Tecnologia:* memórias de semicondutores (mesma tecnologia dos demais circuitos da UCP)
- *Temporalidade:* armazenam informação por muito pouco tempo
- *Custo:* dispositivo de maior custo entre os diversos tipos de memória

# Hierarquia de Memória

## Memória Cache



- Memória entre a UCP e a Memória Principal
- Função: acelerar a velocidade de transferência das informações entre UCP e MP e, com isso, aumentar o desempenho do sistema.
- A UCP procura informações primeiro na Cache. Caso não as encontre, as mesmas são transferidas da MP para a Cache.
- Podem ser inseridas em dois níveis:
  - Nível 1 - interna ao processador, encapsulada na mesma pastilha
  - Nível 2 - cache externa, pastilha (chip) separada

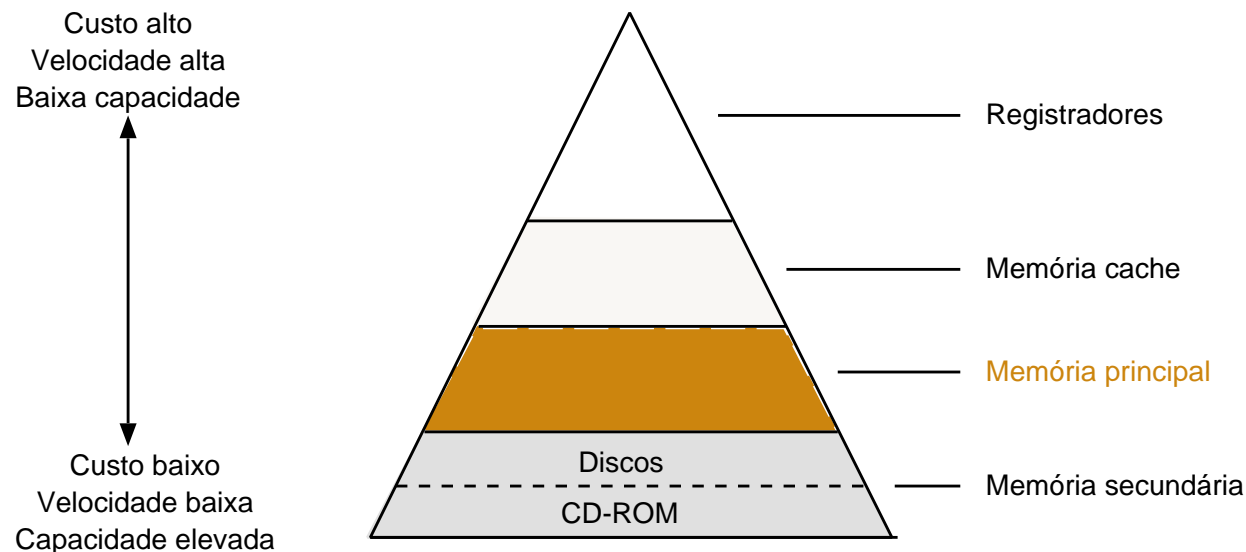
# Hierarquia de Memória

## Memória Cache - Parâmetros:

- *Tempo de acesso:* menores do que os da memória principal
- *Capacidade:* adequada para armazenar uma apreciável quantidade de informações
- *Volatilidade:* dispositivos voláteis, como registradores
- *Tecnologia:* circuitos eletrônicos de alta velocidade, são memórias estáticas denominadas SRAM
- *Temporalidade:* tempo de permanência do dado ou instrução é menor do que o tempo de duração do programa a que pertence
- *Custo:* custo alto, entre o custo de registradores e MP

# Hierarquia de Memória

## Memória Principal:



- Memória básica de um sistema de computação.
- Dispositivo onde o programa (e seus dados) que vai ser executado é armazenado para que a UCP busque instrução por instrução para executá-las.

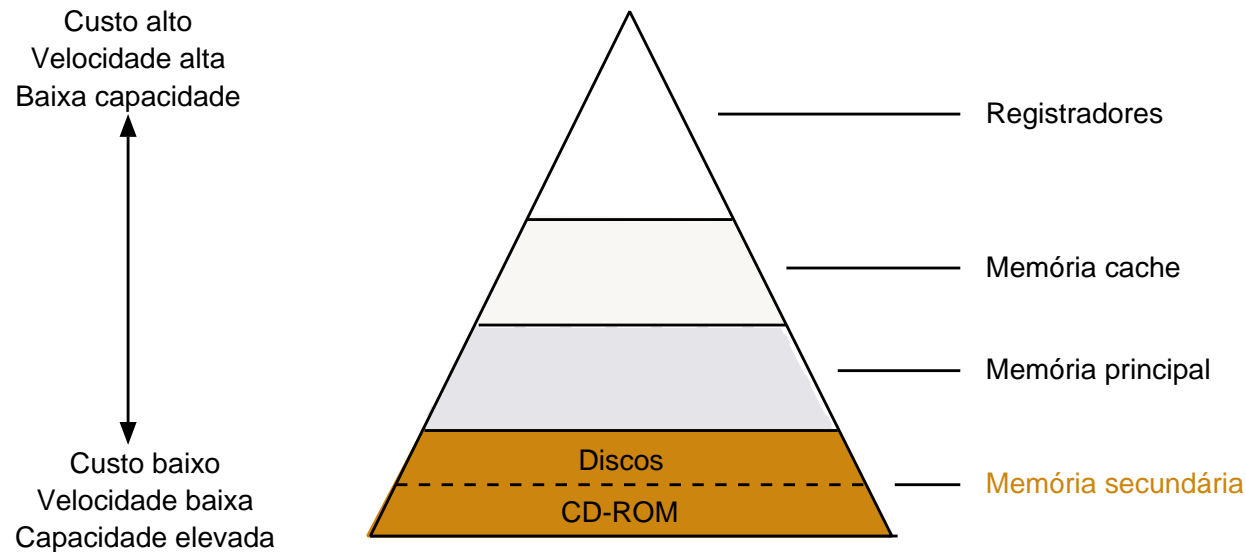
# Hierarquia de Memória

## Memória Principal - Parâmetros:

- *Tempo de acesso:* mais lentas que a memória Cache e mais rápidas do que as memórias secundárias
- *Capacidade:* bem maior do que a da memória Cache
- *Volatilidade:* volátil como a Cache e os registradores
- *Tecnologia:* semicondutores, memória com elementos dinâmicos DRAM
- *Temporariedade:* dados ou instruções permanecem na MP enquanto durar a execução do programa (ou até menos)
- *Custo:* mais baixo que a Cache

# Hierarquia de Memória

## Memória Secundária:



- Memória auxiliar ou memória de massa
- Garantia de armazenamento mais permanente aos dados e programas do usuário
- Alguns diretamente ligados: discos rígidos
- Alguns conectados quando necessário: disquetes, fitas de armazenamento, CD-ROM



# Hierarquia de Memória

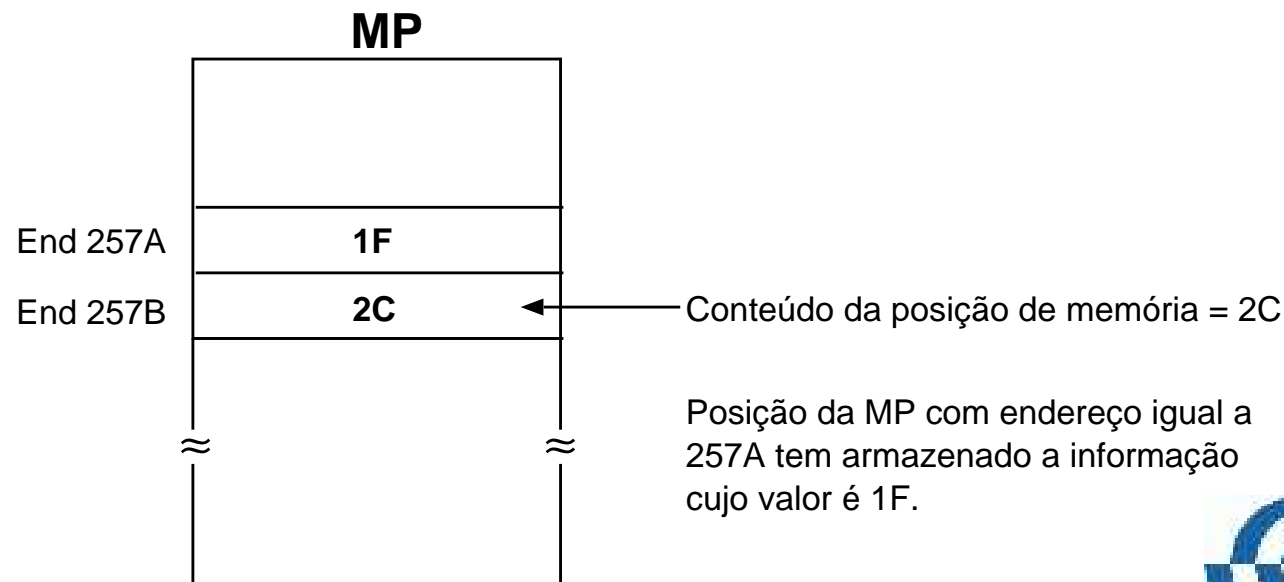
## Memória Secundária:

- *Tempo de acesso:* são geralmente dispositivos eletromecânicos e portanto mais lentos do que os puramente eletrônicos.
- *Capacidade:* bem maior do que as demais memórias.
- *Volatilidade:* como armazenam informação de forma magnética ou ótica, não se perdem quando não há alimentação de energia elétrica.
- *Tecnologia:* varia bastante, diferentes tecnologias
- *Temporalidade:* caráter permanente

# Memória Principal

## Organização da Memória Principal (MP)

- *Instruções e dados são armazenados na MP e a UCP vai buscando-os um a um durante a execução*
- *Os comandos dos programas são descritos seqüencialmente*
- *Palavra é a unidade de informação do sistema UCP/MP que deve representar o valor de um número ou uma instrução de máquina.*
- *Endereço e conteúdo de memória:*



(Fig. 5.7 do livro texto)

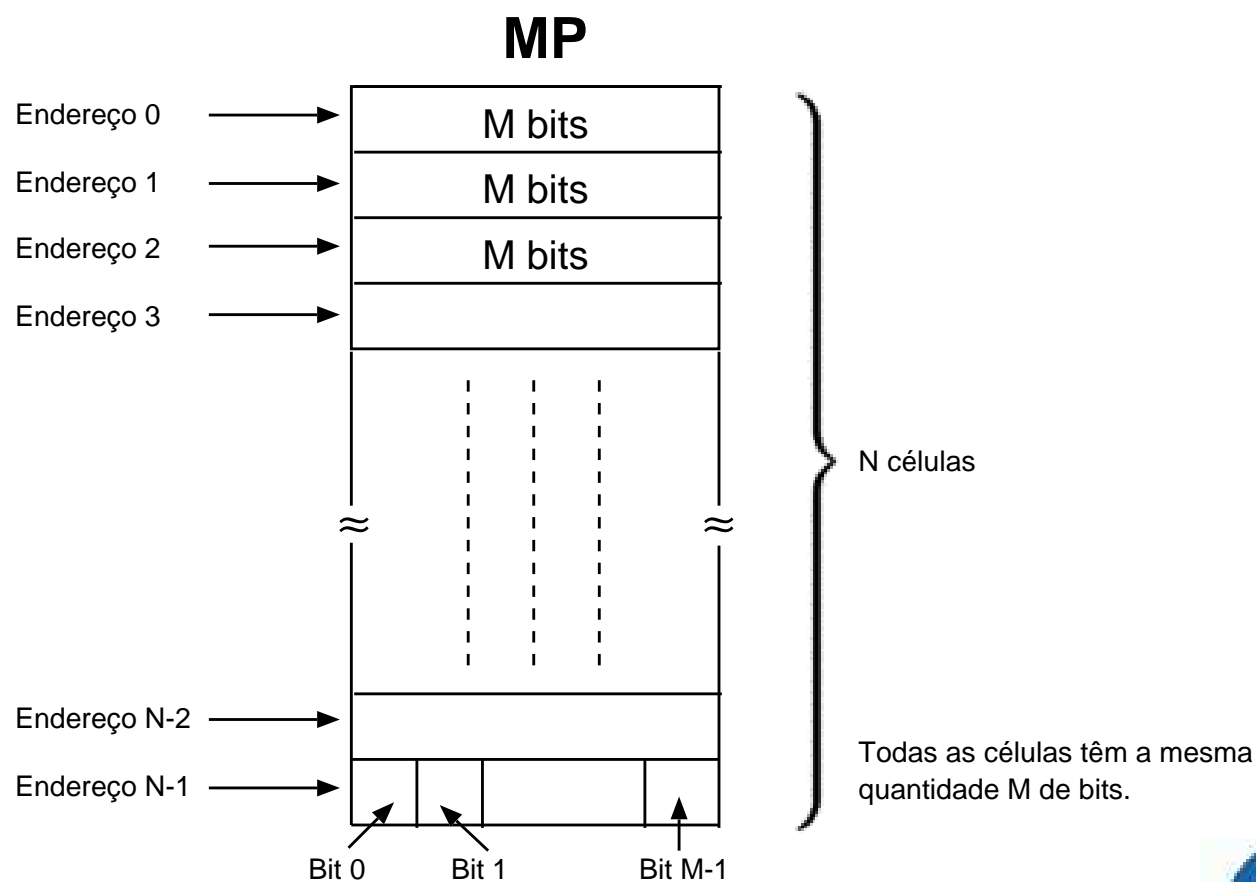
# Memória Principal

## Organização da Memória Principal (MP):

- *Unidade de armazenamento: célula*  
*Palavra x célula*  
  
*Células de 1 byte - 8 bits*  
*Palavras de 16, 32 e até 64 bits*
- *Unidade de transferência: quantidade de bits que é transferida da/para memória em uma operação de leitura/escrita*

# Memória Principal

## Organização da Memória Principal (MP):



(Fig. 5.8 do livro texto)

# Memória Principal

## Organização da Memória Principal (MP)

Características das Memórias de Semicondutores Atuais:

- Memórias de acesso aleatório (RAM- Random Access Memory)
- Ocupam relativamente pouco espaço, muitos bits em uma pastilha (chip)
- Possuem tempo de acesso pequeno

# Memória Principal

## Organização da Memória Principal (MP)

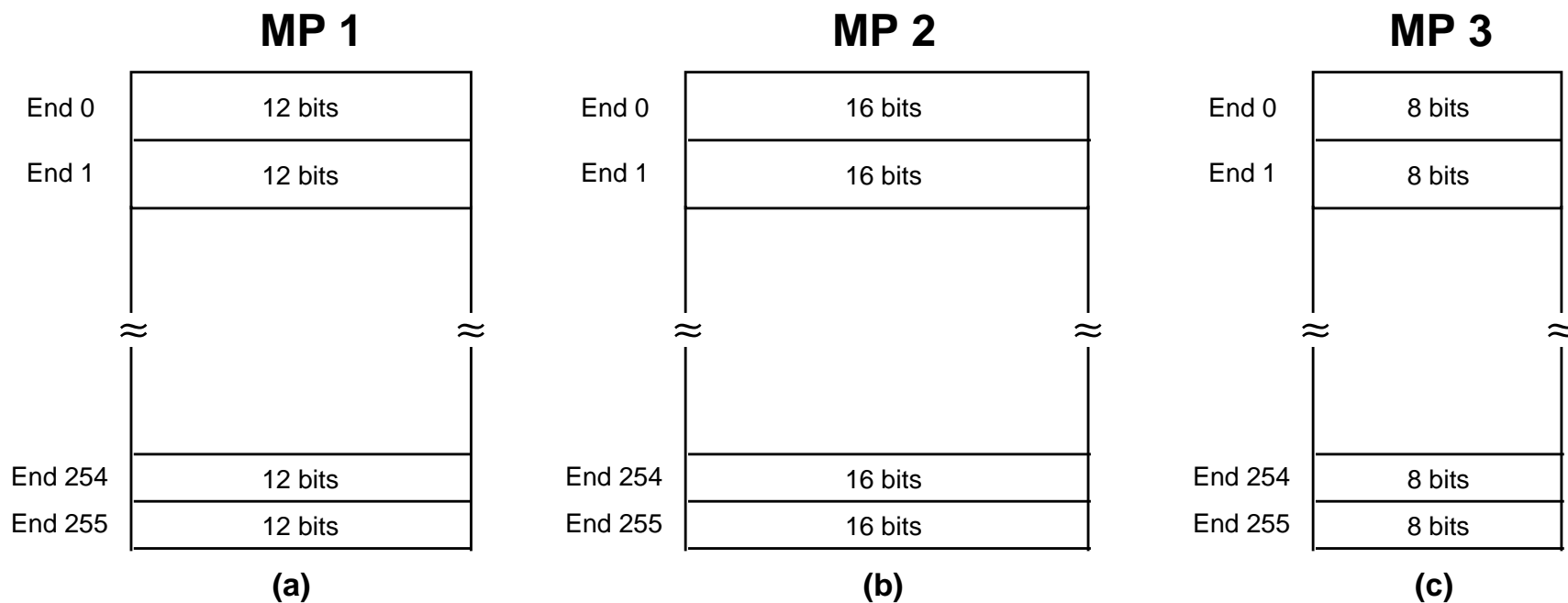
Memórias Somente de Leitura (ROM):

- Tipo de RAM que permite apenas leitura por parte da UCP ou de outros programas
- Gravação deve ser realizada eventualmente e não através de processos comuns
- Mantém permanentemente grupo de instruções que são executadas ao ligarmos o computador com o objetivo de iniciar o sistema

# Memória Principal

## Considerações sobre a Organização da MP

- *Quantidade de bits de uma célula:* requisito definido pelo fabricante. Usualmente 8 bits.
- *Relação endereço x conteúdo de uma célula:*



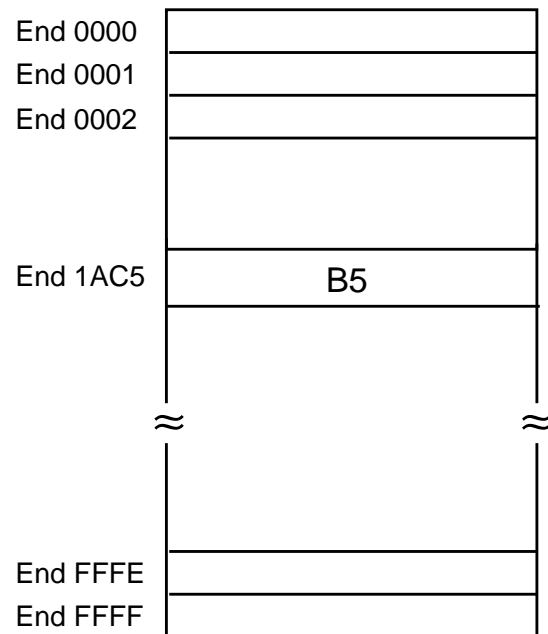
(Fig. 5.9 do livro texto)

# Memória Principal

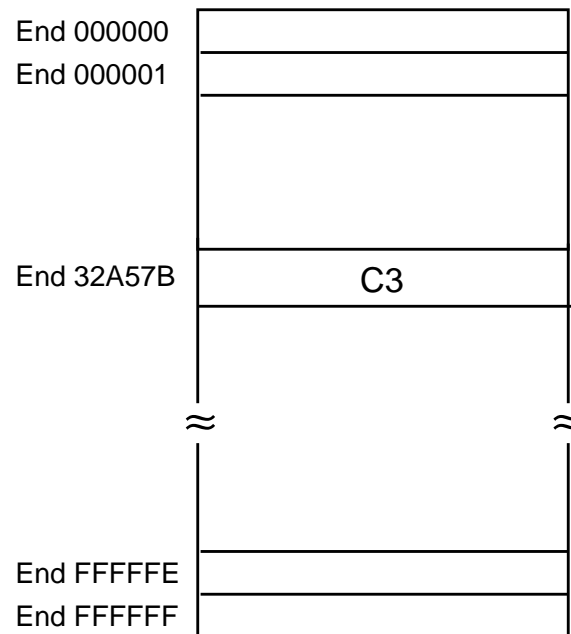
## Considerações sobre a Organização da MP

- Quantidade de bits do número que representa um endereço define a quantidade máxima de endereços que uma memória pode ter.

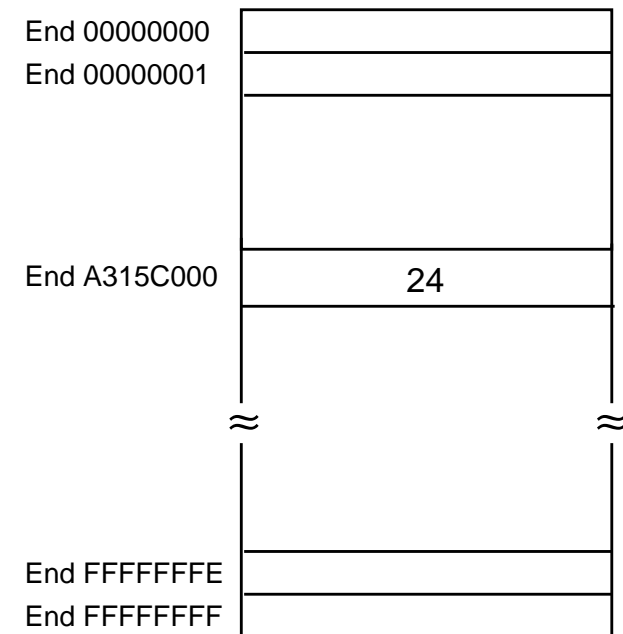
**MP 1**



**MP 2**



**MP 3**



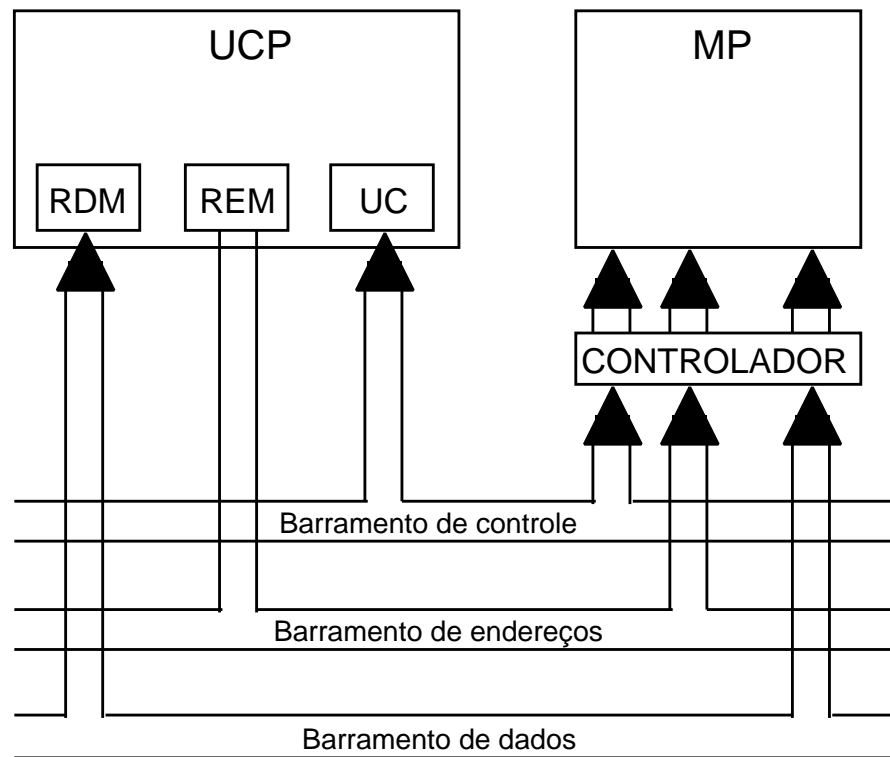
(Fig. 5.10 do livro texto)



# Memória Principal

## Operações com a MP

- *Escrita* - armazena informação na memória
- *Leitura* - recupera uma informação na memória



(Fig. 5.11 do livro texto)

# Memória Principal

## Operações com a MP

- *Registrador de Dados de Memória (RDM)*: armazena a informação que está sendo transferida da/para memória para/da UCP (leitura/escrita)
- *Barramento de Dados*: interliga o RDM à MP para transferência de informações (dados ou instruções)
- *Registrador de Endereços de Memória (REM)*: armazena um endereço de memória
- *Barramento de Endereços*: interliga o REM à MP para transferência de endereço (unidirecional)
- *Barramento de Controle*: sinais de controle (leitura, escrita, wait)
- *Controlador*: decodifica o endereço colocado no barramento para localizar a célula desejada. Controla processos de leitura/escrita

# Memória Principal

## Operações com a MP

*Linguagem de Transferência entre Registradores (LTR):*

$(REM) \leftarrow (CI)$  // Conteúdo de CI é copiado para REM

$(RDM) \leftarrow (MP(REM))$  // Conteúdo da célula da MP cujo endereço está em REM copiado para RDM

# Memória Principal

## Operações com a MP - leitura:

1)  $(REM) \leftarrow (\text{outro registrador da UCP})$

1ª) O endereço é colocado no barramento de endereços

2) Sinal de leitura é colocado no barramento de controle

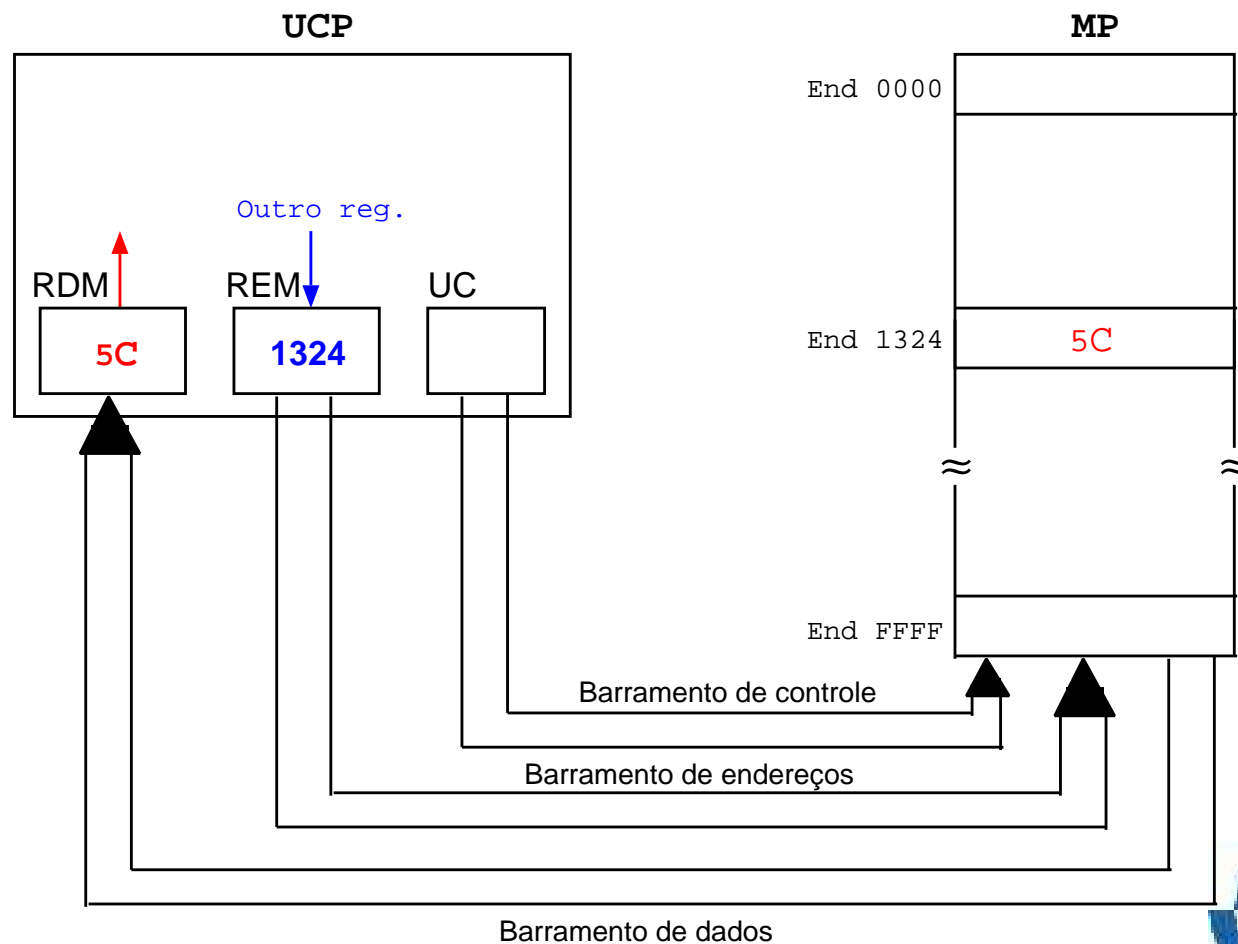
2ª) Decodificação do endereço e localização da célula

3)  $(RDM) \leftarrow (MP(REM))$  pelo barramento de dados

4)  $(\text{outro registrador da UCP}) \leftarrow (RDM)$

# Memória Principal

## Operações com a MP - leitura:

[Exemplo](#)[Voltar](#)

# Memória Principal

## Operações com a MP - escrita:

1)  $(REM) \leftarrow (\text{outro registrador})$

1ª) O endereço é colocado no barramento de endereços

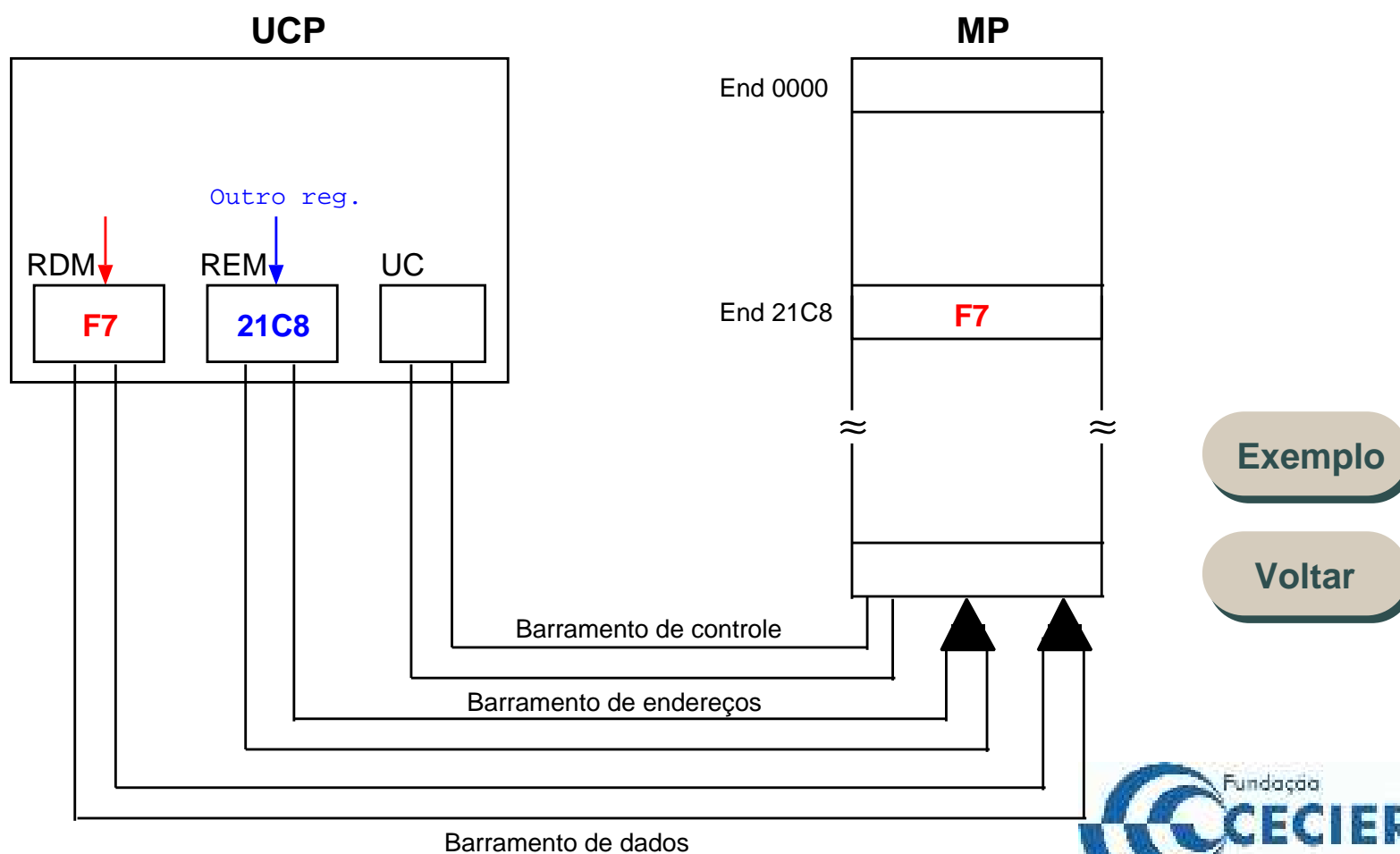
2)  $(RDM) \leftarrow (\text{outro registrador})$

3) Sinal de escrita é colocado no barramento de controle

4)  $(MP(REM)) \leftarrow (RDM)$

# Memória Principal

## Operações com a MP - escrita:



(Fig. 5.13 do livro texto)

# Memória Principal

## Capacidade de MP - cálculos:

*Quantidade de informações que podem ser armazenadas.  
Usualmente, mede-se em função da quantidade de bytes:*

- 1 Kbyte =  $2^{10}$  bytes (Kilo)
- 1 Mbyte =  $2^{20}$  bytes (Mega)
- 1 Gbyte =  $2^{30}$  bytes (Giga)
- 1 Tbyte =  $2^{40}$  bytes (Tera)
- 1 Pbyte =  $2^{50}$  bytes (Peta)



# Memória Principal

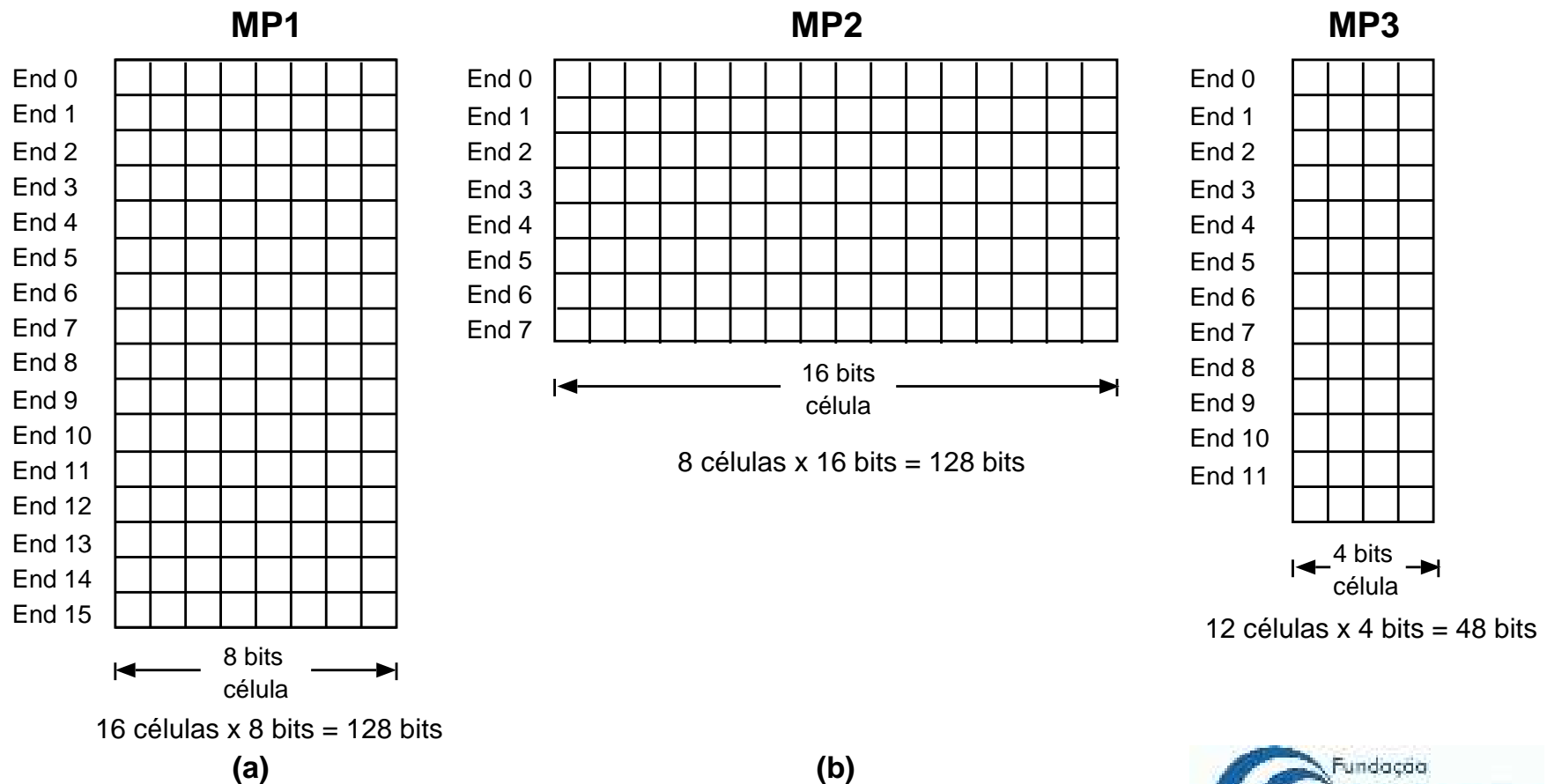
## Capacidade de MP - cálculos:

*Considere uma memória com  $N$  células, cada uma com  $M$  bits:*

- *Pode-se armazenar em cada célula valores de 0 até  $2^M - 1$*
- *Sendo  $E$  o número de bits de um endereço,  $N = 2^E$*
- *O total de bits  $T = N \times M$*

# Memória Principal

## Capacidade de MP - cálculos:



(Fig. 5.14 do livro texto)

# Memória Principal

## Capacidade de MP - cálculos:

### *Exemplo 1:*

Uma memória RAM tem um espaço máximo de endereçamento de 2K. Cada célula pode armazenar 16 bits. Qual o valor total de bits que pode ser armazenado na memória e qual o tamanho de cada endereço?

# Memória Principal

## Capacidade de MP - cálculos:

*Solução:*

- Espaço de endereçamento  $N = 2K = 2 \times 2^{10} = 2^{11}$
- Para endereçar  $N$  células são necessários  $E$  bits, onde  $N = 2^E$ .
- Portanto, o tamanho de cada endereço é 11 bits.
- A memória tem  $2K$  endereços e cada um corresponde a uma célula de 16 bits.
- Portanto, o total  $T$  de bits que a memória pode armazenar é :  
 $2K \times 16 = 32 \text{ K bits}$

# Memória Principal

## Capacidade de MP - cálculos:

### *Exemplo 2:*

*Uma memória RAM é fabricada com a possibilidade de armazenar um máximo de 256 K bits. Cada célula pode armazenar 8 bits. Qual é o tamanho de cada endereço e qual é o total de células que podem ser utilizadas naquela RAM?*

# Memória Principal

## Capacidade de MP - cálculos:

*Solução:*

- *Total de bits =  $T = 256 K = 2^8 \times 2^{10} = 2^{18}$*
- *1 célula = 8 bits.  $M = 8 = 2^3$*
- *Sendo  $T = N \times M$ , então  $N = T/M = 256K/8 = 32 K = 2^{15}$*
- *$N = 2^E = 2^{15}$ .  $E = 15$*
- *Portanto, o tamanho de cada endereço é 15 bits e o total de células é 32 K*

# Memória Principal

## Capacidade de MP - cálculos:

### *Exemplo 3:*

*Um computador, cuja memória RAM tem uma capacidade máxima de armazenamento de 2K palavras de 16 bits cada, possui um REM e um RDM. Qual o tamanho destes registradores? Qual o valor do maior endereço dessa MP e qual a quantidade de bits que nela pode ser armazenada?*

# Memória Principal

## Capacidade de MP - cálculos:

*Solução:*

- *Total de endereços =  $N = 2K = 2^{11} = 2^E$ . Logo  $E = 11$  bits*
- *REM deve ter um tamanho de 11 bits, pois guarda endereços*
- *Se a palavra (M) tem 16 bits, RDM deve ter o mesmo tamanho, pois guarda dados*
- *Total de bits =  $T = N \times M = 2K \times 16 = 32K$  bits*
- *Como o total de endereços é  $2K$ , o maior endereço é  $2K - 1 = 2047$*



# Memória Principal

## Capacidade de MP - cálculos:

### *Exemplo 4:*

*Um processador possui um RDM com capacidade de armazenar 32 bits e um REM com capacidade de armazenar 24 bits. Sabendo-se que em cada acesso são lidas 2 células da memória RAM e que o barramento de dados tem tamanho igual ao da palavra, pergunta-se: a) Qual é a capacidade máxima de endereçamento do microcomputador em questão? b) Qual é o tamanho total de bits que podem ser armazenados na memória RAM? c) Qual é o tamanho da palavra e de cada célula?*

# Memória Principal

## Capacidade de MP - cálculos:

*a) Qual é a capacidade máxima de endereçamento do microcomputador em questão?*

*Solução:*

- *Se REM=24 bits, que armazena o endereço, a capacidade de endereçamento é  $2^{24}$*
- *$2^{24} = 16 \text{ M}$  endereços ou células*

# Memória Principal

## Capacidade de MP- cálculos:

*b) Qual é o total máximo de bits que podem ser armazenados na memória RAM?*

*Solução:*

- *Total de bits =  $T = N \times M$  , onde  $M = 1$  célula e  $N = \text{Total de células} = 16 \text{ M} = 2^{24}$*
- *Como o RDM = 32 bits guarda 2 células*
- *1 célula = 16 bits*
- *$T = 16 \times 16 \text{ M} = 256 \text{ M bits}$*

# Memória Principal

## Capacidade de MP- cálculos:

*c) Qual é o tamanho da palavra e de cada célula da máquina?*

*Solução:*

- *Tamanho da palavra é igual ao  $BD = RDM = 32$  bits*
- *O tamanho de cada célula é 16 bits (em cada acesso são lidas 2 células)*

# Memória Principal

## Capacidade de MP - cálculos:

### *Exemplo 5:*

*Um processador possui um BE com capacidade de permitir a transferência de 33 bits de cada vez. Sabe-se que o BD permite a transferência de 4 palavras em cada acesso e que cada célula da memória RAM armazena um oitavo de cada palavra. Considerando que a memória pode armazenar um máximo de 64 Gbits, pergunta-se:*

*a) Qual é a quantidade máxima de células que podem ser armazenadas na memória? b) Qual é o tamanho do REM e BD existentes neste processador? c) Qual é o tamanho de cada célula e da palavra desta máquina?*

# Memória Principal

## Capacidade de MP - cálculos:

*a) Qual é a quantidade máxima de células que podem ser armazenadas na memória RAM?*

*Solução:*

- *Se REM=33 bits, que armazena o endereço, a capacidade de endereçamento é  $2^{33}$*
- *$2^{33} = 8 \text{ G}$  endereços ou células*

# Memória Principal

## Capacidade de MP - cálculos:

*b) Qual é o tamanho do REM e BD existentes neste processador?*

*Solução:*

- Como Total de bits =  $T = N \times M$ , onde  $M = 1$  célula, então  $M = T/N$  ou  $M = 64\text{ G} / 8\text{ G} = 8\text{ bits}$
- Como palavra = 8 células, palavra = 64 bits
- Como BD transfere 4 palavras, BD = 256 bits
- REM = BE = 33 bits

# Memória Principal

## Capacidade de MP - cálculos:

*c) Qual é o tamanho de cada célula e palavra desta máquina?*

*Solução:*

- *Como Total de bits =  $T = N \times M$ , onde  $M = 1$  célula, então  $M = T/N$  ou  $M = 64\text{ G} / 8\text{ G} = 8\text{ bits}$*
- *Como palavra = 8 células, palavra = 64 bits*

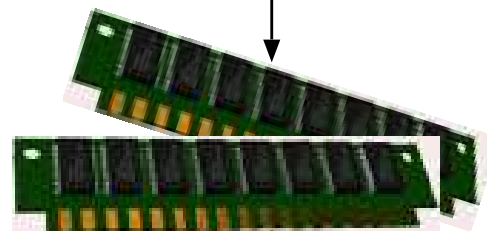


# Memória Principal

## Tipos e Nomenclatura de MP



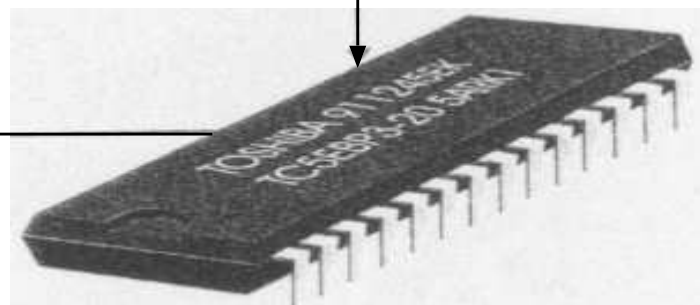
Disco rígido (HD)



Memória principal (RAM)



Processador (UCP) registradores



Memória cache

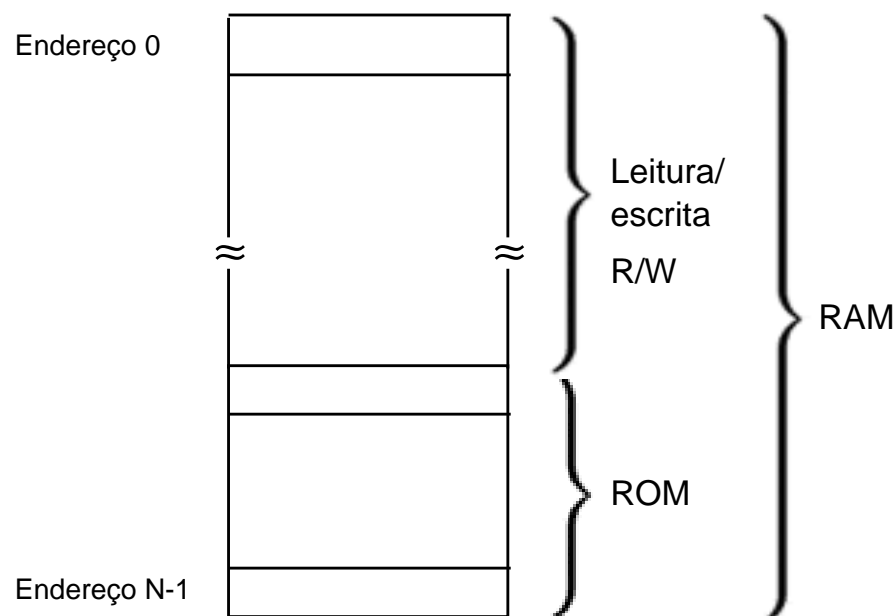
# Memória Principal

## Tipos e Nomenclatura de MP

- *MP é popularmente denominada Memória RAM (random access memory)*
- *Variações:*
  - *SRAM - Static RAM - mais cara, mais rápida, usada na fabricação de cache*
  - *DRAM - Dynamic RAM - usado na fabricação de MP tradicional, diversos fabricantes com muitas nuances*
  - *ROM - Read only memory - não volátil - programa de bootstrap, boot ou IPL - Initial Program Load*

# Memória Principal

## Tipos e Nomenclatura de MP



(Fig. 5.16(a) do livro texto)

# Memória Principal

## Memória do Tipo ROM

### *Objetivos:*

- *Ter desempenho semelhante ao das memórias R/W de semicondutores*
- *Não ser volátil*
- *Ter segurança, permitir apenas leitura de seu conteúdo por determinados programas. Há determinados programas críticos que não gostaríamos de ver infectados por vírus.*

# Memória Principal

## Memória do Tipo ROM

### *Aplicações:*

- *Guardar conjunto de rotinas básicas do Sistema Operacional.  
Por exemplo: em microcomputadores, sistema básico de entrada e saída- BIOS*
- *Sistemas de controle de processos, como forno de microondas, videogames, sistemas de injeção eletrônica*

# Erros

- *A memória principal (MP) utiliza um meio de transmissão (barramento de dados) para trânsito da informação entre MP e a UCP*
- *Esse trânsito sofre interferências que podem alterar o valor de 1 ou mais bits (de 0 para 1 ou de 1 para 0)*

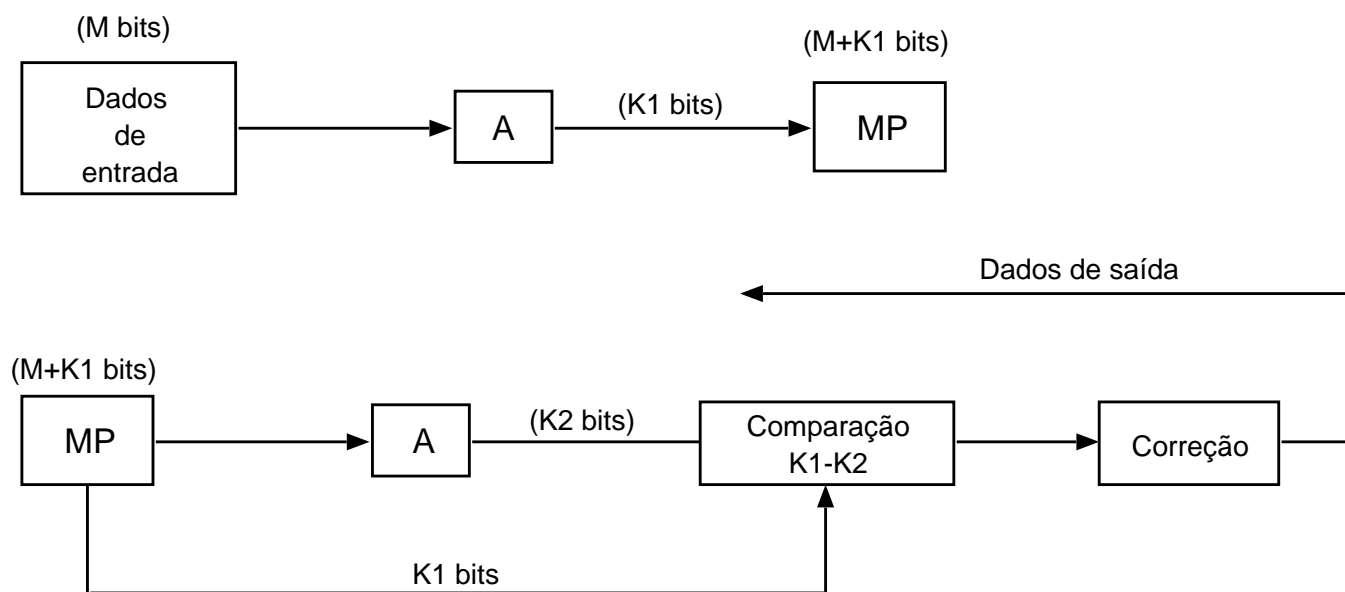
# Erros

## *Processo básico de detecção e correção de erros:*

- *Grupos de  $M$  bits a serem gravados nas células da MP sofrem um processamento, segundo um algoritmo  $A$  e produz como resultado  $K + M$  bits*
- *Serão gravados em células com capacidade para armazenar  $M+K$  bits*
- *Ao ser recuperada uma determinada célula, o sistema de detecção é acionado, o mesmo algoritmo  $A$  é executado sobre os  $M$  bits obtendo-se um novo conjunto  $K_2$*
- *Conjuntos têm o mesmo valor - ausência de erros*
- *Conjuntos com valores diferentes - existência de erros*

# Erros

## *Código de Correção de Erros:*



(Fig. 5.19 do livro texto)



## Aula 2

### Professores:

Lúcia M. A. Drummond  
Simone de Lima Martins

### Conteúdo:

#### Subsistemas de memória

- Memória Cache
- Detalhes

# Memória Cache

## Diferença de velocidade UCP/MP

- MP transfere bits para a UCP em velocidades sempre inferiores às que a UCP pode receber e operar os dados - tempo de espera na UCP (wait state)
- Difícil de resolver. Enquanto o desempenho dos processadores dobra a cada 18 ou 24 meses, o mesmo não acontece com as memórias DRAM (aumento de 10% a cada ano)
- Tecnologia para aumentar a velocidade de MP é bem conhecida - problema: CUSTO!!

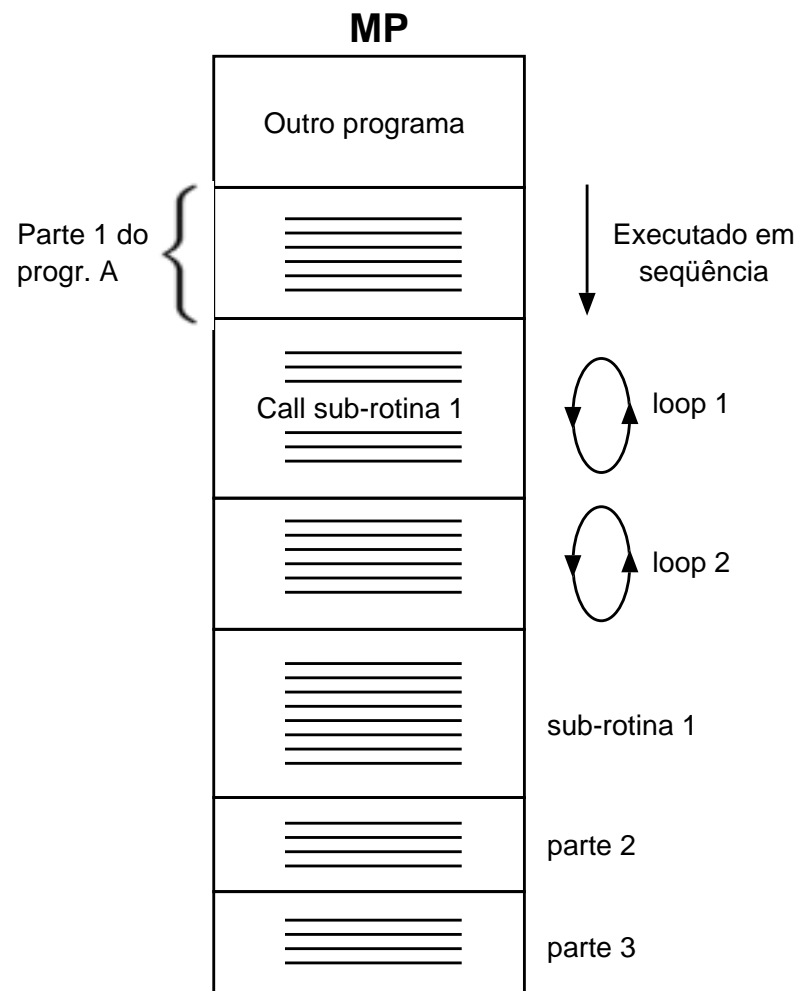
# Memória Cache

## Conceitos de Localidade

- A execução de programas se realiza, na média, em pequenos grupos de instruções.
- Duas facetas: espacial e temporal
- **Localidade espacial:** se o programa acessa uma palavra de memória, há uma boa probabilidade de que ele acesse uma palavra subsequente ou de endereço adjacente
- **Localidade temporal:** Se um programa acessa uma palavra de memória, há uma boa probabilidade de que em breve ele acesse a mesma palavra novamente

# Memória Cache

## Conceitos de Localidade

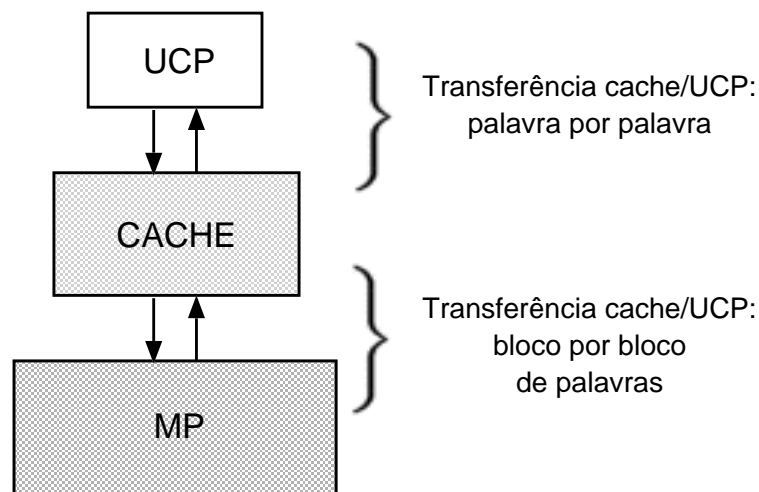


(Fig. 5.20 do livro texto)

# Memória Cache

## Utilização da Memória Cache

- Sempre que a UCP vai buscar uma nova instrução (ou dado), ela acessa a memória cache
- Se a instrução estiver na cache, chama-se de acerto (hit). Ela é transferida em alta velocidade
- Se a instrução não estiver na cache, chama-se de falta (miss).
  - O grupo de instruções a que a instrução pertence é transferida da MP para a cache, considerando o princípio da localidade

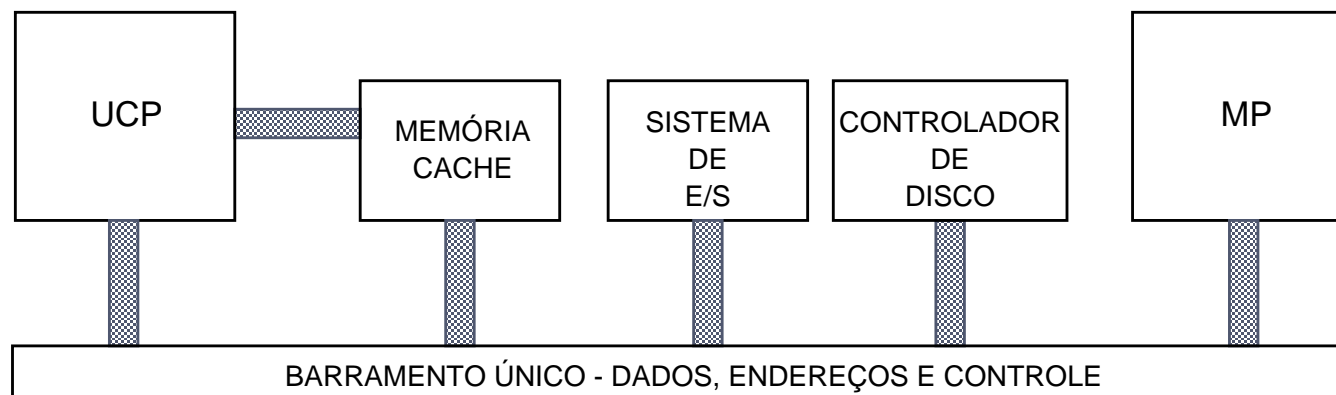


(Fig. 5.21 do livro texto)

# Memória Cache

## Utilização da Memória Cache

- Para melhorar o desempenho é necessário muito mais acertos do que faltas



(Fig. 5.22 do livro texto)

# Memória Cache

## Tipos de Memória Cache

Dois tipos básicos de emprego de cache:

- Na relação UCP/MP (cache de RAM)
- Na relação MP/disco (cache de disco)
  - Funciona segundo o mesmo princípio da cache de memória RAM porém em vez de utilizar a memória de alta velocidade SRAM para servir de cache, o sistema usa uma parte da memória principal, DRAM como se fosse um espaço em disco

# Memória Cache

## Níveis de Cache da Memória RAM

Para não aumentar muito o custo da cache, conforme o aumento da sua capacidade: sistema hierárquico de caches

- Nível 1 ou L1 sempre localizada no interior do processador
- Nível 2 ou L2 localizada em geral na placa mãe, externa ao processador
- Nível 3 ou L3 existente em poucos processadores, localizada externamente ao processador

Cache também pode ser dividida: dados e instrução



# Memória Cache

## Níveis de Cache da Memória RAM

Processadores	Fabricante	Tamanho
486	Intel	8KB, unificado
C6	Cyrix	64KB, dividido
K5	AMD	24KB, dividido
K7	AMD	128KB, dividido
Pentium	Intel	16KB, dividido
Pentium MMX	Intel	32KB, dividido
Pentium PRO	Intel	16KB, dividido
Power PC601	Motorola/IBM	32KB
Pentium III	Intel	32KB, dividido

# Memória Cache

## Elementos de Projeto de uma Memória Cache

- Definição do tamanho das memórias cache, L1 e L2
- Função de mapeamento de dados MP/cache
- Algoritmos de substituição de dados na cache
- Política de escrita pela cache

# Memória Cache

## Elementos de Projeto de uma Memória Cache

Tamanho da Memória Cache (fatores):

- Tamanho da memória principal
- Relação acertos/faltas
- Tempo de acesso da MP
- Custo médio por bit, da MP, e da memória cache L1 ou L2
- Tempo de acesso da cache L1 ou L2
- Natureza do programa em execução (princípio da localidade)

# Memória Cache

## Elementos de Projeto de uma Memória Cache

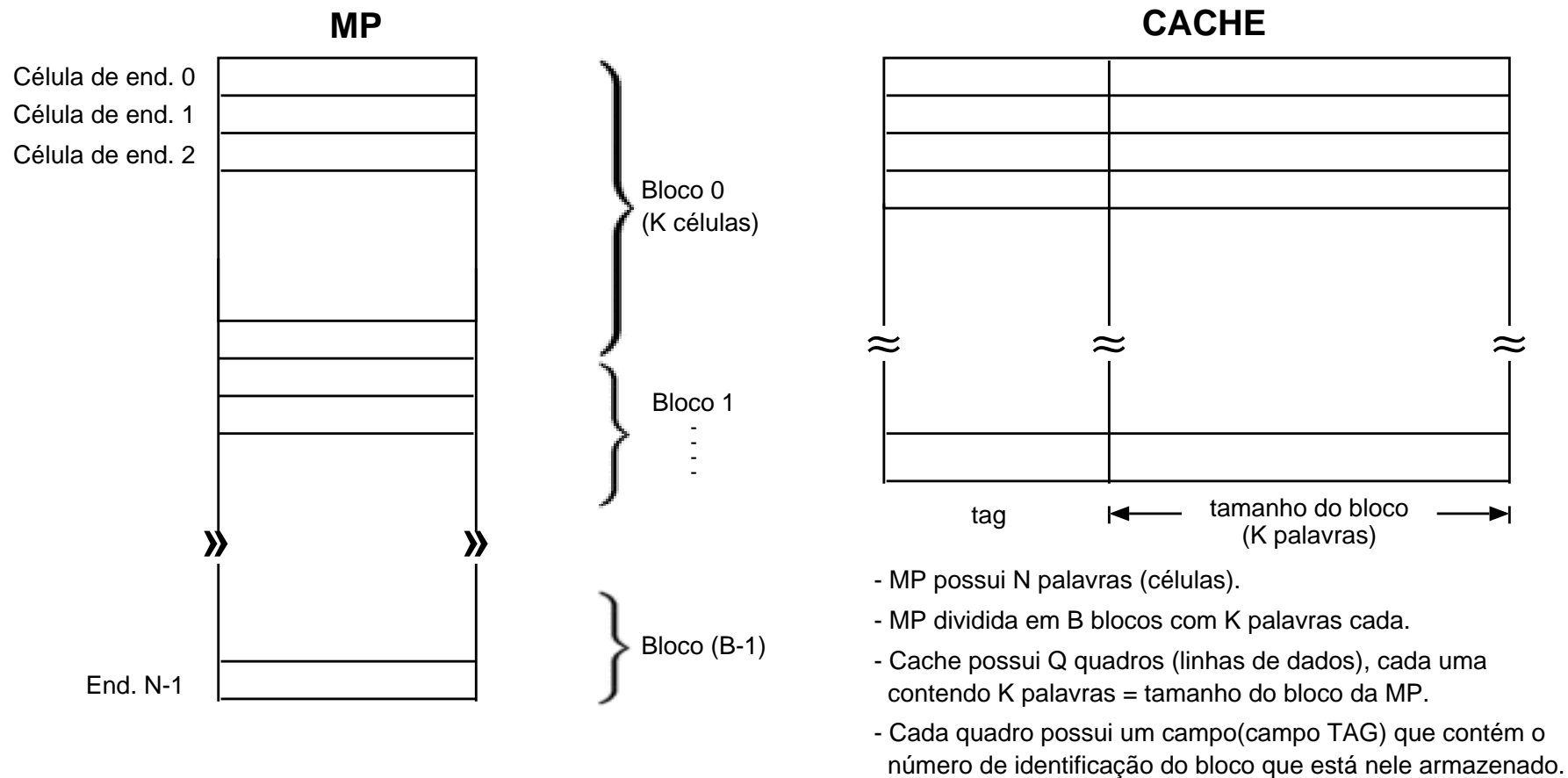
Mapeamento de dados MP/Cache:

- A memória RAM está dividida em conjuntos de  $B$  blocos, cada um com  $K$  células e a cache com  $Q$  linhas, cada uma com  $K$  células.
- $Q$  é muito menor do que  $B$
- Para garantir acerto de 90% a 95% - conceito da localidade

# Memória Cache

## Elementos de Projeto de uma Memória Cache

Mapeamento de dados MP/Cache:



(Fig. 5.23 do livro texto)

# Memória Cache

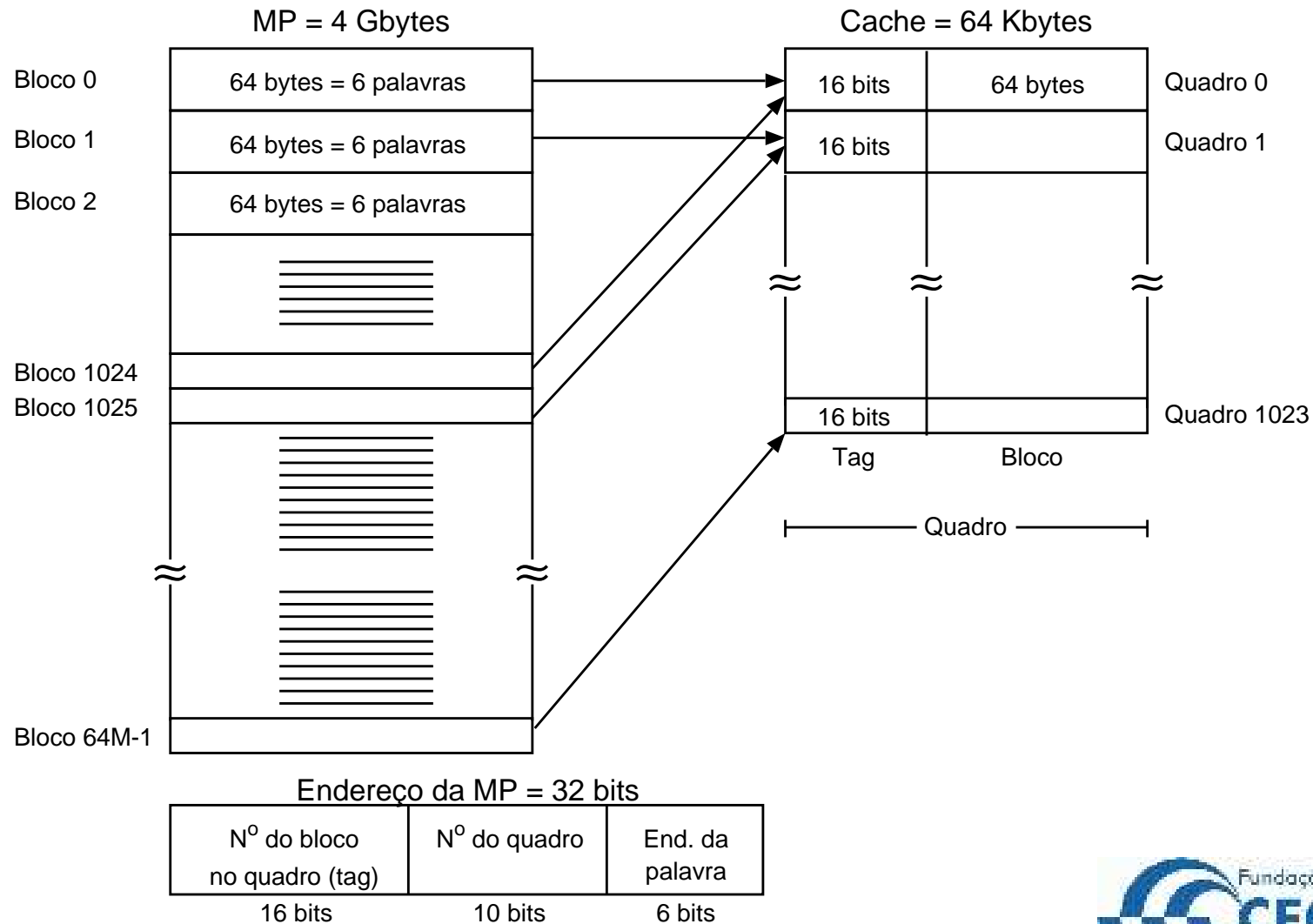
## Elementos de Projeto de uma Memória Cache

Para efetuar a transferência de um bloco da MP para uma específica linha da memória cache, escolhe-se um das 3 alternativas:

- Mapeamento Direto
- Mapeamento Associativo
- Mapeamento Associativo por conjuntos

# Memória Cache

## Mapeamento Direto



(Fig. 5.24 do livro texto)

# Memória Cache

## Mapeamento Direto

- Cada bloco da MP tem uma linha de cache
- Como há mais blocos do que linhas de cache, muitos blocos vão ser destinados a uma mesma linha



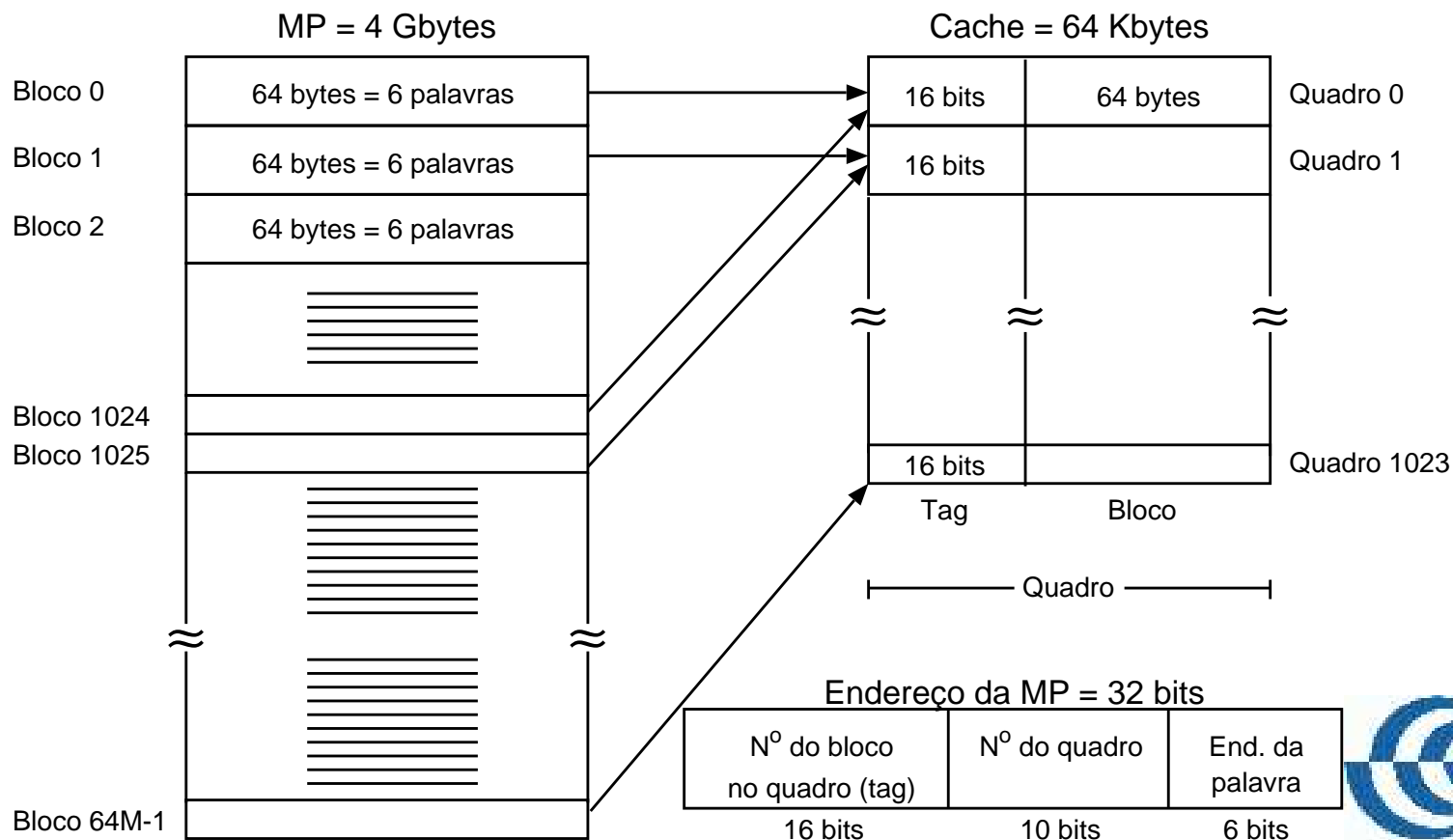


# Memória Cache

## Mapeamento Direto

Cada endereço da memória pode ser dividido da seguinte forma:

- 6 bits menos significativos: indicam a palavra ( $2^6 = 64$  palavras no bloco B e na linha Q)
- 10 bits do meio: indicam o endereço da linha da cache ( $2^{10} = 1024$  linhas)
- 16 bits mais significativos: qual o bloco dentre os 64 K blocos que podem ser alocados na linha

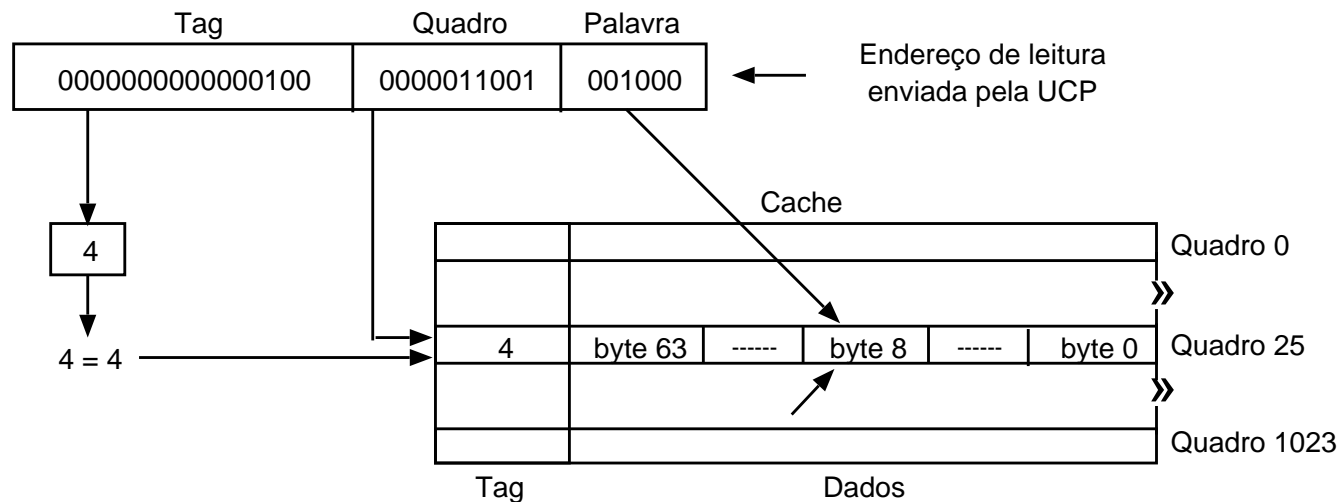


# Memória Cache

## Mapeamento Direto

### Exemplo:

- UCP apresenta endereço de 32 bits ao circuito da cache: 0000000000000010000000011001001000
- Parte 1: 00000000000000100 (comparado com o tag do quadro 25 da cache)
- Parte 2: 0000011001 (quadro 25)
- Parte 3: 001000 (palavra 8 é acessada)

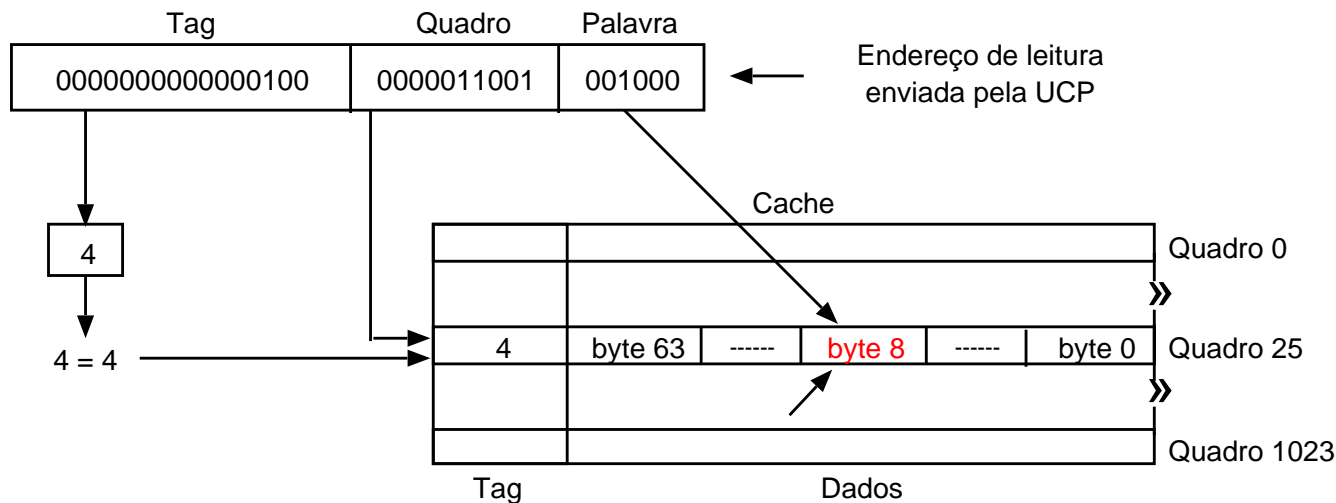


(Fig. 5.25 do livro texto)

# Memória Cache

## Mapeamento Direto

- Se os valores dos campos tag, do endereço e da linha cache não forem iguais - bloco deve ser transferido da MP para o quadro 25, substituindo o atual bloco.
- Em seguida, o byte 8 é transferido para a UCP
- 26 bits mais significativos são utilizados como o endereço do bloco desejado ( $2^{26} = 64M$ )



Mapeamento  
Direto

Voltar

# Memória Cache

## Mapeamento Direto

### Considerações:

- simples, de baixo custo, não acarreta sensíveis atrasos de processamento de endereços
- Problema: fixação da localização para os blocos (65.536 blocos destinados a uma linha)

Se durante a execução houver repetidas referências a palavras situadas em blocos alocados na mesma linha: muitas substituições de blocos

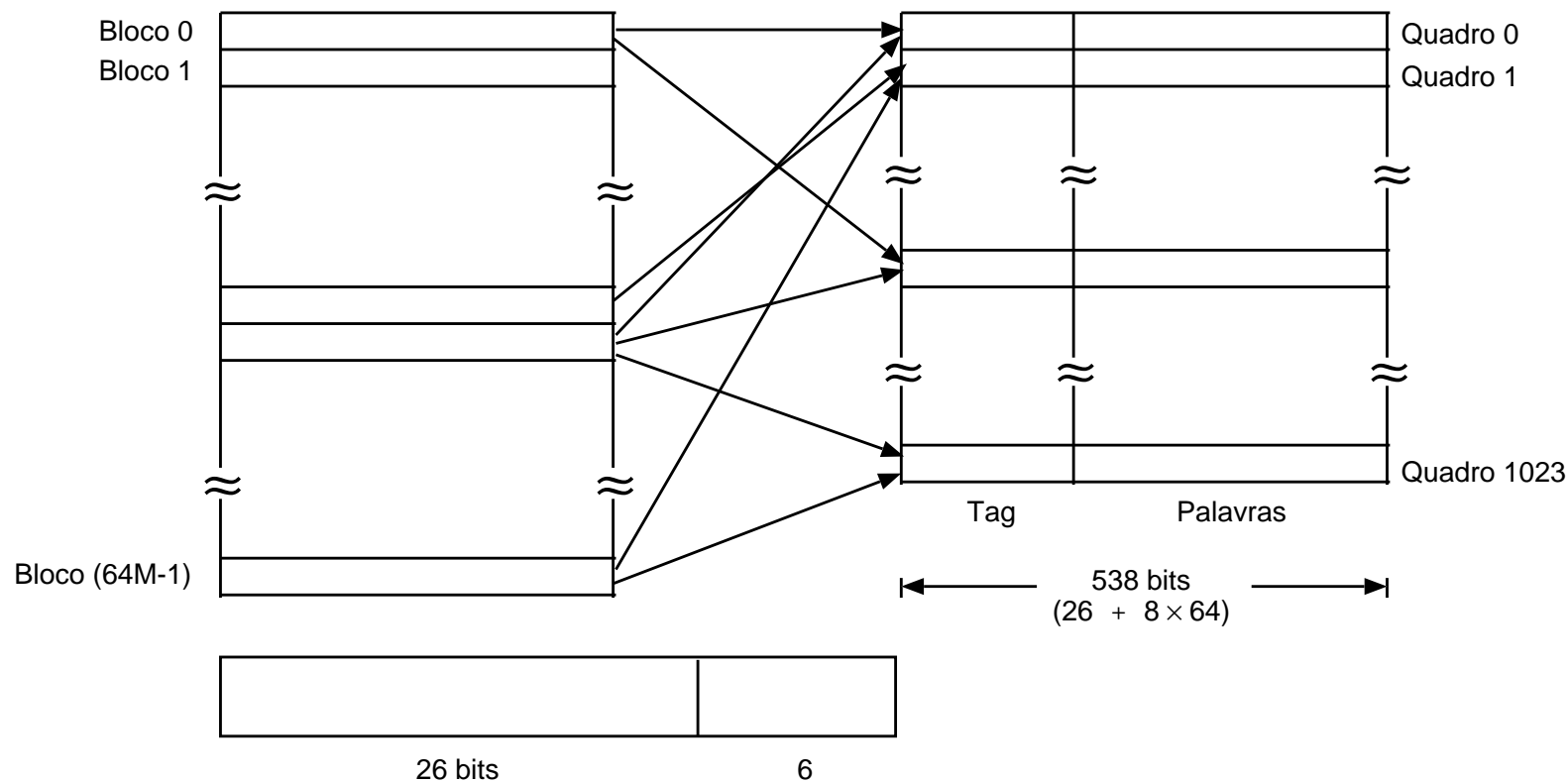
# Memória Cache

## Mapeamento Associativo

- Os blocos não têm uma linha fixada previamente para seu armazenamento
- Se for verificado que o bloco não está armazenado na cache, este será transferido, substituindo um bloco já armazenado
- Endereço da MP é dividido em duas partes:
- 6 bits menos significativos: palavra desejada
- 26 bits restantes: endereço do bloco desejado

# Memória Cache

## Mapeamento Associativo



(Fig. 5.26 do livro texto)

Sempre que a UCP realizar um acesso, o controlador da cache deve examinar e comparar os 26 bits de endereço do bloco com o valor dos 26 bits do campo de tag de todas as 1024 linhas.

# Memória Cache

## Mapeamento Associativo

### Considerações:

- Evita a fixação de blocos às linhas
- Necessidade de uma lógica complexa para examinar cada campo de tag de todas as linhas de cache



# Memória Cache

## Mapeamento Associativo por Conjuntos

- Compromisso entre as duas técnicas anteriores: tentar resolver o problema de conflito de blocos e da busca exaustiva e comparação do campo tag
- Organiza as linhas da cache em grupos, denominados conjuntos
- Nos conjuntos, as linhas são associativas

# Memória Cache

## Mapeamento Associativo por Conjuntos

A cache é dividida em C conjuntos de D linhas:

- Quantidade de linhas  $Q = C \times D$
- Endereço da linha no conjunto  $K = E \text{ módulo } C$

# Memória Cache

## Mapeamento Associativo por Conjuntos

O algoritmo estabelece que:

- O endereço da MP é dividido da seguinte forma:

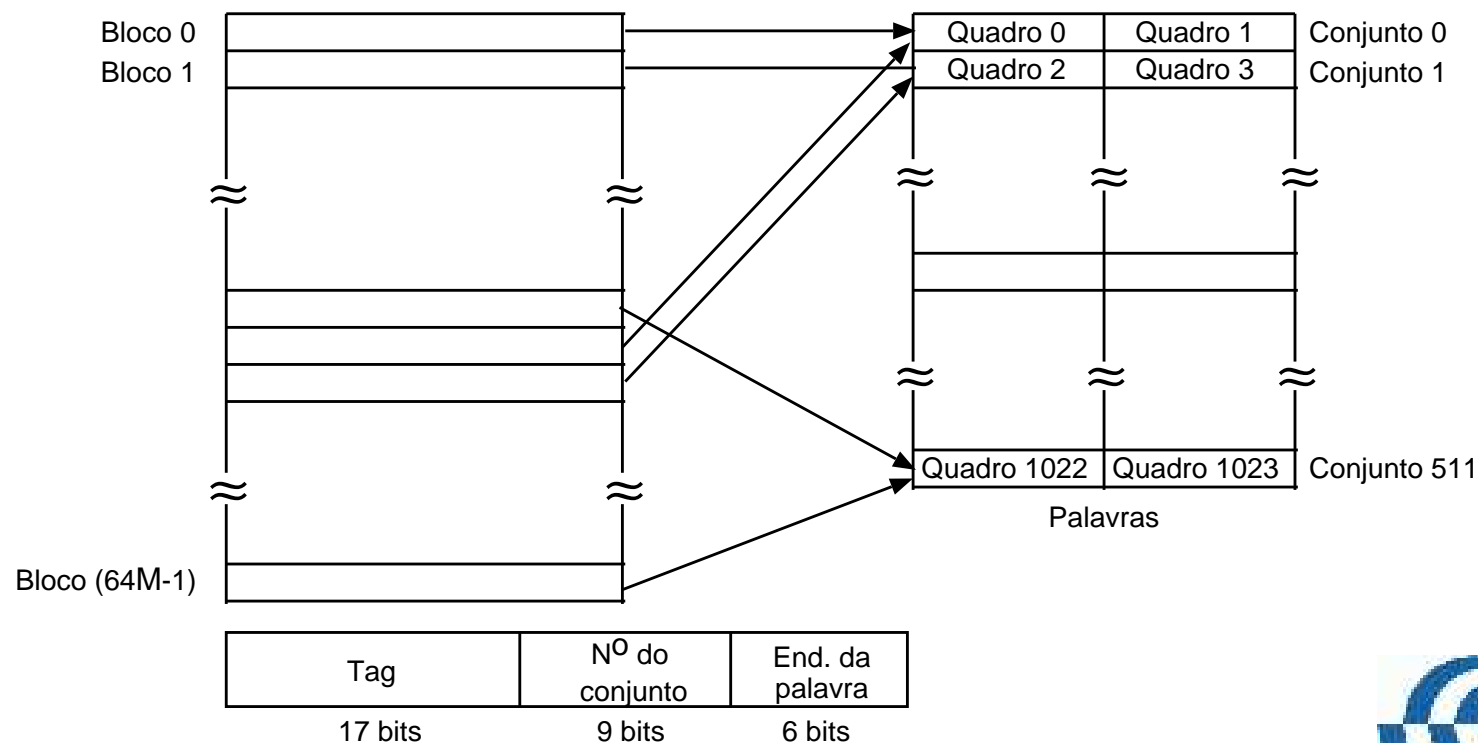
Tag	Número do conjunto	Endereço da palavra
17 bits	9 bits	6 bits

# Memória Cache

## Mapeamento Associativo por Conjuntos

O algoritmo estabelece que:

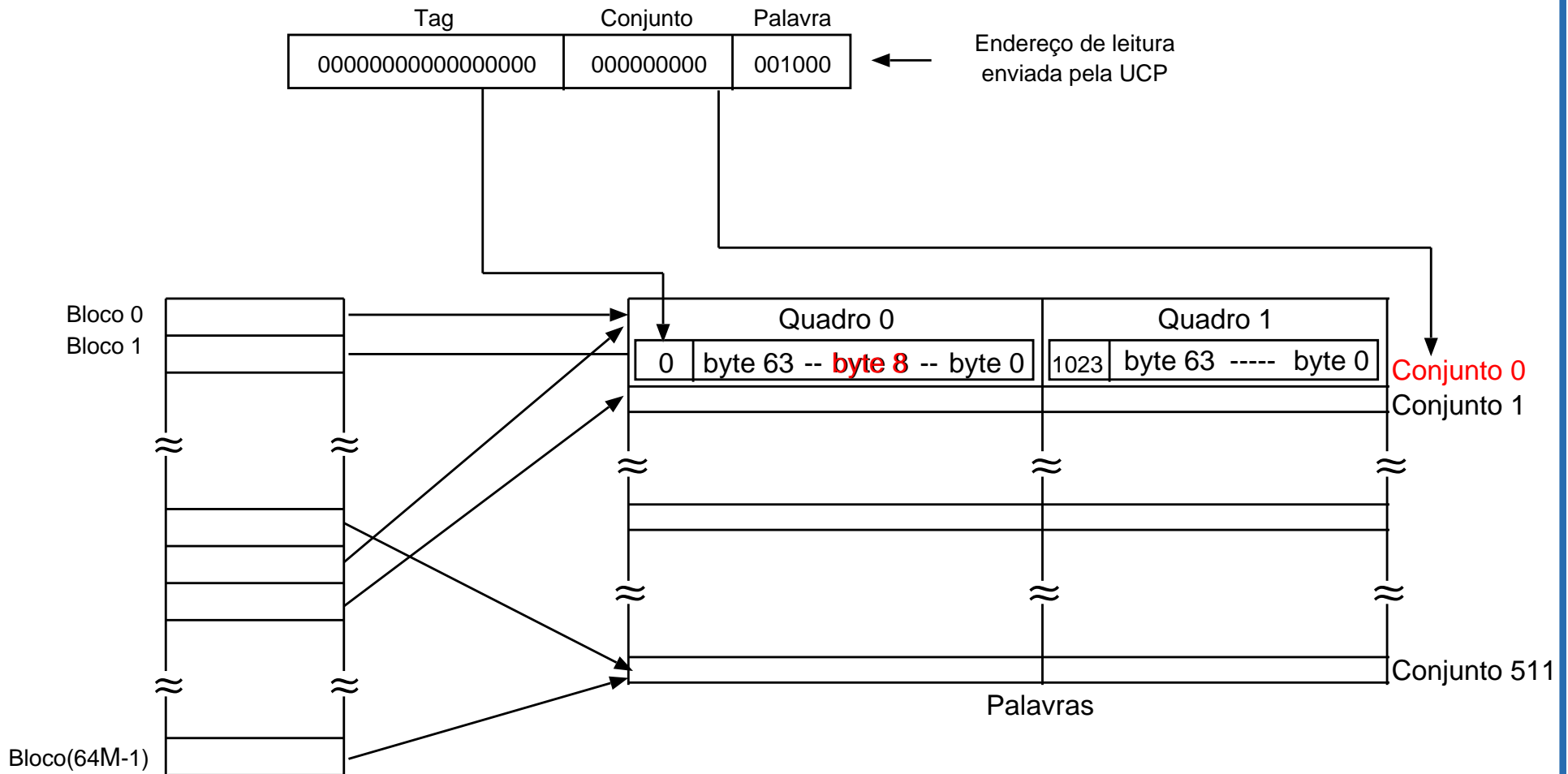
- Ao se iniciar uma operação de leitura, o controlador da cache interpreta os bits do campo de conjuntos para identificar qual o conjunto desejado.
- Em seguida, o sistema compara, no conjunto encontrado, o valor do campo tag do endereço com o valor do campo tag de cada linha do conjunto encontrado.



(Fig. 5.27 do livro texto)

# Memória Cache

## Mapeamento Associativo por Conjuntos



Exemplo

Voltar

# Memória Cache

## Algoritmos de Substituição de Dados na Cache

Definir qual dos blocos atualmente armazenados na cache deve ser retirado para dar lugar a um novo bloco que está sendo transferido (já que  $Q \ll B$ )

# Memória Cache

## Algoritmos de Substituição de Dados na Cache

Dependendo de qual técnica de mapeamento se esteja usando, pode-se ter algumas opções de algoritmos:

- Se o método de mapeamento for o direto, somente há uma única linha possível para um dado bloco
- Para os outros dois métodos - associativo e associativo por conjunto - existem várias opções

# Memória Cache

## Algoritmos de Substituição de Dados na Cache

- **LRU**: o sistema escolhe para ser substituído o bloco que está há mais tempo sem ser utilizado
- **FILA**: o primeiro a chegar é o primeiro a ser atendido. O sistema escolhe o bloco que está armazenado há mais tempo na cache.
- **LFU**: o sistema escolhe o bloco que tem tido menos acessos por parte da CPU
- **Escolha aleatória**: trata-se de escolher aleatoriamente um bloco para ser substituído



# Memória Cache

## Política de Escrita pela Memória Cache

- Toda vez que a UCP realiza uma operação de escrita, esta ocorre imediatamente na cache.
- Quando atualizar a MP?

# Memória Cache

## Política de Escrita pela Memória Cache

### Considerações:

- MP pode ser acessada tanto pela cache quanto por elementos de E/S. É possível que uma palavra da MP tenha sido alterada só na cache, ou um elemento de E/S pode ter alterado a palavra da MP e a cache esteja desatualizada
- MP pode ser acessada por várias UCP's. Uma palavra da MP é atualizada para atender à alteração de uma cache específica e as demais caches estarão desatualizadas.

# Memória Cache

## Política de Escrita pela Memória Cache

### Técnicas:

- Escrita em ambas (**write through**): cada escrita em uma palavra de cache acarreta escrita igual na palavra correspondente da MP
- Escrita somente no retorno (**write back**): atualiza a MP apenas quando o bloco for substituído e se tiver ocorrido alguma alteração na cache.  
Uso do bit ATUALIZA
- Escrita uma vez (**write once**): é uma técnica apropriada para sistemas multi UCP/cache, que compartilhem o mesmo barramento. Primeira atualização: write through + alerta os demais componentes que compartilham o barramento único.

# Memória Cache

## Política de Escrita pela Memória Cache

### Comparações:

- Com write through pode haver uma grande quantidade de escritas desnecessárias na MP
- Com write back, a MP fica desatualizada para dispositivos de E/S, por exemplo, o que os obriga a acessar o dado através da cache (problema!)
- write once é conveniente para sistemas com múltiplas UCP's

Estudos mostram que a percentagem de escritas na MP é baixa (15%), o que aponta para uma simples política write through

# Detalhes

## Localização da Célula desejada

- O processo de localização de uma determinada célula para efeito de uma operação de leitura ou escrita é o mesmo, qualquer que seja a tecnologia de fabricação de MP

# Detalhes

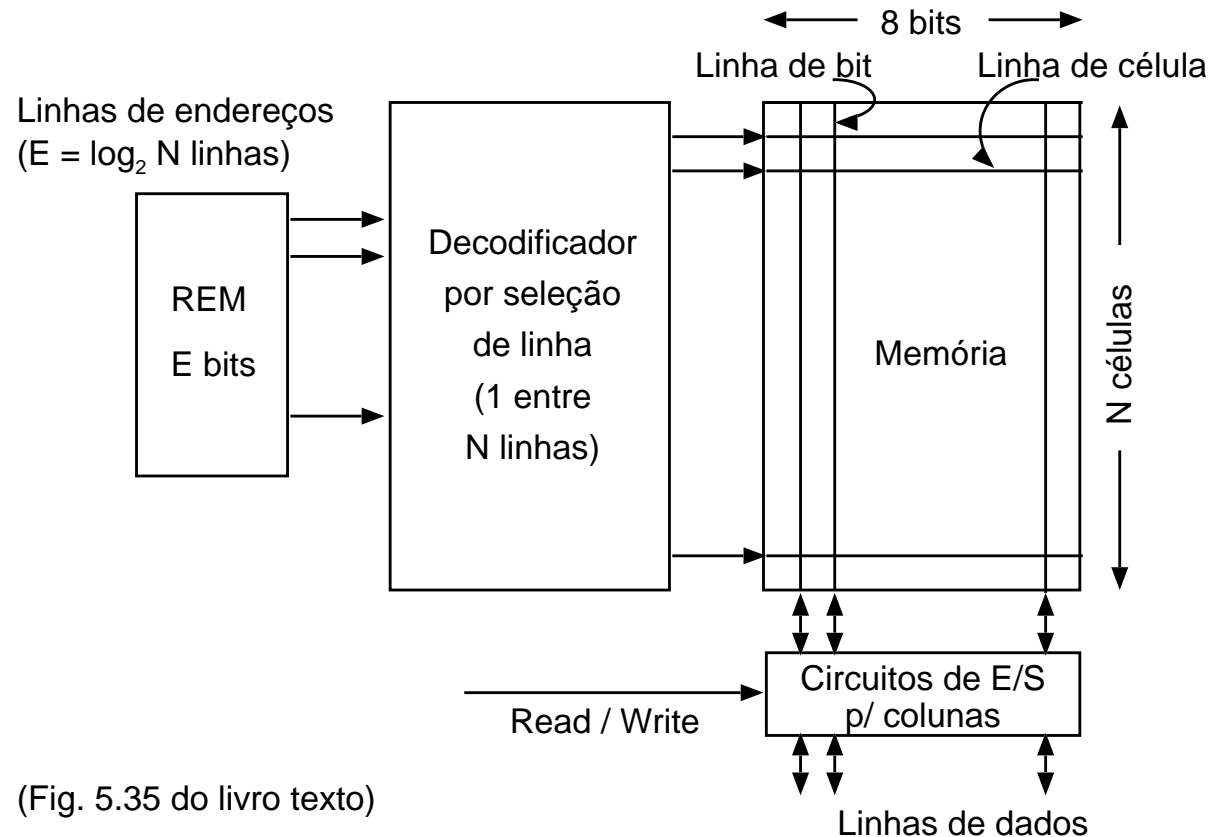
## Organização de memória

### Tipo Seleção Linear - 1 Dimensão

- Com esta técnica todos os bits de uma dada palavra estão na mesma pastilha
- Arranjo físico é igual ao arranjo lógico: o conjunto é organizado em N palavras de M bits cada
- Ex: uma pastilha de 16 K bits pode conter 1024 palavras de 16 bits cada
- Elementos de cada conjunto são conectados por linhas horizontais e verticais

# Detalhes

## Organização de memória Tipo Seleção Linear - 1 Dimensão

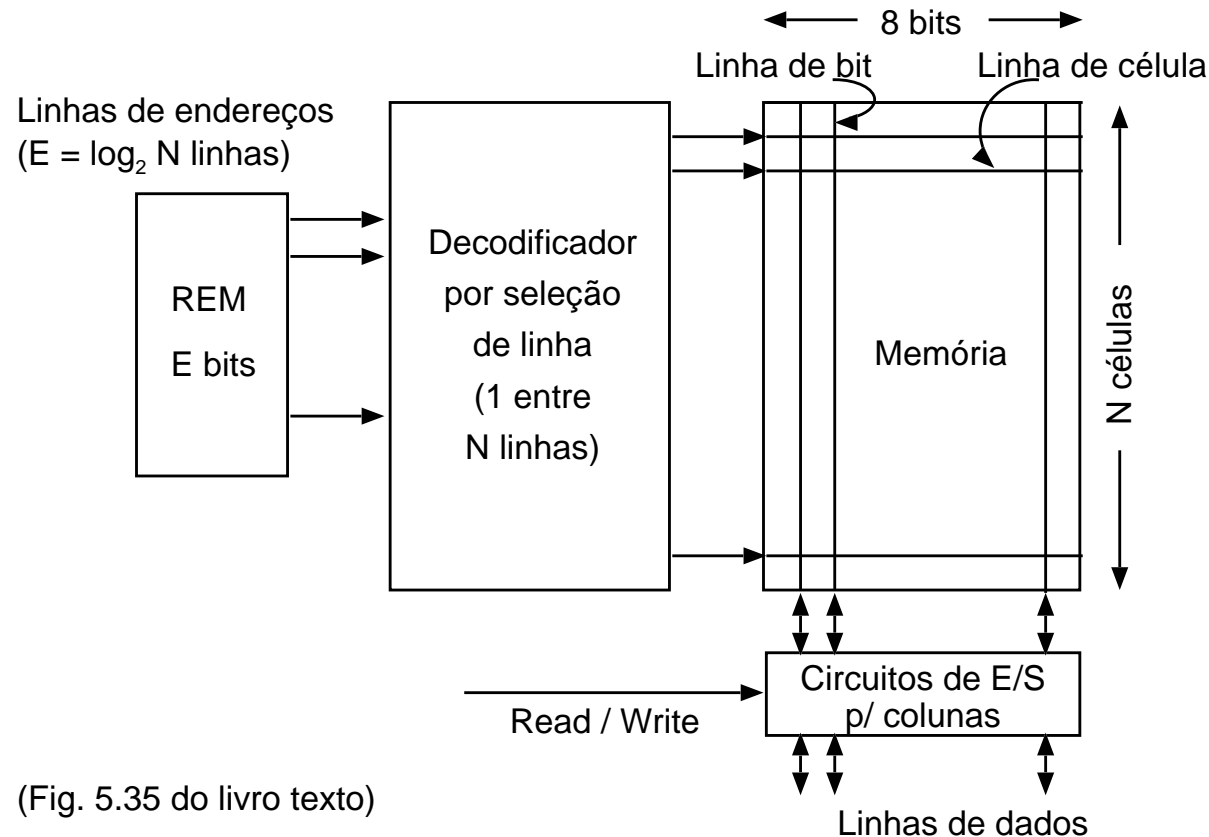


(Fig. 5.35 do livro texto)

- Cada linha horizontal da memória é uma saída do decodificador de linha e cada linha vertical da memória se conecta ao sensor de dados para receber ou enviar 1 bit de palavra
- O decodificador tem  $2^E$  saídas para E entradas, isto é, se cada endereço armazenado no REM tem E bits, a ligação REM-decodificador tem E linhas

# Detalhes

## Organização de memória Tipo Seleção Linear - 1 Dimensão



(Fig. 5.35 do livro texto)

- Ex: a pastilha de 16K bits teria um REM de 10 bits, 10 linhas de saída do REM para o decodificar e deste sairiam 1024 linhas, uma para cada célula da memória
- Tomemos, o endereço 12 decimal, armazenado no REM. Isto acarretaria uma saída 1 na 13ª linha do decodificador e as demais linhas seriam iguais zero.



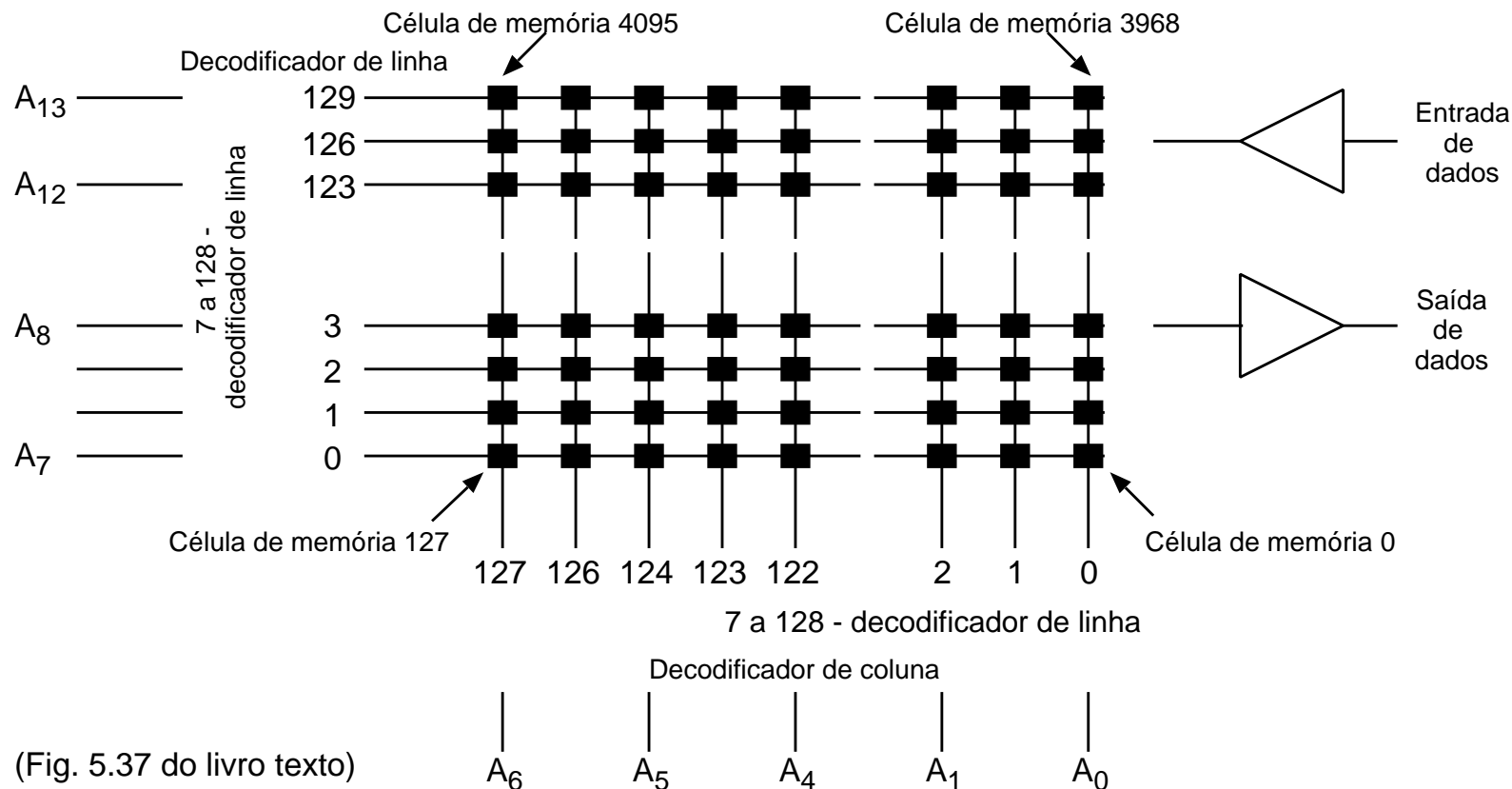
# Detalhes

## Organização de memória Tipo Matriz Linha/Coluna

- Uma memória com 16K células, onde cada endereço é um número com 14 bits, visto que  $2^{14}=16K$
- Cada pastilha possui 14 linhas de endereço, identificadas pela nomenclatura  $A_0$  a  $A_{13}$
- Divididas em duas:  $A_0$  a  $A_6$  conectadas ao decodificador de colunas,  $A_7$  a  $A_{13}$  conectadas ao decodificador de linhas

# Detalhes

## Organização de memória Tipo Matriz Linha/Coluna



(Fig. 5.37 do livro texto)

- O cruzamento de linha e coluna ativadas pelos respectivos decodificadores corresponde à célula desejada endereçada pelos 14 bits
- Ocorrem  $128 \times 128$  cruzamentos = 16K células. Uma memória com 16K células, onde cada endereço é um número com 14 bits, visto que  $2^{14} = 16K$

# Detalhes

## Organização de memória

### Comparação

- Seleção linear: número de linhas de saída muito maior, mais circuitos em uma única pastilha
- Na técnica de matriz por linhas e colunas que seleciona 1 bit de palavra por pastilha são usadas várias pastilhas mais simples para totalizar a quantidade de bits por palavra.

# Memória Cache

## Exercícios:

Capítulo 5 do livro texto.

### **Questões:**

1 até 15, 17, 20, 21, 23, 24 e 28.

# Organização de Computadores

## Professores:

Lúcia Maria de A. Drummond

Simone de Lima Martins

# Organização de Computadores

## Livro Texto:

"Introdução à Organização de Computadores"

Mário A. Monteiro

LTC editora

## Aula 3

### Professores:

Lúcia M. A. Drummond  
Simone de Lima Martins

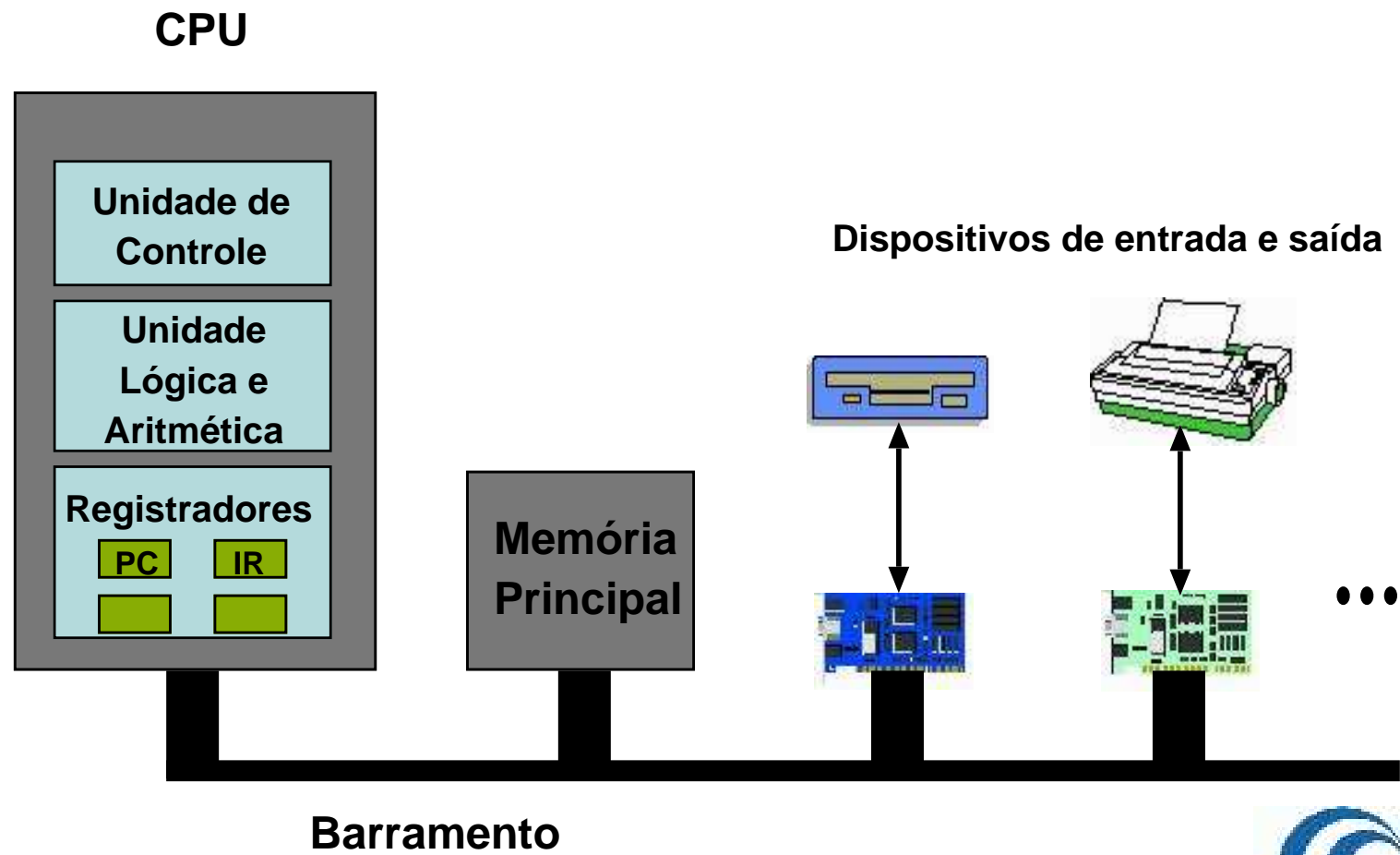
### Conteúdo:

#### Unidade Central de Processamento 1

- Introdução
- Funções básicas da UCP
- Instruções de máquina

# Introdução

## Organização de um Computador





# Introdução

- Unidade Central de Processamento
  - Responsável por computar, calcular e processar
  - Processadores atuais são fabricados em um único invólucro, denominado *chip*, contendo todos os elementos necessários à realização de suas funções

# Introdução

- Características da Unidade Central de Processamento
  - Fabricante (Intel, AMD, Cyrix)
  - Velocidade do processador (MHz)
  - Tecnologia de fabricação
  - Quantidade de transistores
  - Largura do barramento de dados e endereço
  - Capacidade máxima de memória principal

# Introdução

- Características da Unidade Central de Processamento
  - Tipo de soquete, encapsulamento e número de pinos do soquete
  - Cache (L1 e L2)
  - Processadores de inteiros
  - Processadores de ponto flutuante
  - Pipeline
  - Tamanho dos registradores (bits)

# Funções básicas da UCP

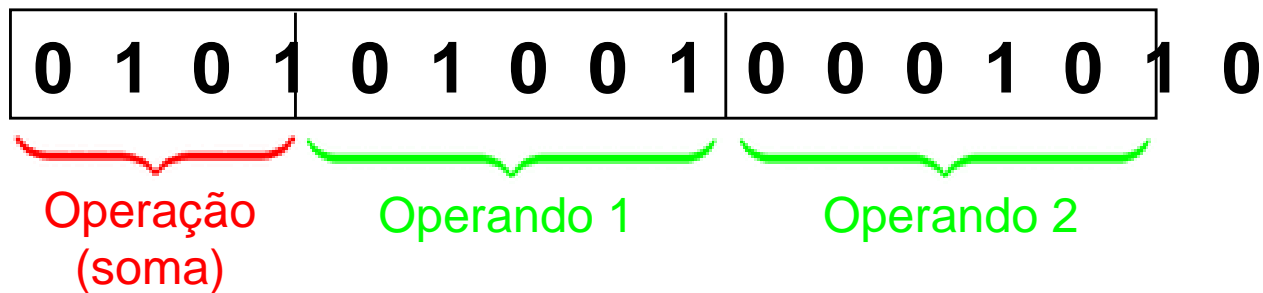
- Executar operações com dados
  - Somar ou subtrair dois números
- Controlar o funcionamento de todos os componentes do computador
  - Memória e dispositivos de entrada e saída

# Funções básicas da UCP

- Executa operações primitivas
  - Somar
  - Subtrair
  - Mover dado de um local de armazenamento para outro
  - Transferir dado para um dispositivo de saída (monitor de vídeo, por exemplo)

# Funções básicas da UCP

- Operações e a localização de dados que elas manipulam estão representados por uma seqüência de 0s e 1s (bits), denominada instrução de máquina

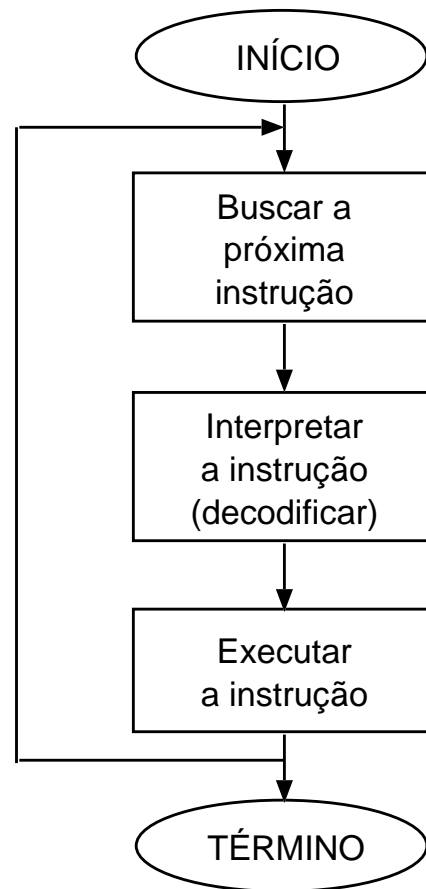


# Funções básicas da UCP

- Um programa executável é constituído de um conjunto de instruções de máquina seqüencialmente organizadas
- Para iniciar execução de um programa:
  - As instruções devem estar armazenadas em células sucessivas da memória principal
  - O endereço da primeira instrução deve estar armazenado na UCP

# Funções básicas da UCP

- A UCP executa indefinidamente o ciclo de instrução, até que o sistema seja desligado, ocorra algum erro ou seja executada uma instrução de parada

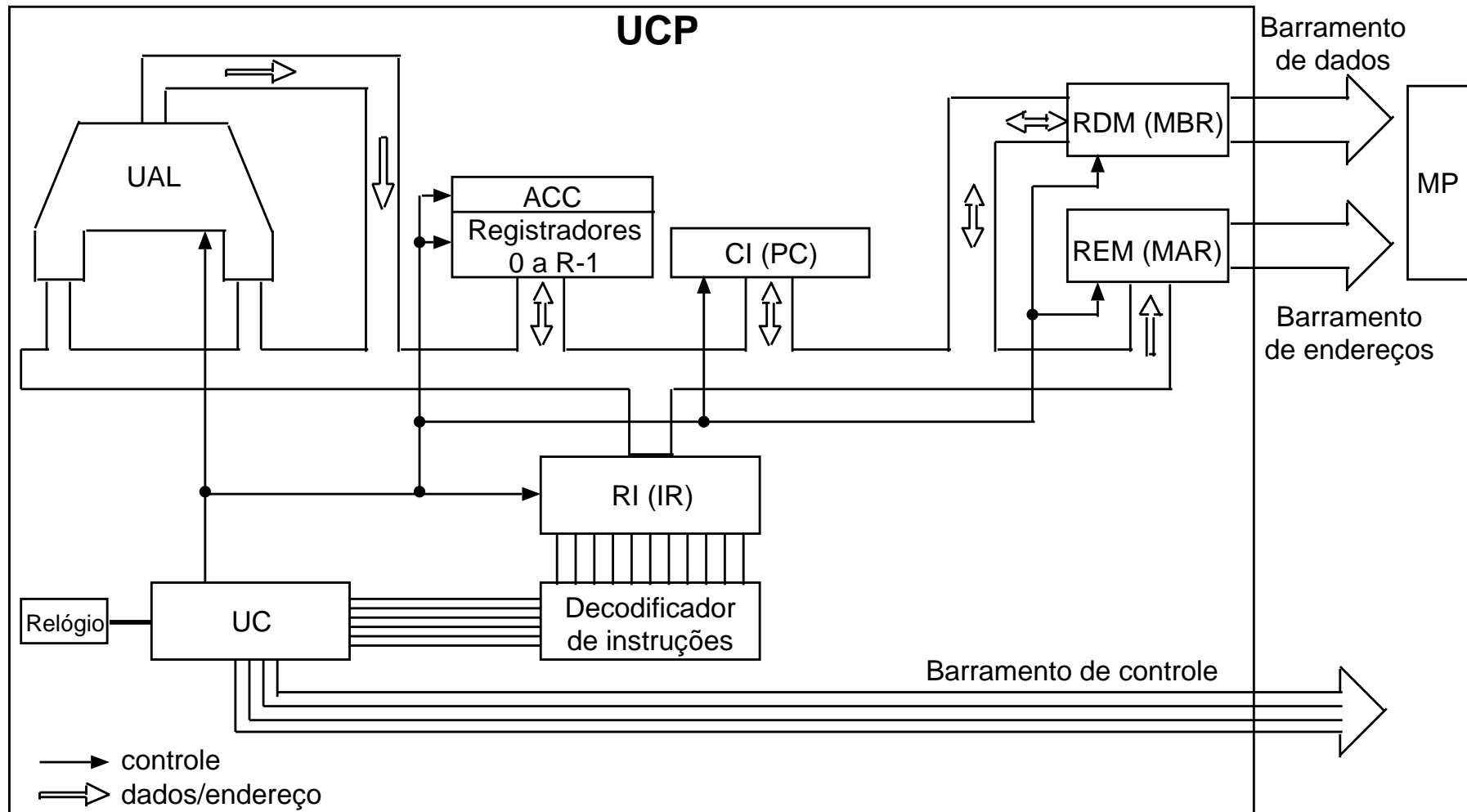


(Fig. 6.2 do livro texto)



# Funções básicas da UCP

- Organização lógica de uma UCP simples que executa instruções de forma seqüencial



(Fig. 6.3 do livro texto)

- Processadores atuais realizam várias instruções de forma simultânea (pipelining)

# Funções básicas da UCP

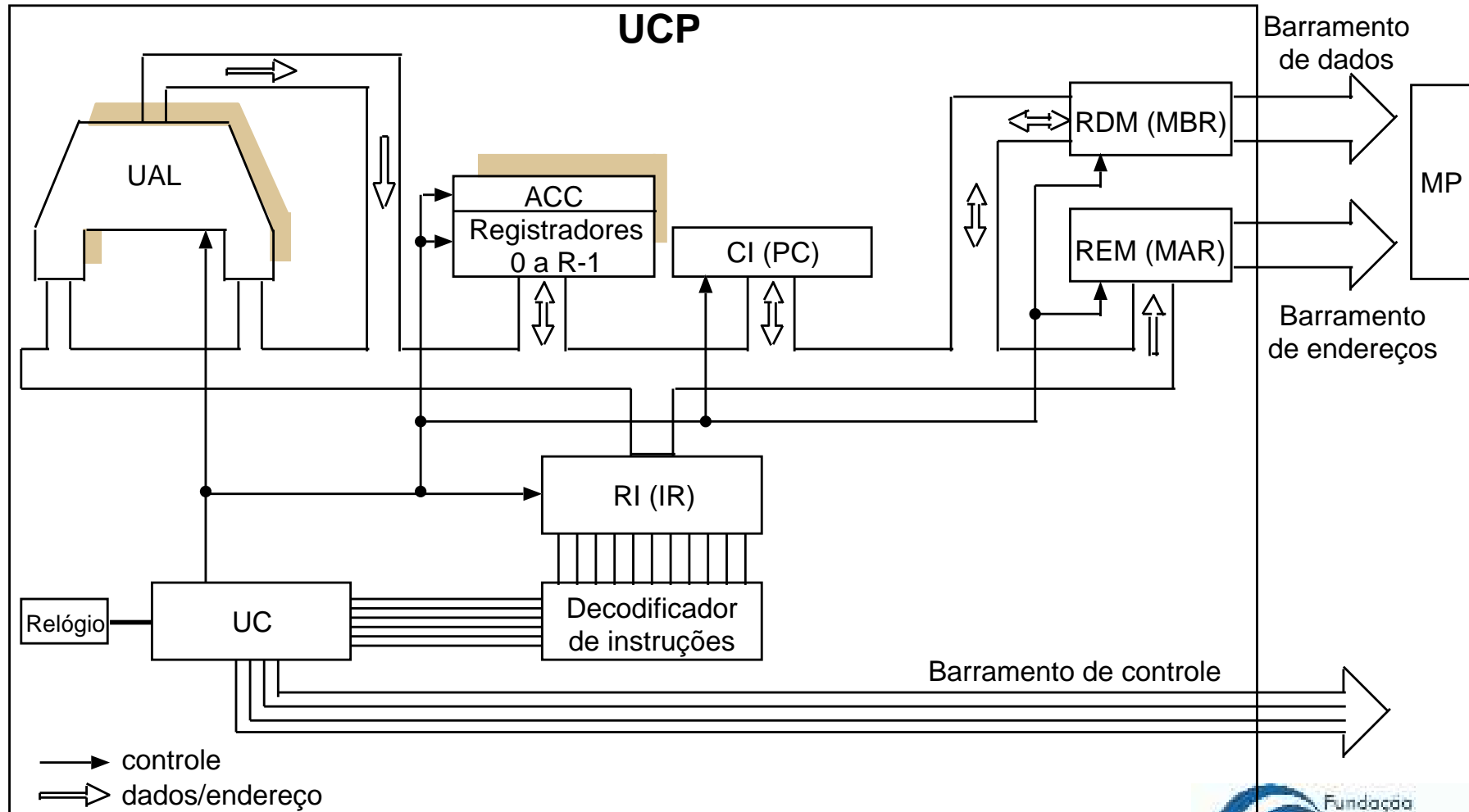
- As atividades da UCP podem ser divididas em duas grandes categorias funcionais:
  - Função processamento
  - Função controle

# Funções básicas da UCP

- Função processamento de dados
  - Operações aritméticas (somar, subtrair, multiplicar e dividir)
  - Operações lógicas (and, or, xor)
  - Movimentação de dados (memória-UCP, UCP-memória, registrador-registrador)
  - Desvios (alteração de seqüência de execução de instruções)
  - Operações de entrada e saída

# Funções básicas da UCP

- Função processamento de dados



(Fig. 6.4 do livro texto)

# Funções básicas da UCP

- Unidade Aritmética e Lógica (UAL)
  - Executa as operações com os dados
    - Operações aritméticas (soma, subtração, multiplicação, divisão)
    - Operações lógicas (and, or, xor, complemento)
    - Deslocamento à direita e à esquerda
    - Incremento e decremento
  - Unidades que tratam de números inteiros e números fracionários

# Funções básicas da UCP

- Registradores
  - UAL obtém dados de entrada dos registradores
  - Resultados da UAL são armazenados inicialmente em registradores
- UCP possui um certo número de registradores
- Arquiteturas mais antigas utilizavam o registrador acumulador (ACC) para transferência de dados com a UAL

# Funções básicas da UCP

- Arquiteturas mais modernas possuem registradores especializados no armazenamento de dados, de endereços e de segmentos
  - Dados (AH, AL, BH, BL)
  - Endereços (IP)
  - Segmentos (CS, DS)
- Em geral, capacidade de armazenamento dos registradores é igual ao tamanho da palavra do processador
  - Intel 486, palavra de 32 bits, registradores de 32 bits

# Funções básicas da UCP

- Registradores são utilizados para armazenar:
  - Dados
  - Instruções (Registrador de Instruções)
  - Endereços de instruções de um programa (Contador de Instruções)
  - Endereços de memória (REM)
  - Dados de memória (RDM)



# Funções básicas da UCP

- Alguns processadores possuem registradores especiais de estado
- Bits indicam estado do processador
- Códigos de condição do Motorola 68000

<b>N</b>	<b>Z</b>	<b>V</b>	<b>C</b>	<b>X</b>
----------	----------	----------	----------	----------

**N** - Negative FLAG - Valor 1 se o resultado de uma operação aritmética for negativo.

**Z** - Zero FLAG - Valor 1 se o resultado for igual a zero.

**V** - Overflow FLAG - Valor 1 se ocorrer overflow.

**C** - Carry FLAG - Valor 1 se ocorrer "vai 1" para fora do número.

**X** - Extend FLAG - Semelhante ao FLAG C, porém pouco usado.

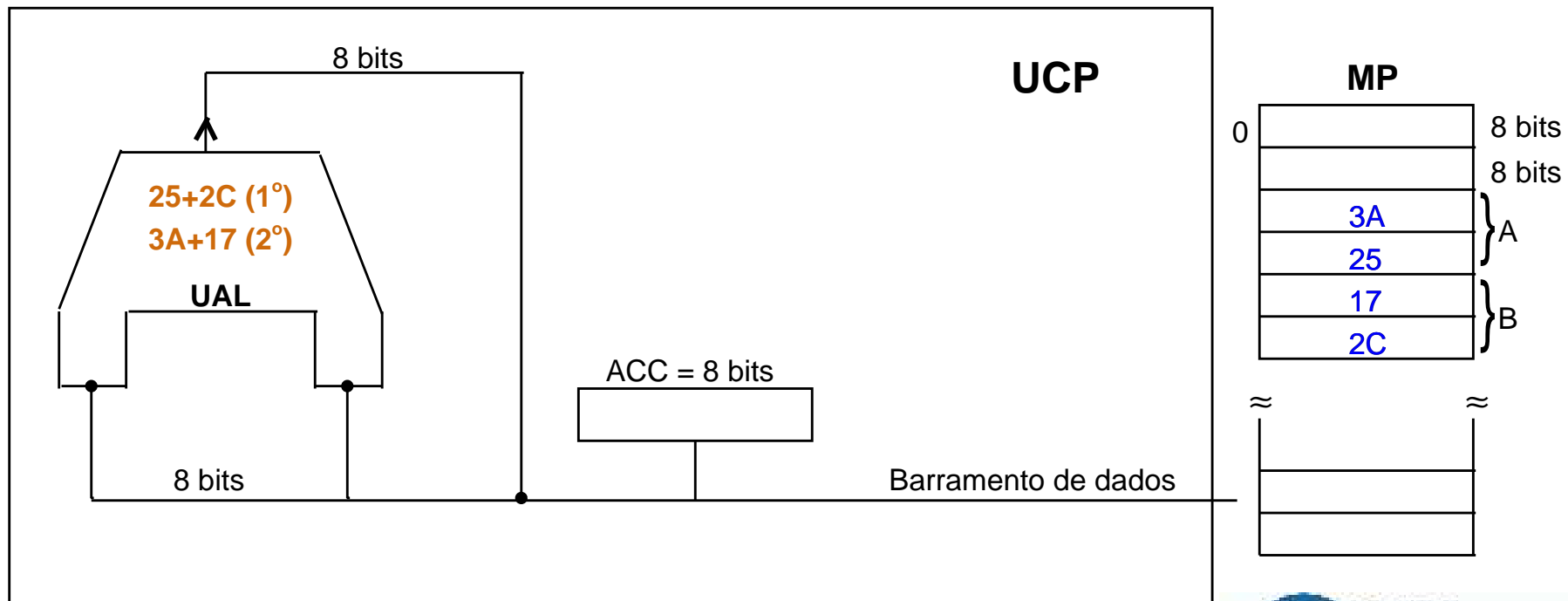
(Fig. 6.9(b) do livro texto)

# Funções básicas da UCP

- Tamanho da palavra
  - Número de bits manipulados de uma vez pela UAL
  - Capacidade de armazenamento dos registradores
- Barramentos de dados possuem tamanho mínimo igual ao tamanho da palavra
- Tamanho da palavra pode influenciar desempenho do processador

# Funções básicas da UCP

- Exemplo de influência do tamanho da palavra
  - Somar dois valores  $A=3A25$  e  $B=172C$
  - Sistema 1 possui memória principal com 64K células com 8 bits em cada uma e palavra de 8 bits
  - Tempo para efetuar a operação igual a  $T_1$



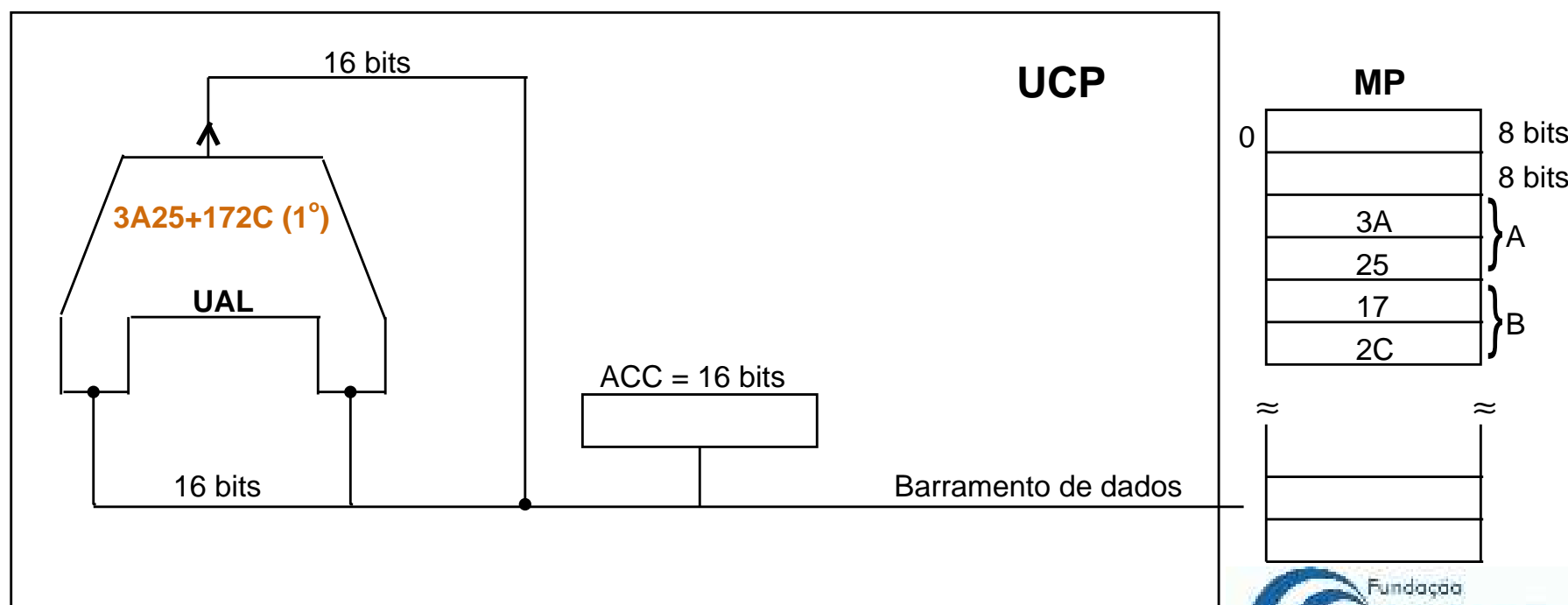
(Fig. 6.10 do livro texto)

Exemplo

Voltar

# Funções básicas da UCP

- Exemplo de influência do tamanho da palavra
  - Somar dois valores  $A=3A25$  e  $B=172C$
  - Sistema 2 possui memória principal com 1M células com 8 bits em cada uma e palavra de 16 bits
  - Tempo para efetuar a operação igual a  $T_2$



(Fig. 6.11 do livro texto)

# Funções básicas da UCP

- Exemplo de influência do tamanho da palavra
  - $T2 \approx T1/2$
- Influência do tamanho da palavra:
  - Tempo de execução das instruções
  - Barramento de dados deve ter tamanho mínimo igual ao tamanho da palavra
  - Implementação física da memória
    - Possibilidade de acessar mais de uma célula em um único ciclo de memória (células de 8 bits, palavra de 32 bits, 4 células devem ser acessadas)

# Funções básicas da UCP

- Função de controle
  - Instruções de máquina que compõem um programa em execução devem estar armazenadas seqüencialmente na memória principal e na cache
  - Como funciona uma instrução de máquina ?
  - Como a instrução é movida da memória para UCP ?
  - Onde a instrução é armazenada na UCP ?
  - Como é identificada e controlada a operação que deve ser realizada ?

# Funções básicas da UCP

- A área de controle de uma UCP é a parte funcional que realiza as atividades do ciclo de instrução
  - Ciclo de instrução = Ciclo de busca + Ciclo de execução
  - Ciclo de busca
    - Busca da instrução na memória e armazenamento em um registrador
    - Interpretação das ações a serem desencadeadas para executar a instrução
  - Ciclo de execução
    - Geração dos sinais de controle para UAL, memória ou E/S

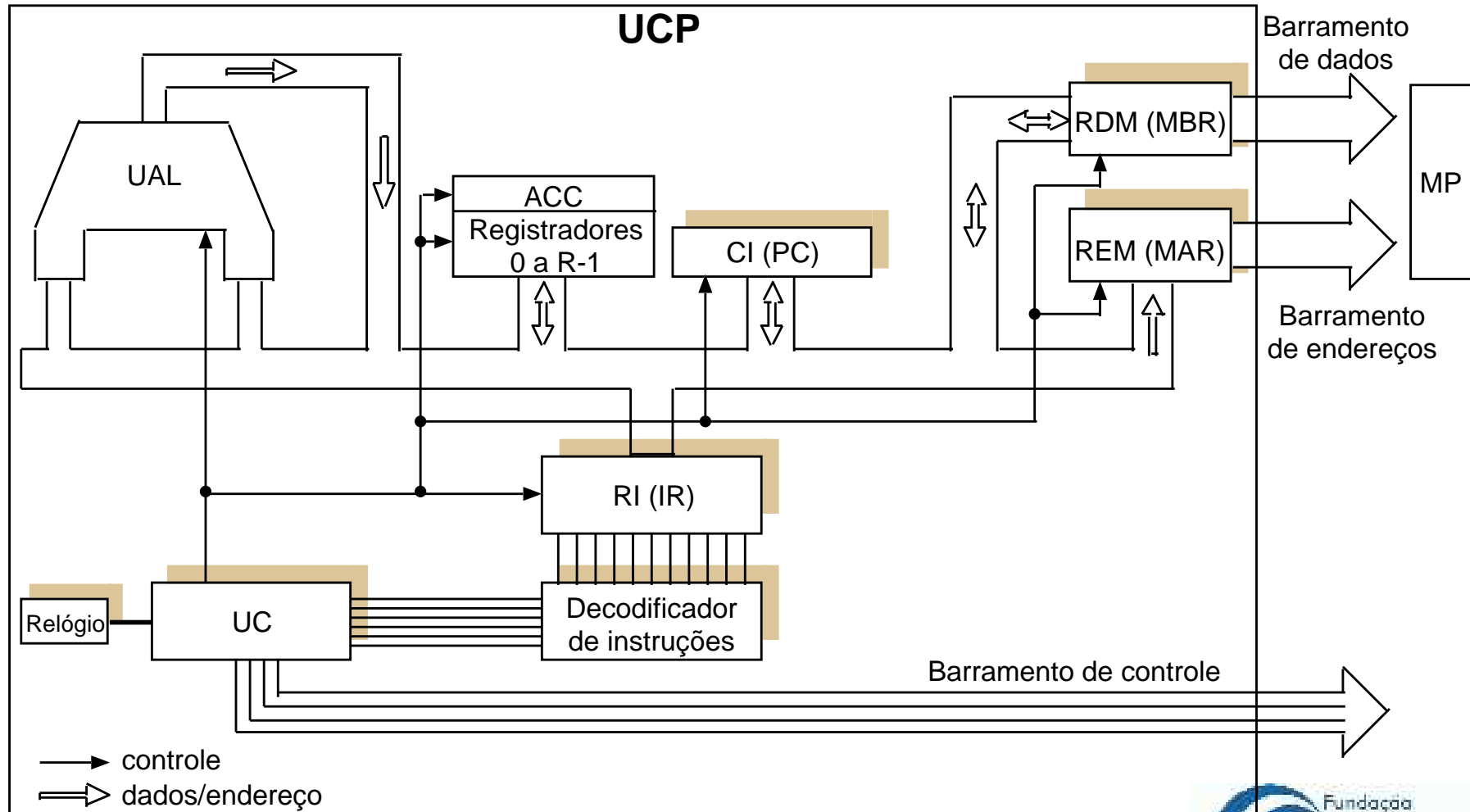
# Funções básicas da UCP

- Dispositivos básicos para realizar a função de controle
  - Unidade de controle
  - Relógio
  - Registrador de Instrução (RI)
  - Contador de Instrução (CI)
  - Decodificador
  - Registradores de endereço de memória (REM) e de dados da memória (RDM)



# Funções básicas da UCP

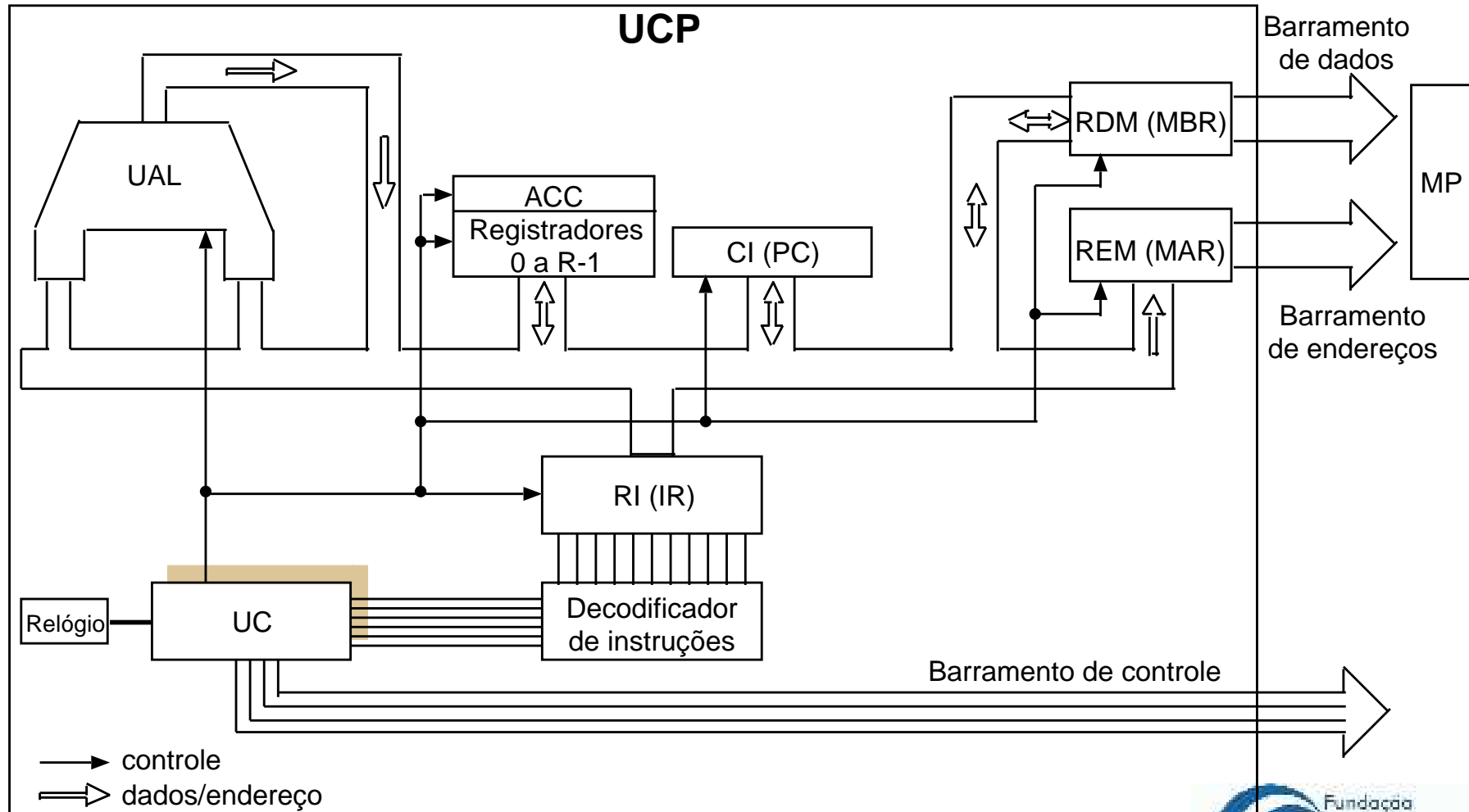
- Dispositivos básicos para realizar a função de controle



(Fig. 6.12 do livro texto)

# Funções básicas da UCP

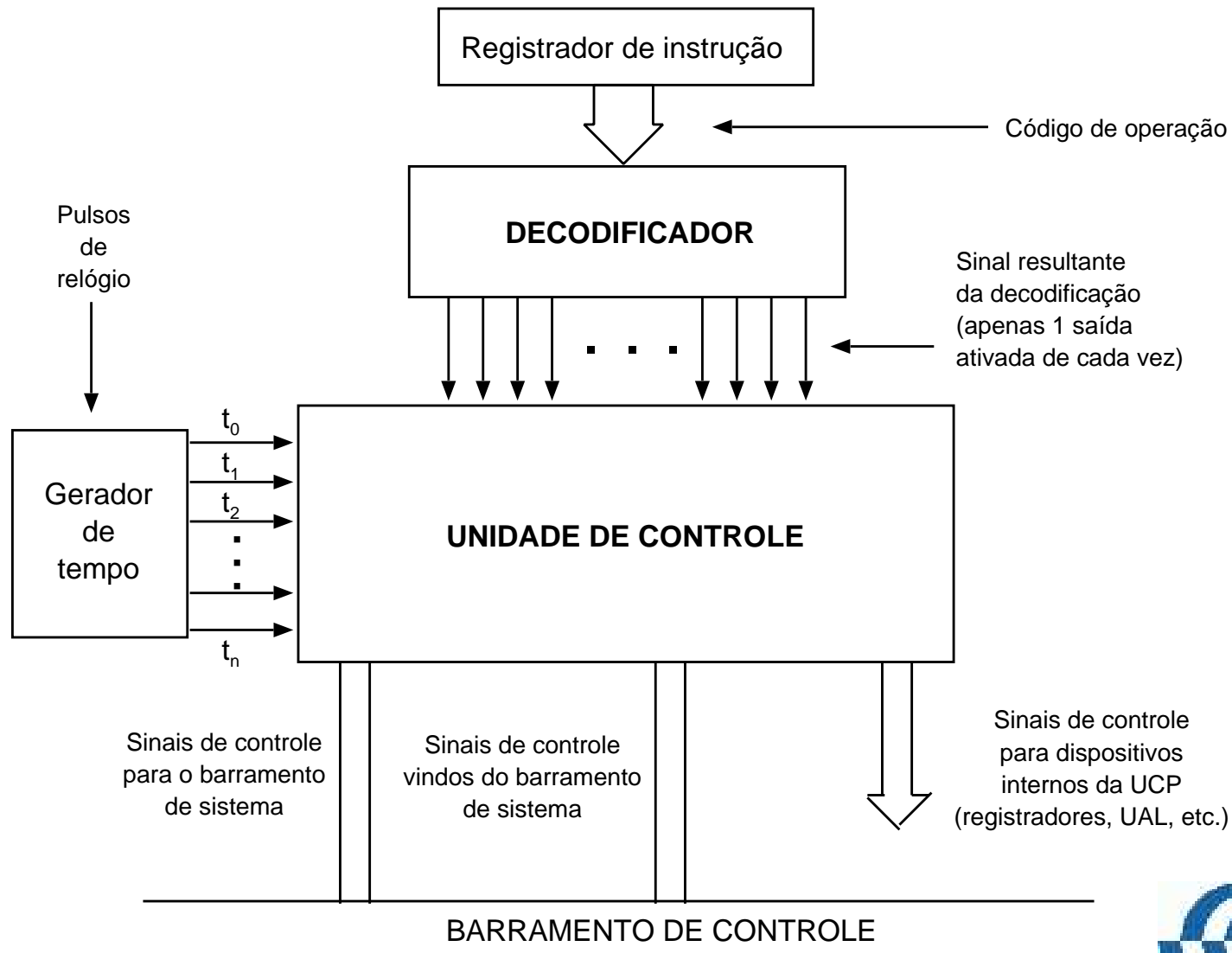
- A **unidade de controle (UC)** possui a lógica necessária para realizar a movimentação de dados e de instruções de e para a UCP, através de sinais de controle emitidos em instantes de tempo programados



(Fig. 6.12 do livro texto)

# Funções básicas da UCP

- Diagrama simplificado da **unidade de controle (UC)**



(Fig. 6.13 do livro texto)

# Funções básicas da UCP

- A execução de cada instrução de máquina é composta de microeventos, cuja execução é comandada pela unidade de controle segundo dois princípios de arquitetura:
  - Microprogramação
  - Programação prévia diretamente no hardware
- Exemplo de microeventos de um ciclo de busca:

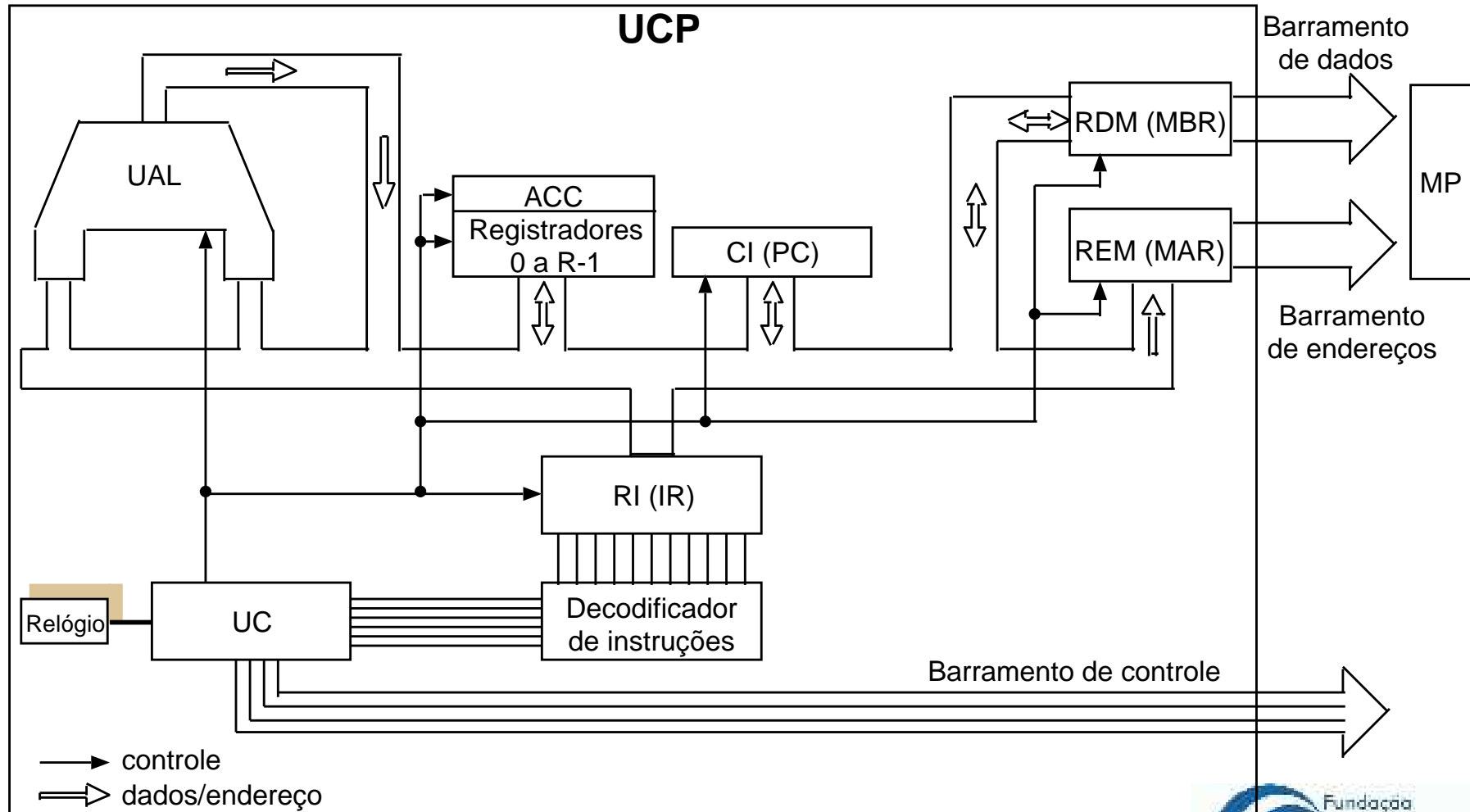
$t_0$  : REM ← (CI)  
 $t_1$  : CI ← CI + N  
       RDM ← M(REM)  
 $t_2$  : RI ← RDM

# Funções básicas da UCP

- Modos de execução das instruções:
  - Execução exclusivamente seqüencial
  - Execução concorrente (pipeline)
  - Execução paralela
  - Execução vetorial

# Funções básicas da UCP

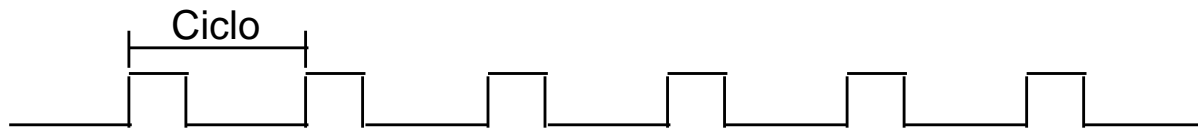
- As atividades da UCP são sincronizadas com o **relógio** da máquina



(Fig. 6.12 do livro texto)

# Funções básicas da UCP

- O **relógio** é um dispositivo gerador de pulsos
  - A duração de um pulso é denominada ciclo de relógio



- O número de vezes que o pulso se repete é denominado frequência e é medido em Hertz (Hz)
  - 1 Hz significa 1 ciclo por segundo
- O ciclo de relógio está relacionado à realização de uma operação elementar

# Funções básicas da UCP

- Suponha que cada instrução de máquina necessite do tempo de um ciclo de relógio para executar
  - M1 com relógio de frequência de 25 MHz
  - Período igual a  $1/25.000.000 = 40 \times 10^{-9} = 40$  nanosegundos
  - M2 com relógio de frequência de 400 MHz
  - Período igual a  $1/400.000.000 = 2,5 \times 10^{-9} = 2,5$  nanosegundos

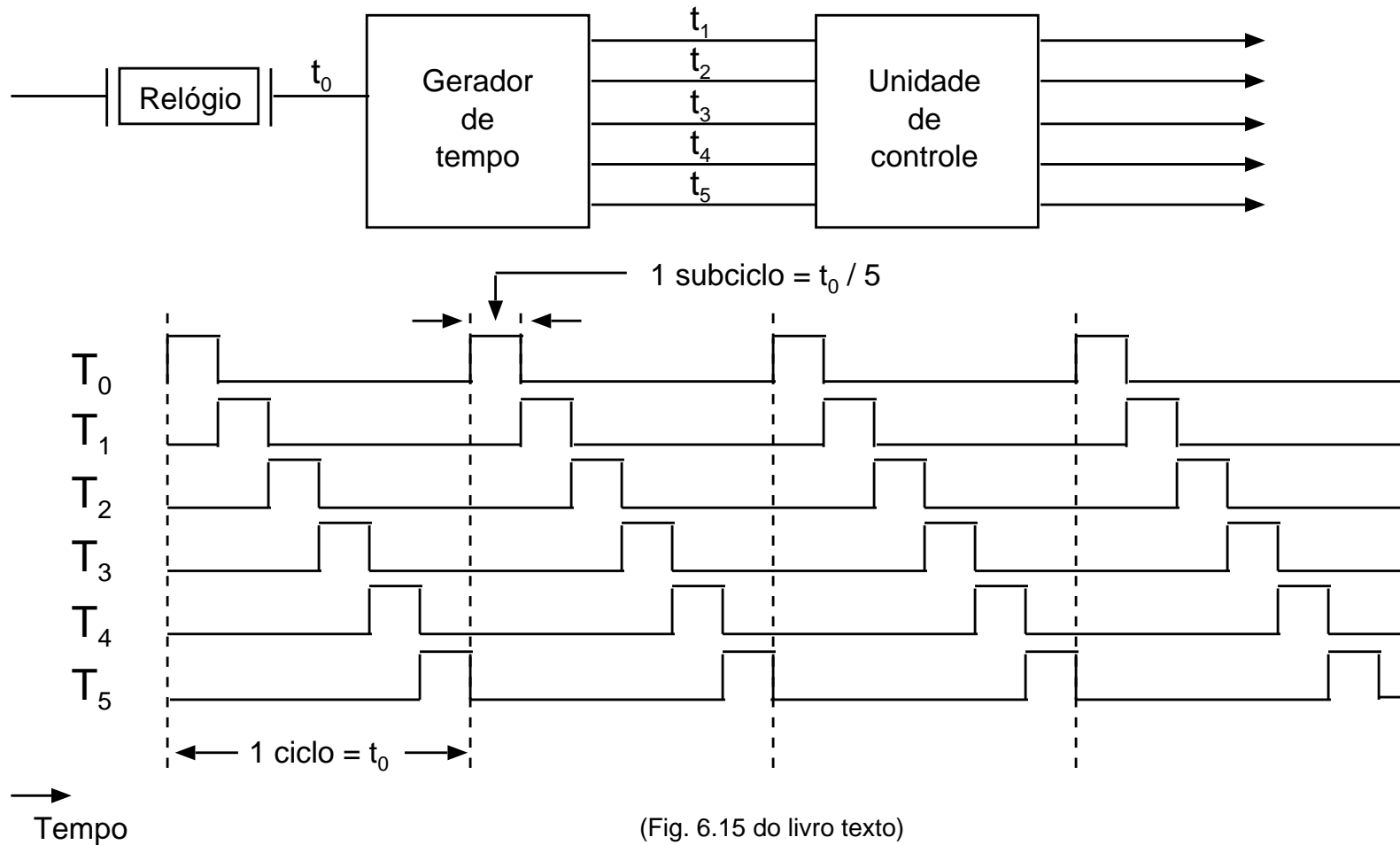


# Funções básicas da UCP

- Instruções executarão mais rapidamente em M2 do que M1 ?
  - Não necessariamente
  - Outros fatores influenciam desempenho, por exemplo:
    - Estágios de pipelining
    - Memória cache maior

# Funções básicas da UCP

- Instrução elementar é realizada em vários passos
- Ciclo do relógio é dividido em ciclos menores

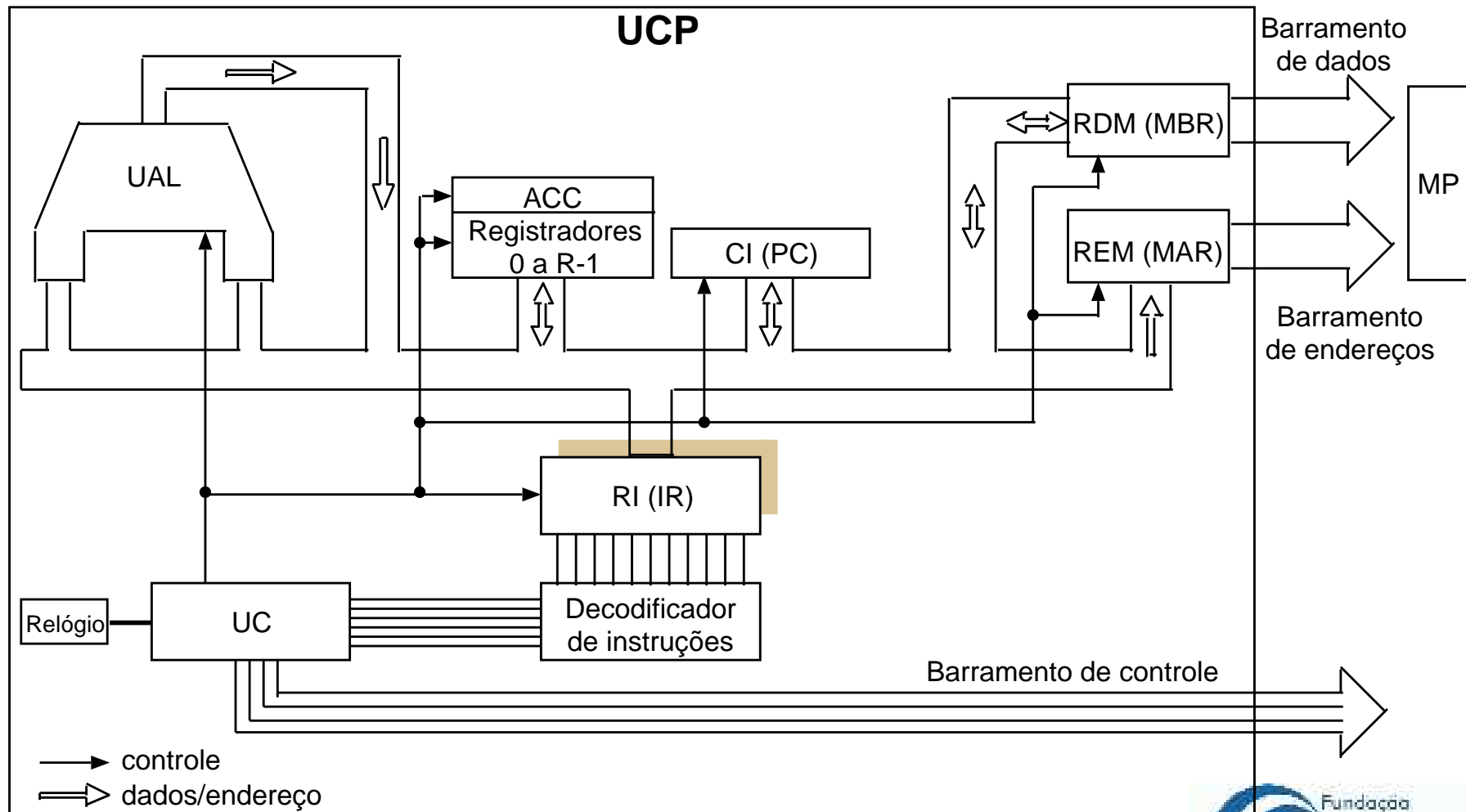


(Fig. 6.15 do livro texto)

$t_0$  : REM  $\leftarrow$  (CI)  
 $t_1$  : CI  $\leftarrow$  CI + N  
 RDM  $\leftarrow$  M(REM)  
 $t_2$  : RI  $\leftarrow$  RDM

# Funções básicas da UCP

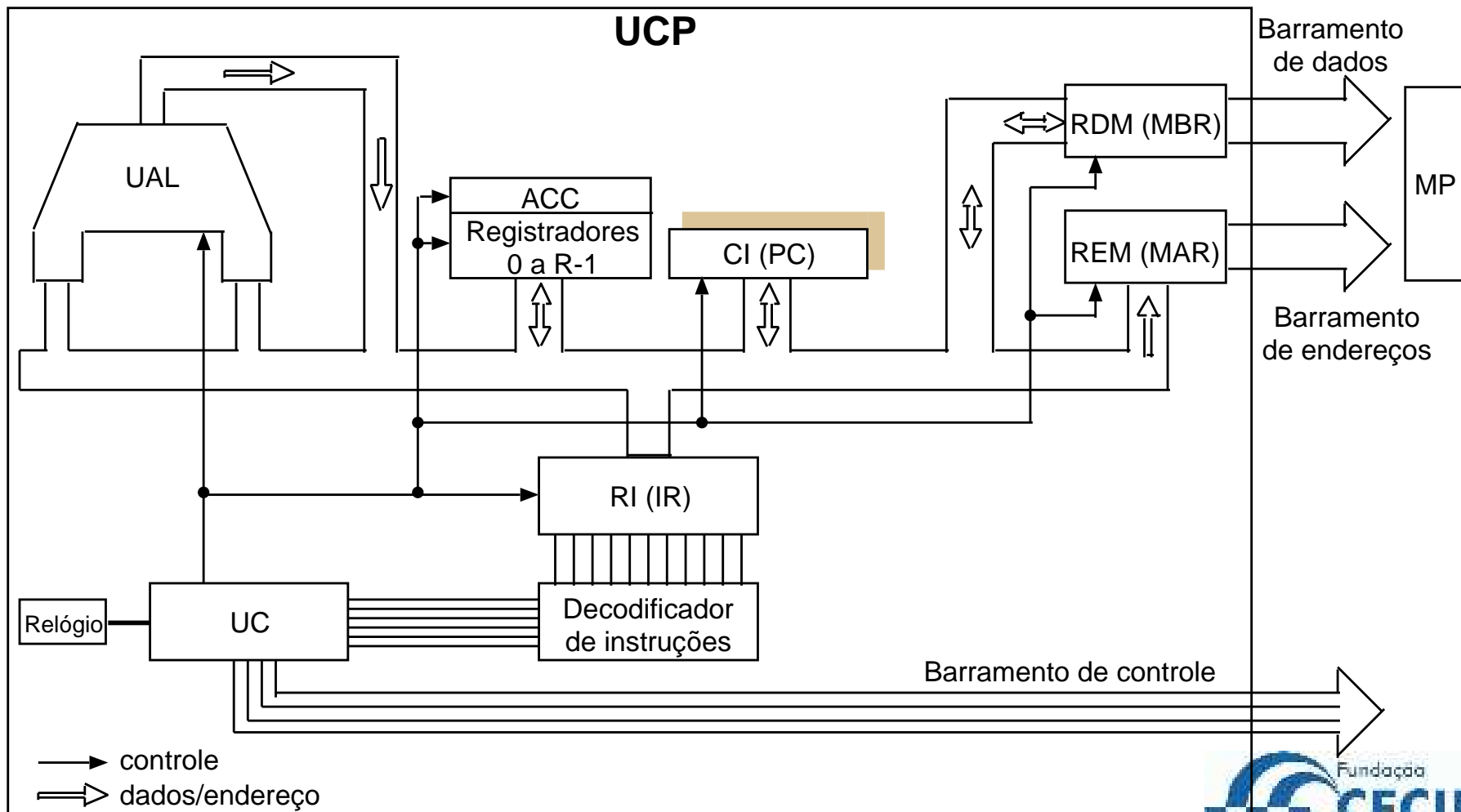
- O **registrador de instrução** (RI) tem a função específica de armazenar a instrução a ser executada pela UCP



(Fig. 6.12 do livro texto)

# Funções básicas da UCP

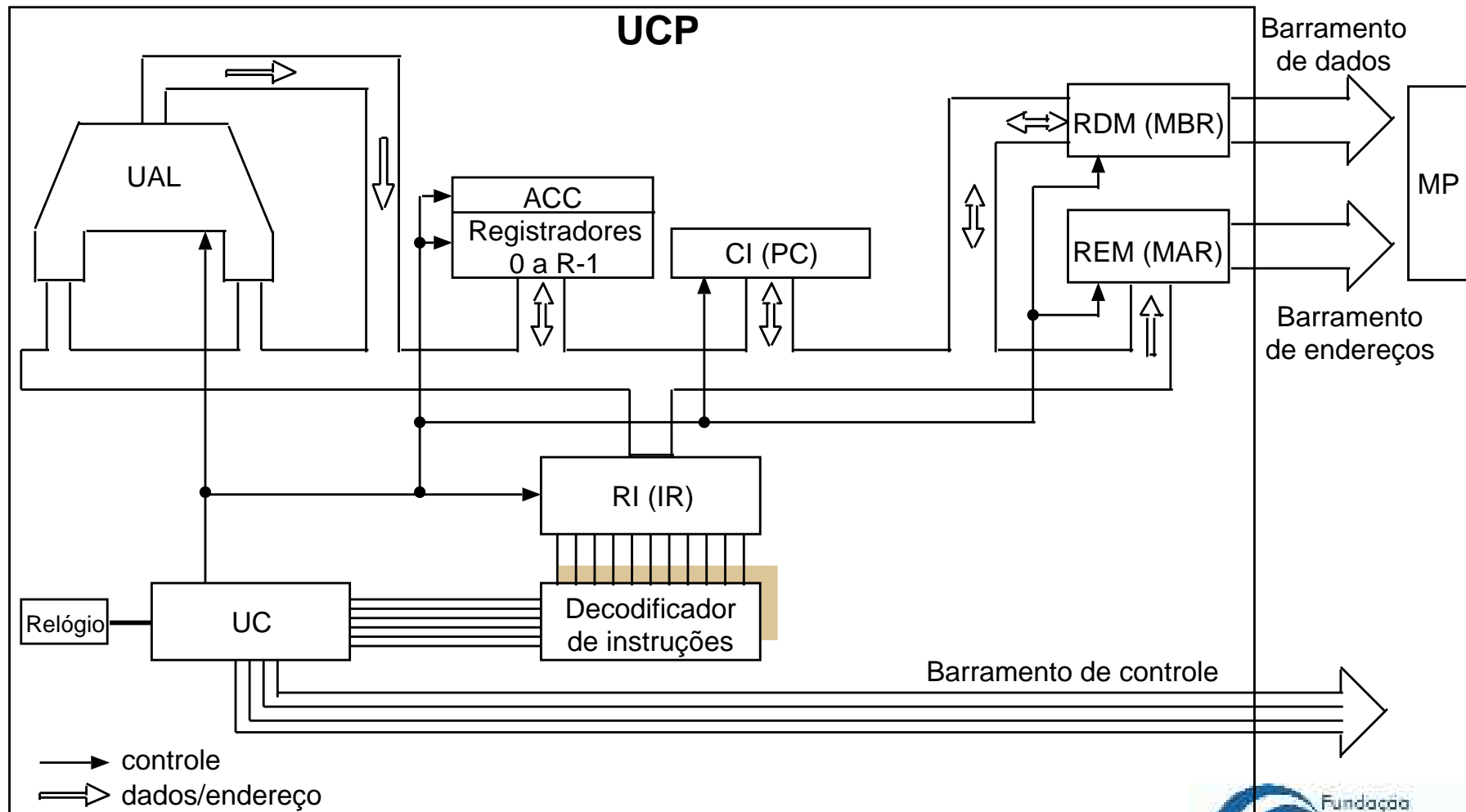
- O **contador de instrução** (CI) tem a função específica de armazenar o endereço da próxima instrução a ser executada pela UCP
- Ao se buscar uma instrução, conteúdo de CI é modificado para armazenar endereço da próxima instrução



(Fig. 6.12 do livro texto)

# Funções básicas da UCP

- O **decodificador** é um dispositivo utilizado para identificar a operação a ser realizada



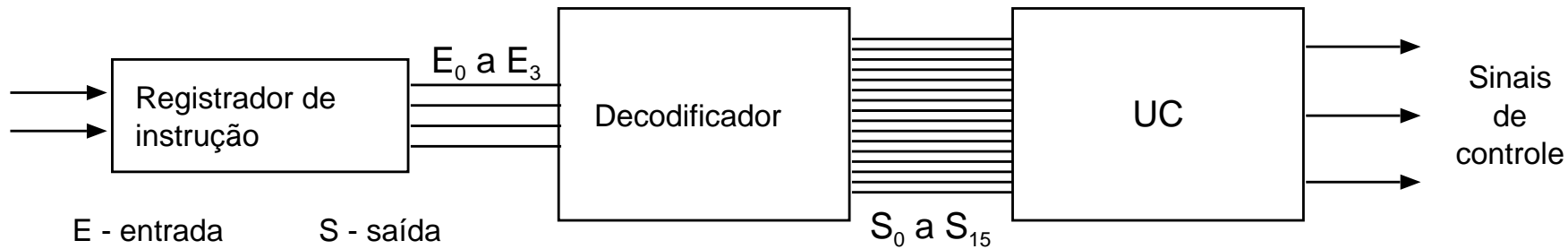
(Fig. 6.12 do livro texto)

# Funções básicas da UCP

- **Decodificador** com 4 bits de entrada e 16 saídas
- Código de operação igual a 4 bits
  - 0000-soma
  - 0001-subtração
  - 0010-multiplicação
  - 0011-divisão
  - 0100-operação AND

# Funções básicas da UCP

- Decodificador** com 4 bits de entrada e 16 saídas

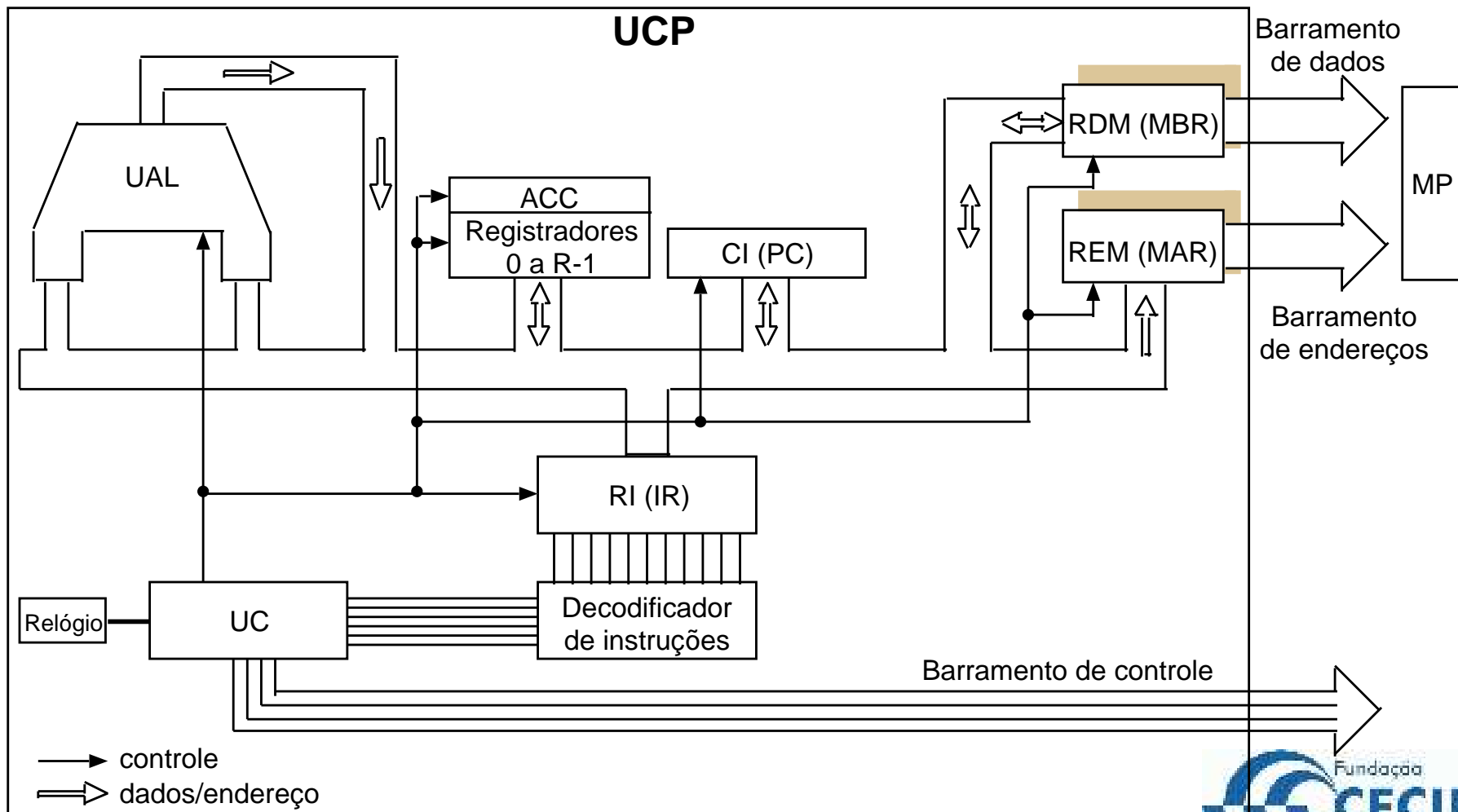


E <sub>0</sub>	E <sub>1</sub>	E <sub>2</sub>	E <sub>3</sub>	S <sub>0</sub>	S <sub>1</sub>	S <sub>2</sub>	S <sub>3</sub>	S <sub>4</sub>	S <sub>5</sub>	S <sub>6</sub>	S <sub>7</sub>	S <sub>8</sub>	S <sub>9</sub>	S <sub>10</sub>	S <sub>11</sub>	S <sub>12</sub>	S <sub>13</sub>	S <sub>14</sub>	S <sub>15</sub>
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
0	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
1	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

(Fig. 6.16 do livro texto)

# Funções básicas da UCP

- **Registrador de dados de memória** (RDM) possui um tamanho igual ao do barramento de dados
- **Registrador de endereços de memória** (REM) possui um tamanho igual ao dos endereços de memória



(Fig. 6.12 do livro texto)



# Funções básicas da UCP

- Instrução de máquina
  - É a formalização de uma operação básica (ou primitiva) que o hardware é capaz de realizar diretamente
  - UAL pode somar ou multiplicar dois números
    - Instrução em linguagem de alto nível  $X = A + B * C$  deve ser decomposta em  $T = B * C$  e  $X = A + T$
    - Quem decompõe as instruções ?

# Funções básicas da UCP

- Projeto do processador:
  - Definir o conjunto de instruções de linguagem de máquina
    - Formato
    - Tamanho
    - Operações
  - Implementar os componentes do processador em função da definição deste conjunto de instruções

# Funções básicas da UCP

- Tecnologia de processadores
  - Sistemas com conjunto de instruções complexo (CISC-Complex Instruction Set Computers)
  - Sistemas com conjunto de instruções reduzido (RISC-Reduced Instruction Set Computers)
- Intel 8080 possuía 78 instruções
- Intel 80486 possuía 286 instruções
- Intel Pentium II possui 217 instruções
- MIPS R2000 possui 110 instruções

# Funções básicas da UCP

- Formato das instruções
  - Grupo de bits dividido em duas partes:
    - A primeira parte é constituída de um só campo que identifica a operação a ser executada (*código de operação*)
    - A segunda parte pode ter um ou mais campos que se referem aos dados que devem ser manipulados pela operação (*operando(s)*)

# Funções básicas da UCP

- Código de operação
  - Campo da instrução cujo valor binário identifica a operação a ser realizada
  - Um processador que possua o campo de código de operação com 8 bits pode ser fabricado contendo um número máximo de instruções igual a  $2^8 = 256$
- Operandos
  - Campos da instrução cujo valor binário identifica a localização dos dados que serão manipulados durante a realização da operação

# Funções básicas da UCP

- Formatos de instruções

C. Op	Operando 1	Operando 2	Operando 3
-------	------------	------------	------------

$(\text{Operando 3}) \leftarrow (\text{Operando 1}) + (\text{Operando 2})$

C. Op	Operando 1	Operando 2
-------	------------	------------

$(\text{Operando 1}) \leftarrow (\text{Operando 1}) + (\text{Operando 2})$

C. Op	Operando
-------	----------

$(\text{ACC}) \leftarrow (\text{ACC}) + (\text{Operando})$

# Funções básicas da UCP

- Formatos de instruções serão vistos com mais detalhes adiante
- Campo do operando indica onde o operando se localiza
  - Modos de endereçamento

# Funções básicas da UCP

- Códigos de operação podem ter tamanho:
  - Fixo
    - Mais simples de manipular e implementar
    - O tamanho deve ser o suficiente para acomodar todos os códigos necessários
  - Variável
    - Permitem codificar uma quantidade maior de instruções com menor quantidade de bits



## **Bibliografia adicional:**

- Organização e Projeto de Computadores,  
A Interface Hardware/Software -  
David A. Patterson; John L. Hennessy- LTC, 2000
- Arquitetura e Organização de Computadores -  
W. Stallings - Prentice Hall

## Aula 4

### Professores:

Lúcia M. A. Drummond  
Simone de Lima Martins

### Conteúdo:

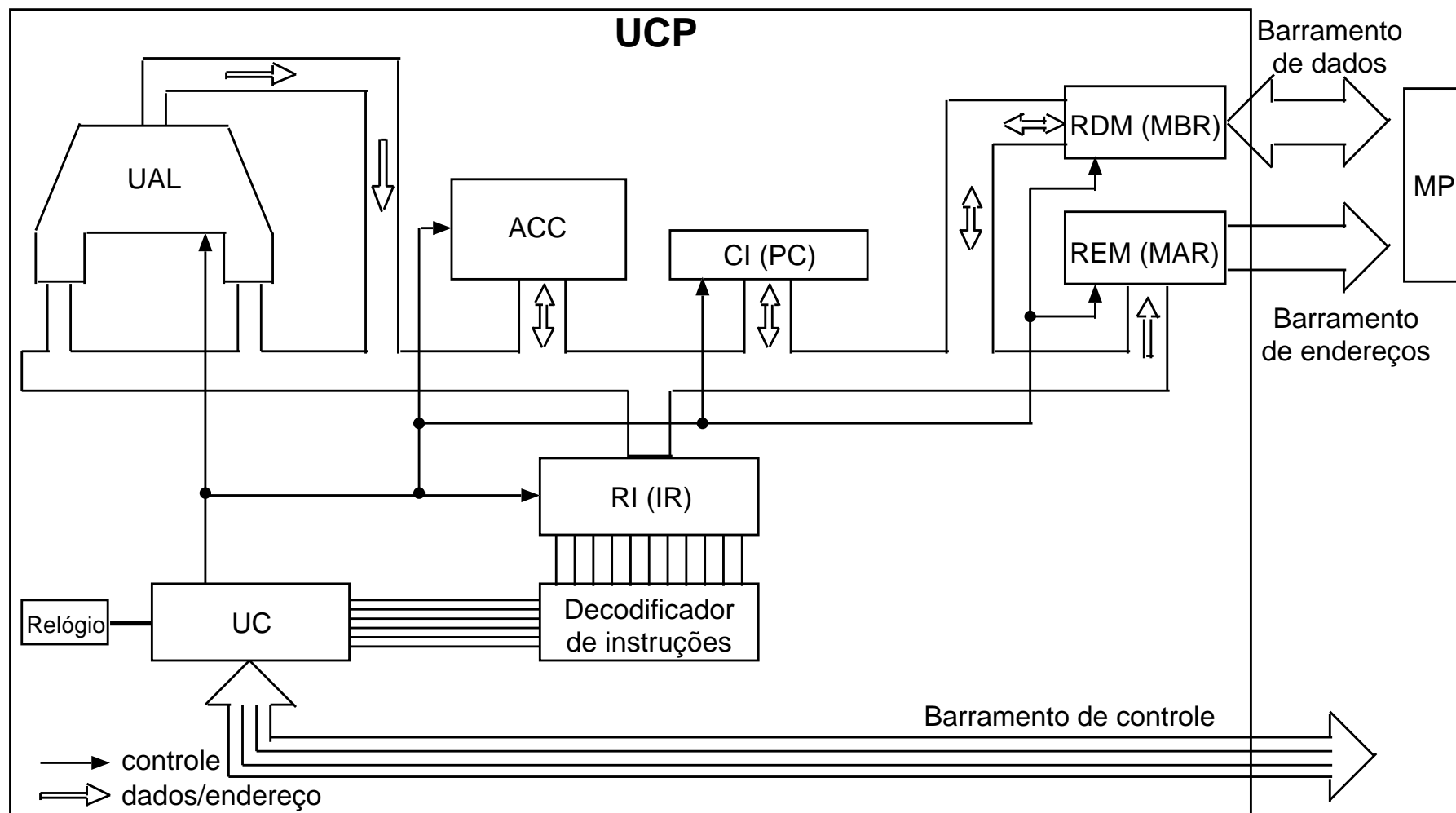
#### Unidade Central de Processamento 2

- Funcionamento da UCP: O ciclo de instrução
- Linguagem de montagem
- Unidade Aritmética e Lógica

## Funcionamento da UCP: O ciclo de instrução

- A base do projeto de uma UCP é a escolha do conjunto de instruções
- Os componentes da arquitetura e sua organização são definidos para interpretar e executar as instruções deste conjunto
- Um subsistema UCP/MP hipotético será utilizado para mostrar as etapas requeridas para a execução de instruções

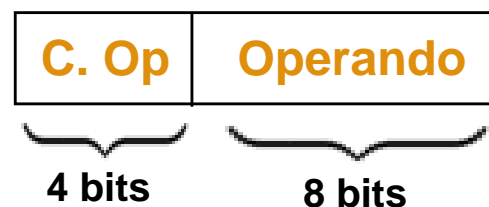
# Funcionamento da UCP: O ciclo de instrução



(Fig. 6.3 do livro texto)

## Funcionamento da UCP: O ciclo de instrução

- Processador hipotético
  - Palavra: 12 bits
  - Endereços: 8 bits ( $2^8 = 256$  células de memória)
  - Células de 12 bits
  - A UCP possui um registrador de dados (ACC) com 12 bits de tamanho, o CI e REM com 8 bits cada um e o RDM com 12 bits
  - Formato das instruções



- Operando indica endereço na memória, exceto em instrução de desvio

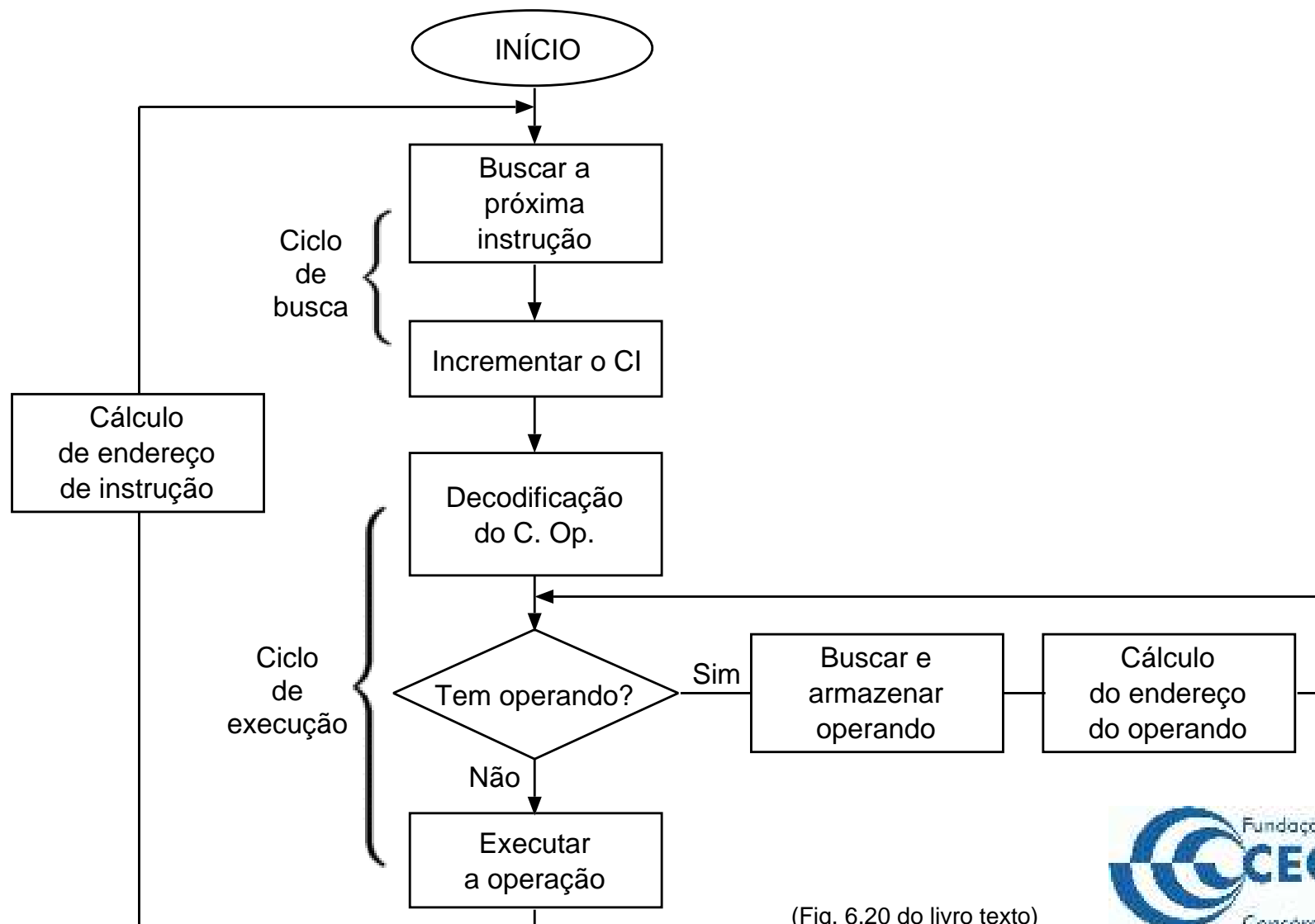
## Funcionamento da UCP: O ciclo de instrução

- Instruções disponíveis

C. Op.	Sigla	Descrição
0	HLT	Parar a execução do programa
1	LDA Op.	$ACC \leftarrow (Op.)$ , Load
2	STR Op.	$(Op.) \leftarrow ACC$ , Store
3	ADD Op.	$ACC \leftarrow ACC + (Op.)$
4	SUB Op.	$ACC \leftarrow ACC - (Op.)$
5	JZ Op.	Se $ACC = 0$ , então $CI \leftarrow Op.$
6	JP Op.	Se $ACC > 0$ , então $CI \leftarrow Op.$
7	JN Op.	Se $ACC < 0$ , então $CI \leftarrow Op.$
8	JMP Op.	$CI \leftarrow Op.$
9	GET Op.	Ler dado da porta de entrada e armazená-lo em (Op.)
A	PRT Op.	Colocar dado armazenado em (Op.) na porta refer. à impressora

# Funcionamento da UCP: O ciclo de instrução

- Fluxograma de um ciclo de instrução



(Fig. 6.20 do livro texto)

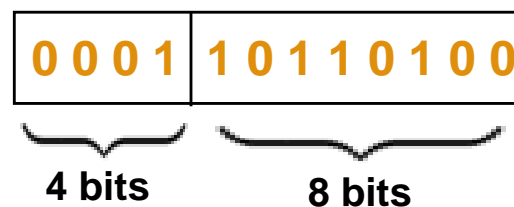
## Funcionamento da UCP: O ciclo de instrução

- Execução de duas instruções armazenadas na memória
  - LDA Op armazenada no endereço 2
  - ADD Op armazenada no endereço 3



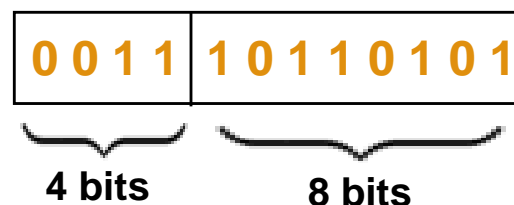
## Funcionamento da UCP: O ciclo de instrução

- **LDA 180**
  - Coloca o conteúdo do endereço de memória 180 no acumulador
  - Código de operação =  $1_{16} = 0001$
  - Endereço de memória =  $180_{10} = 10110100$
  - Instrução =  $1B4_{16}$



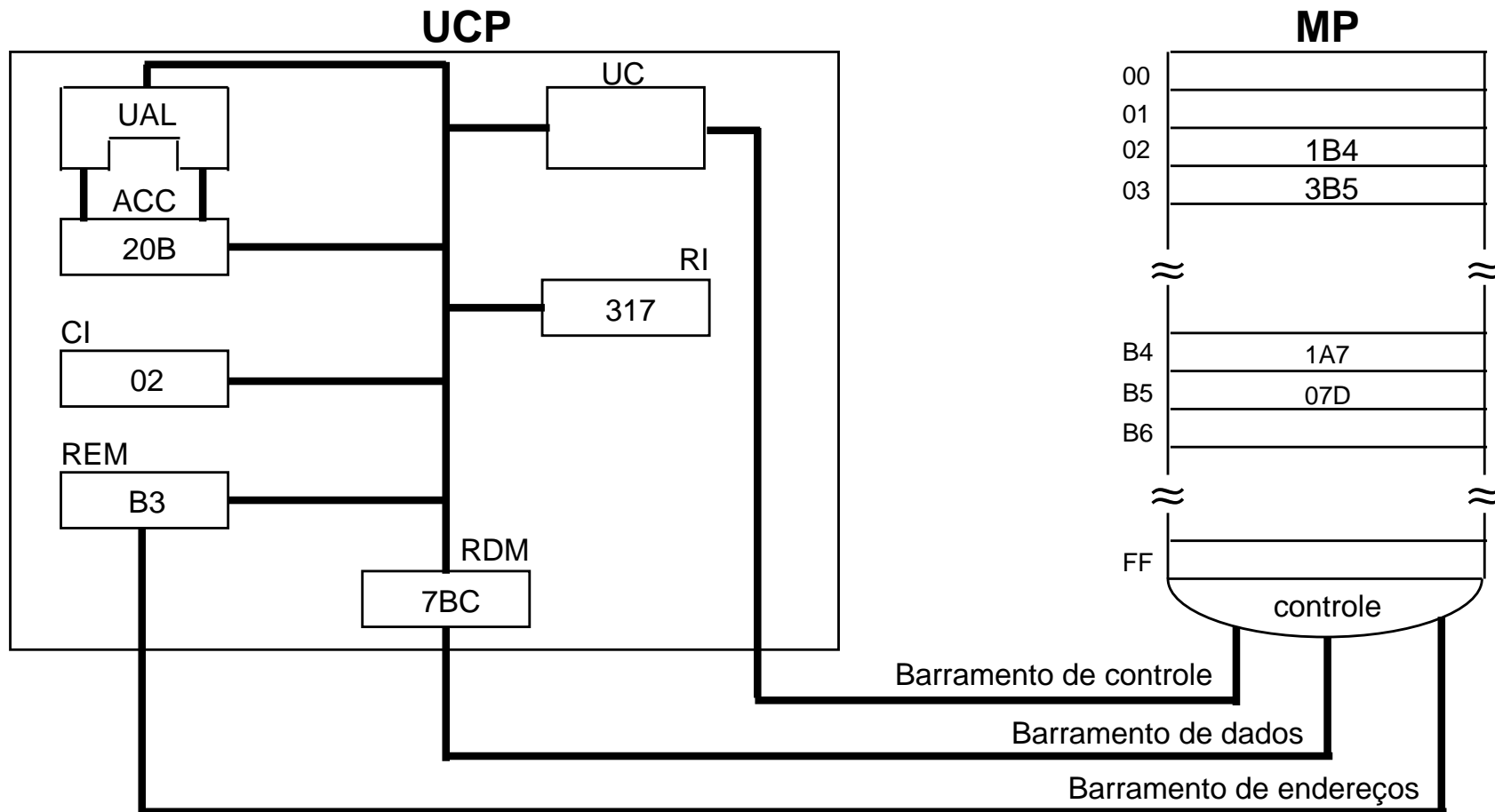
## Funcionamento da UCP: O ciclo de instrução

- **ADD 181**
  - Soma o conteúdo do endereço de memória 181 com o conteúdo do acumulador e armazena resultado no acumulador
  - Código de operação =  $3_{16} = 0011$
  - Endereço de memória =  $181_{10} = 10110101$
  - Instrução =  $3B5_{16}$



# Funcionamento da UCP: O ciclo de instrução

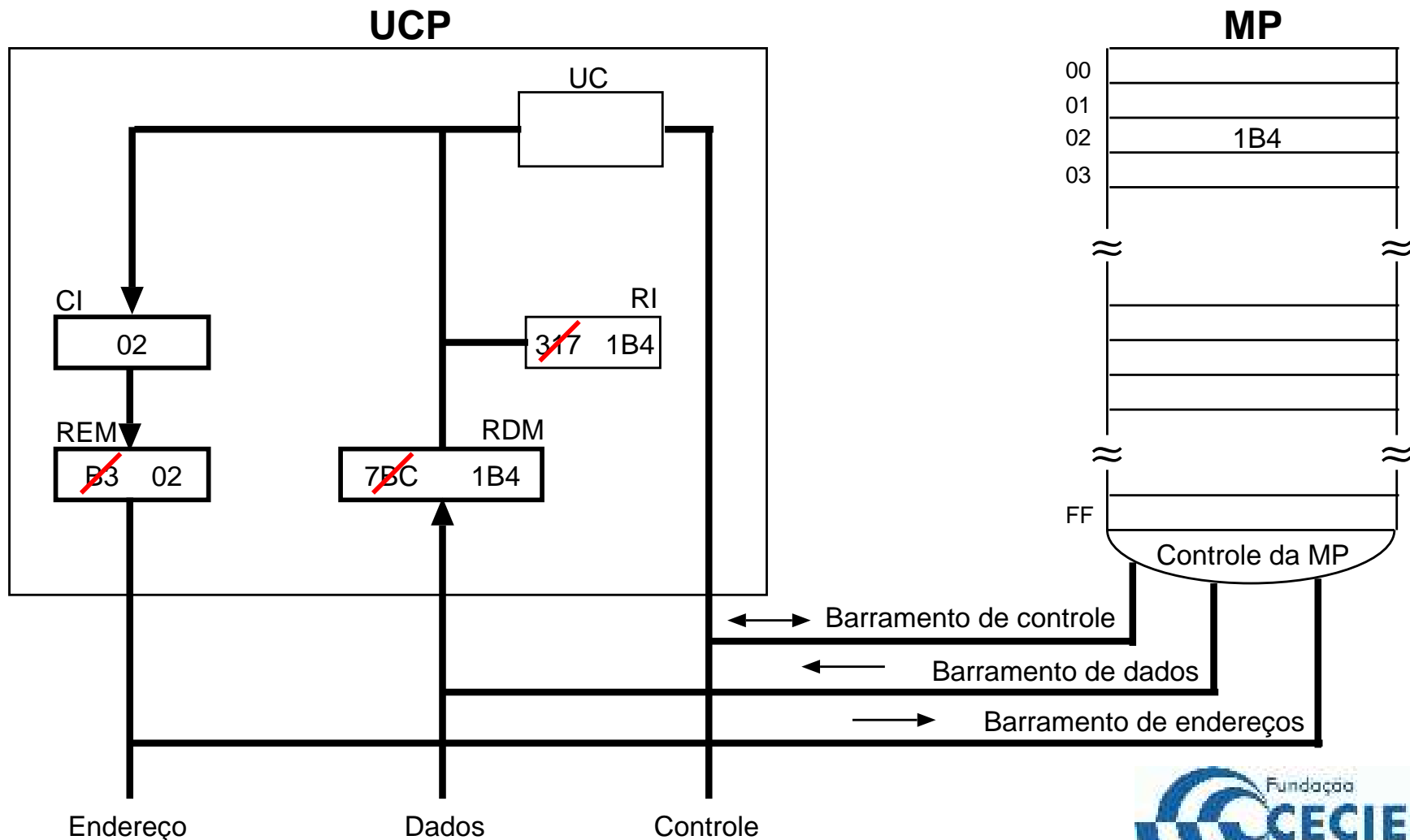
- Cenário de execução das instruções



(Fig. 6.21 do livro texto)

# Funcionamento da UCP: Execução de LDA

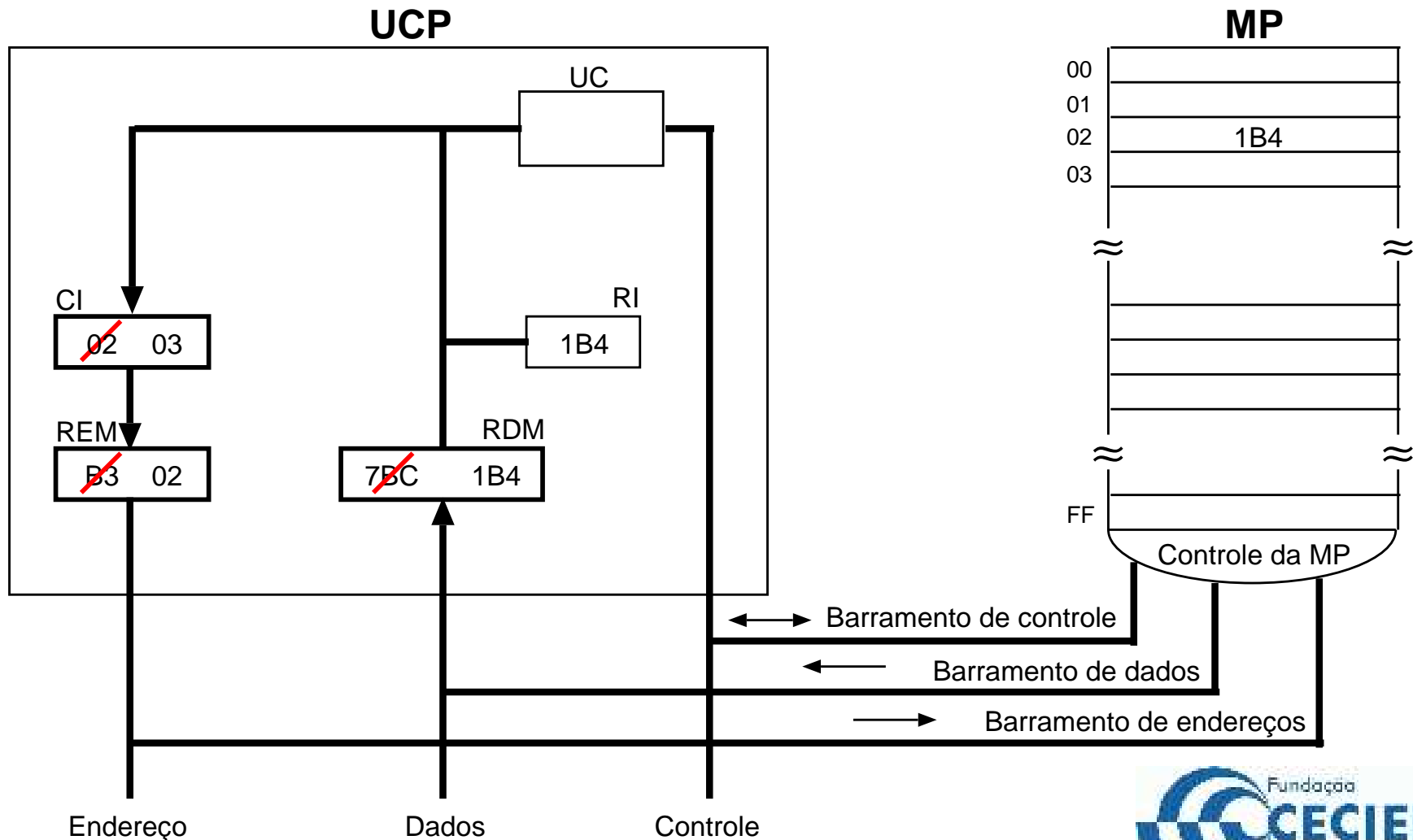
- $RI \leftarrow (CI)$



(Fig. 6.22 do livro texto)

# Funcionamento da UCP: Execução de LDA

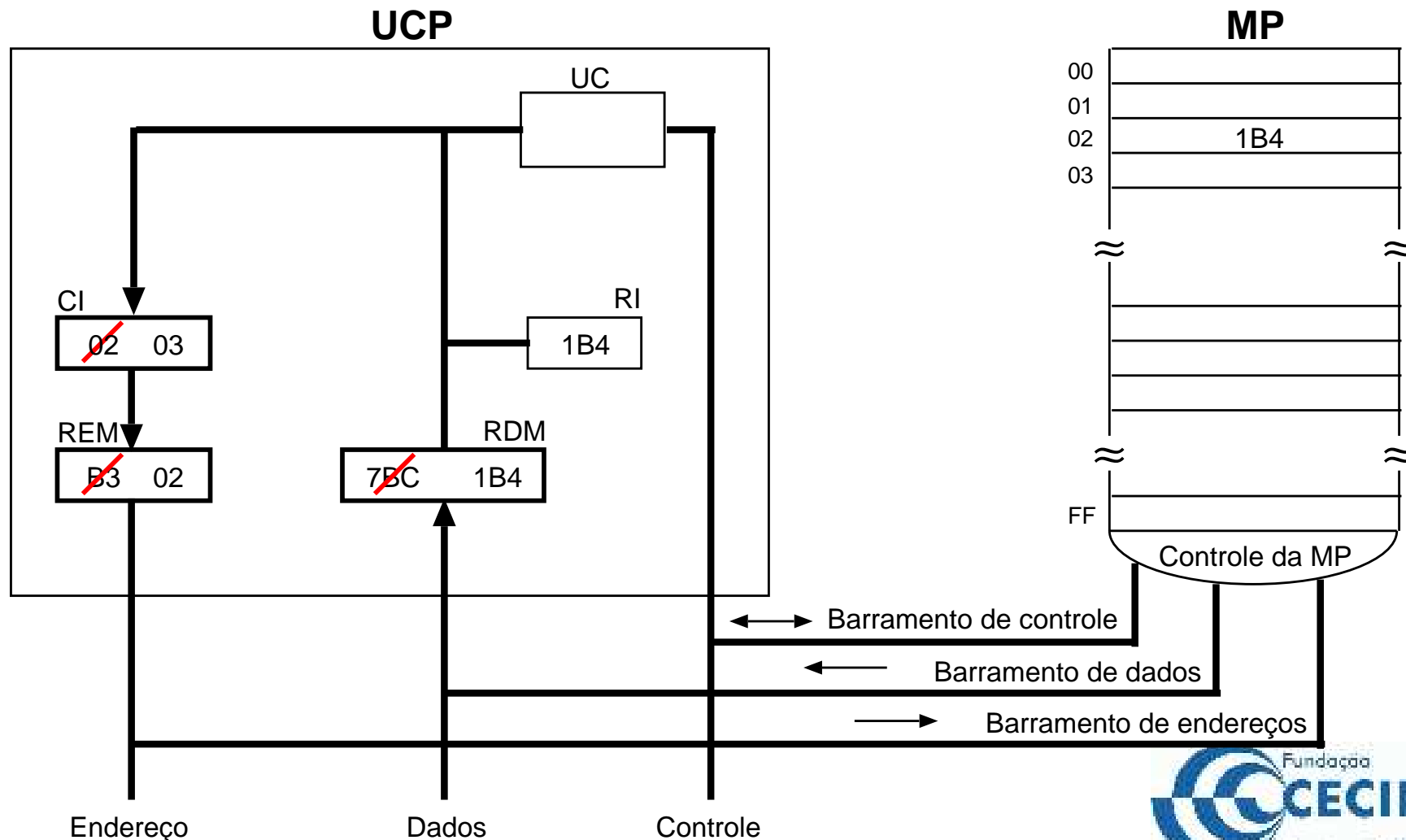
- $CI \leftarrow CI + 1$



(Fig. 6.22 do livro texto)

# Funcionamento da UCP: Execução de LDA

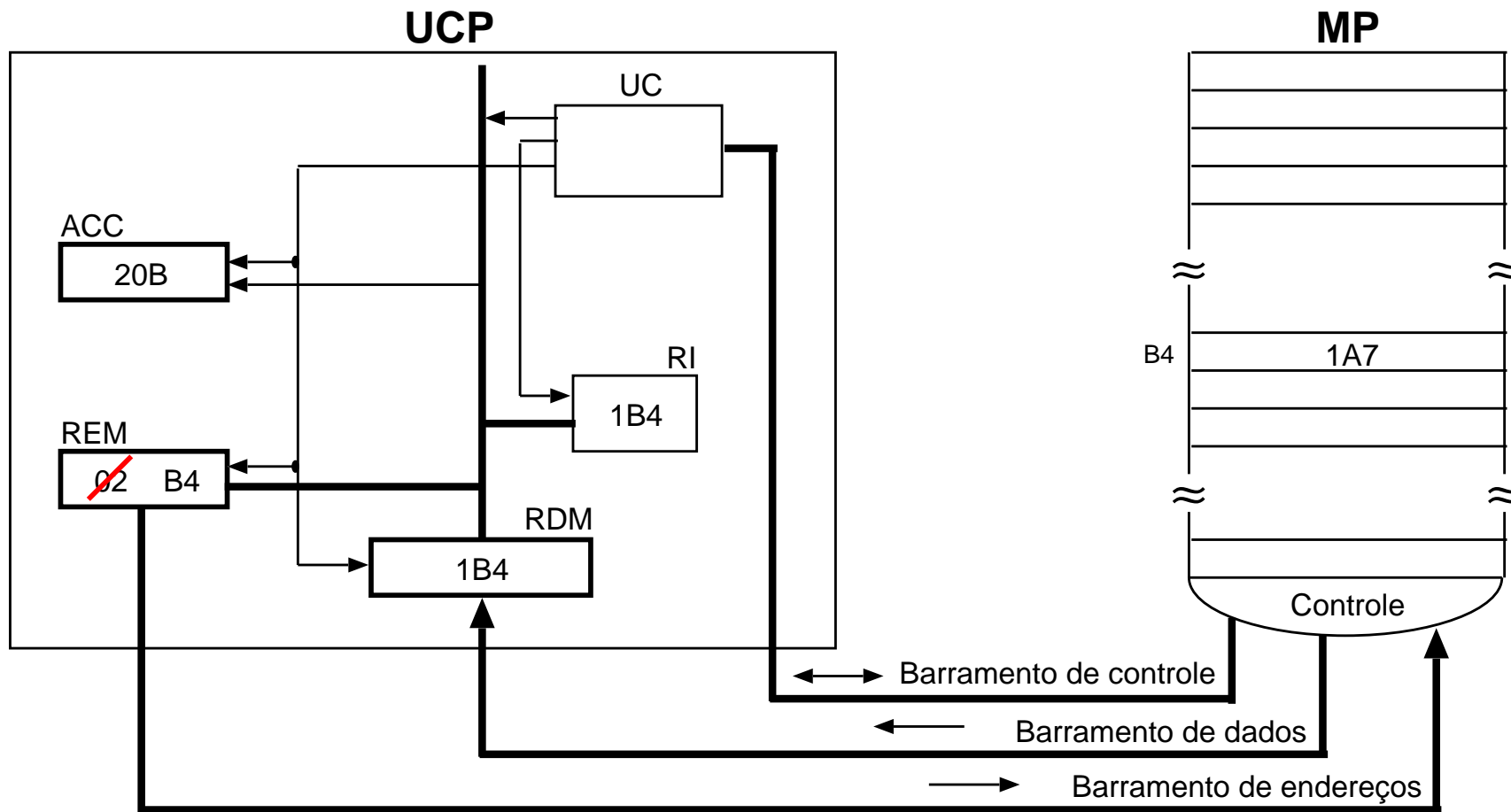
- Decodificação do código de operação
  - Decodificador recebe os 4 bits do código de operação
  - Decodificador gera sinais para que a unidade de controle produza os sinais para executar a operação definida pela instrução (transfere dado da memória para acumulador)



(Fig. 6.22 do livro texto)

# Funcionamento da UCP: Execução de LDA

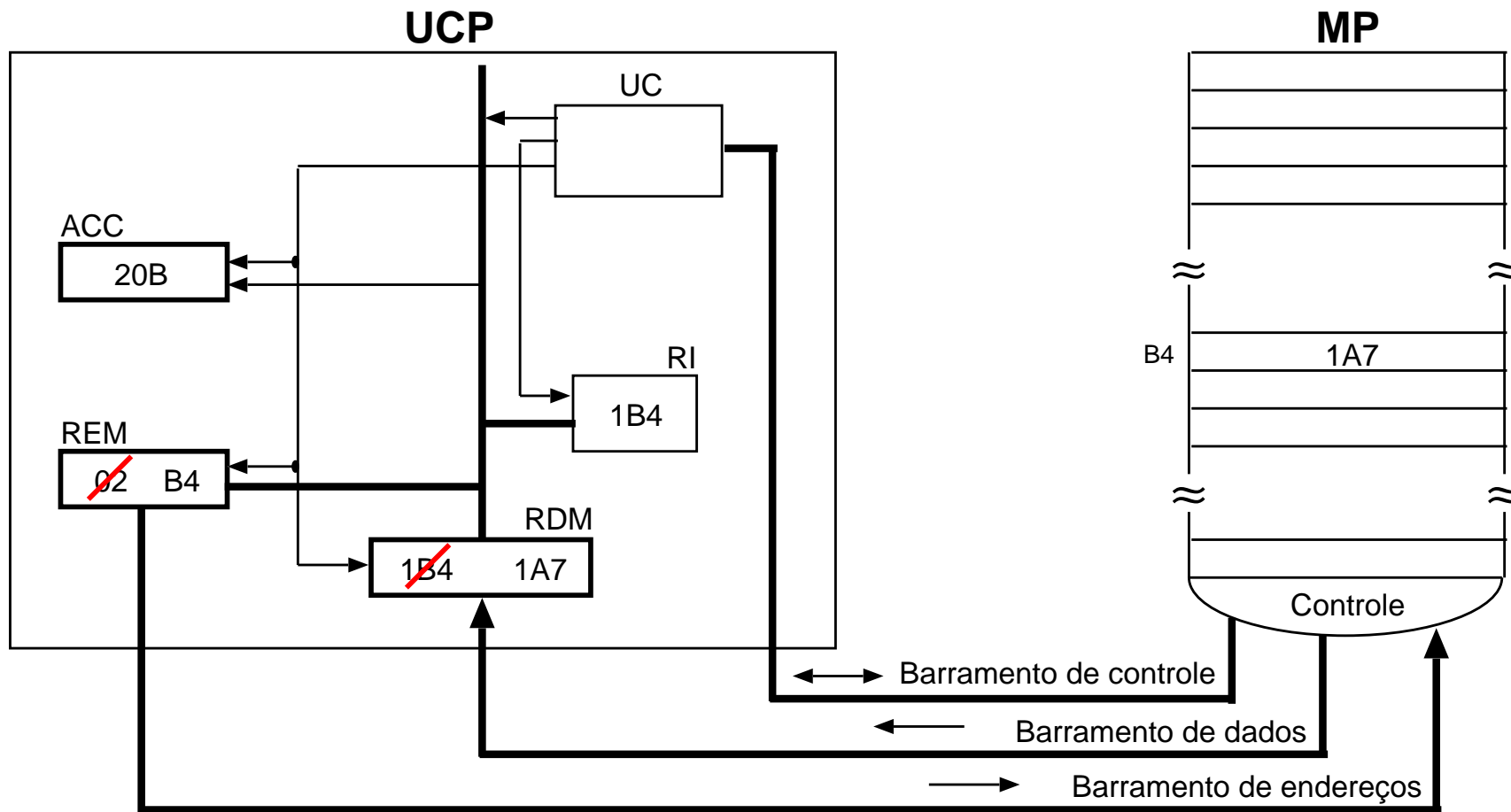
- Execução da operação
  - A UC emite os sinais para que o valor do campo do operando (B4) seja transferido para o REM



(Fig. 6.23 do livro texto)

# Funcionamento da UCP: Execução de LDA

- Execução da operação
  - A UC ativa a linha READ do barramento de controle
  - Conteúdo do endereço de memória B4 é transferido para o RDM

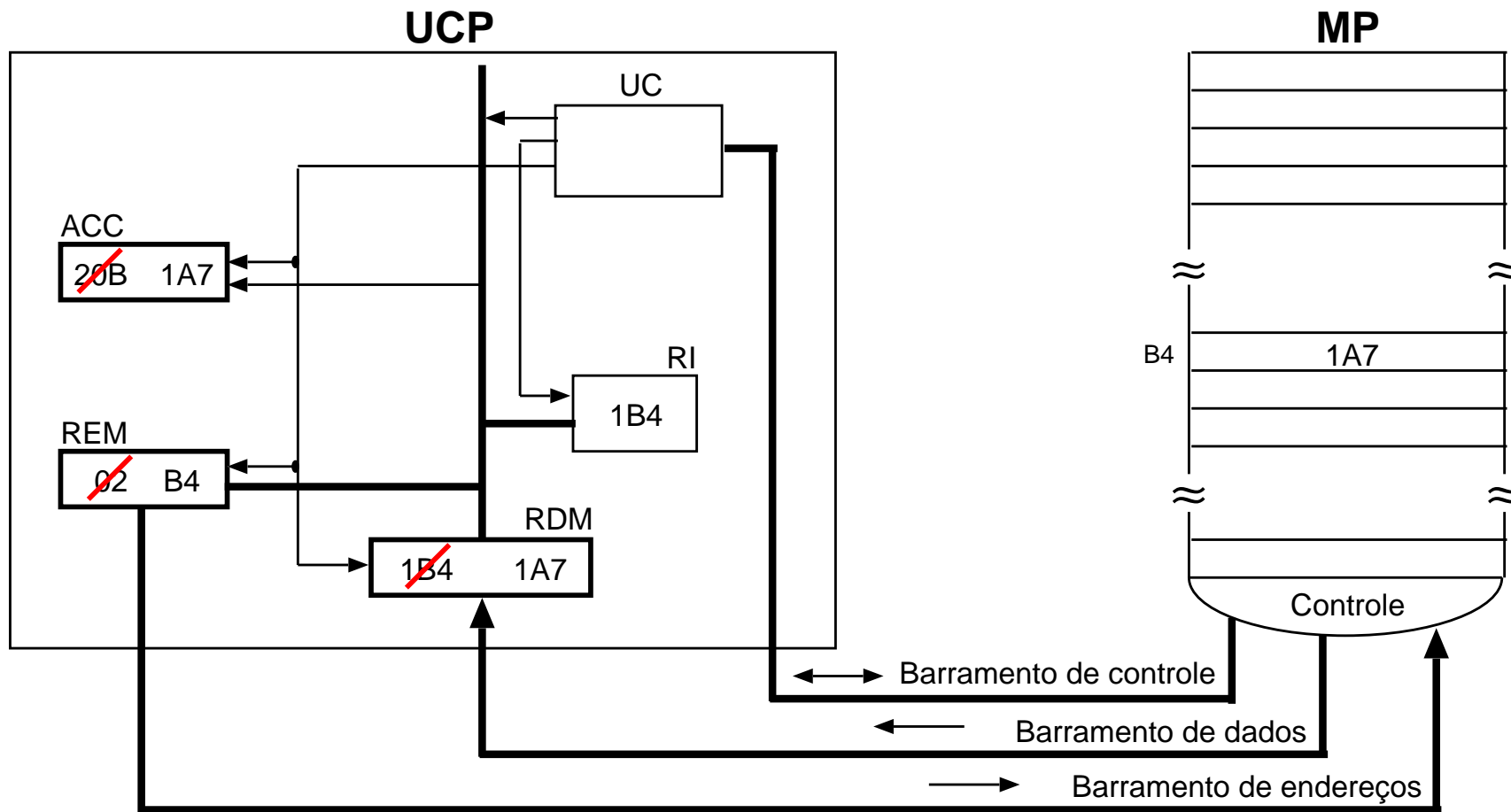


(Fig. 6.23 do livro texto)



# Funcionamento da UCP: Execução de LDA

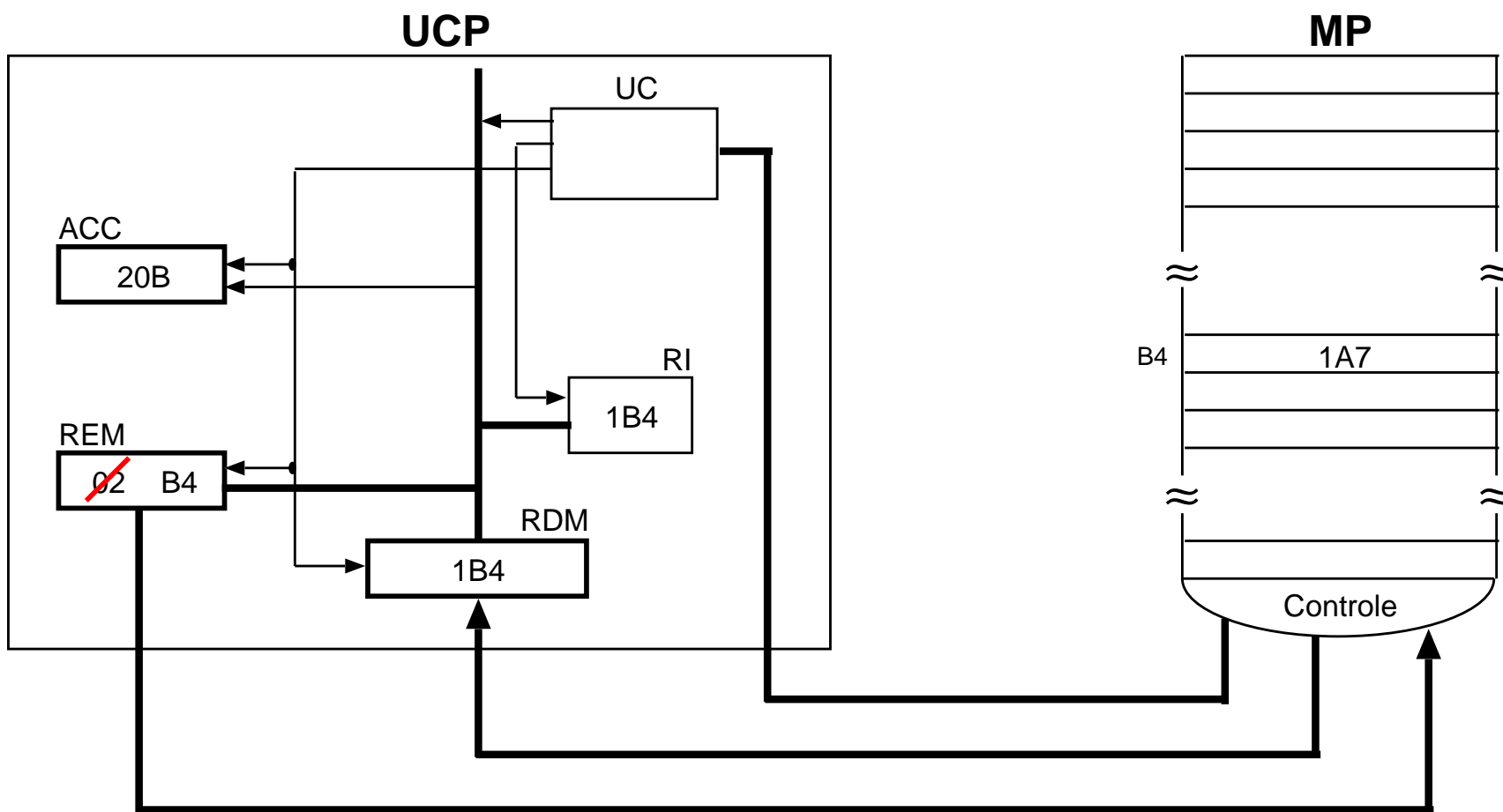
- Execução da operação
  - Conteúdo do RDM é transferido para acumulador



(Fig. 6.23 do livro texto)

## Exemplo: Execução da operação

- 1 A UC emite os sinais para que o valor do campo do operando (B4) seja transferido para o REM
- 2 A UC ativa a linha READ do barramento de controle
- 3 Conteúdo do endereço de memória B4 é transferido para o RDM
- 4 Conteúdo do RDM é transferido para acumulador



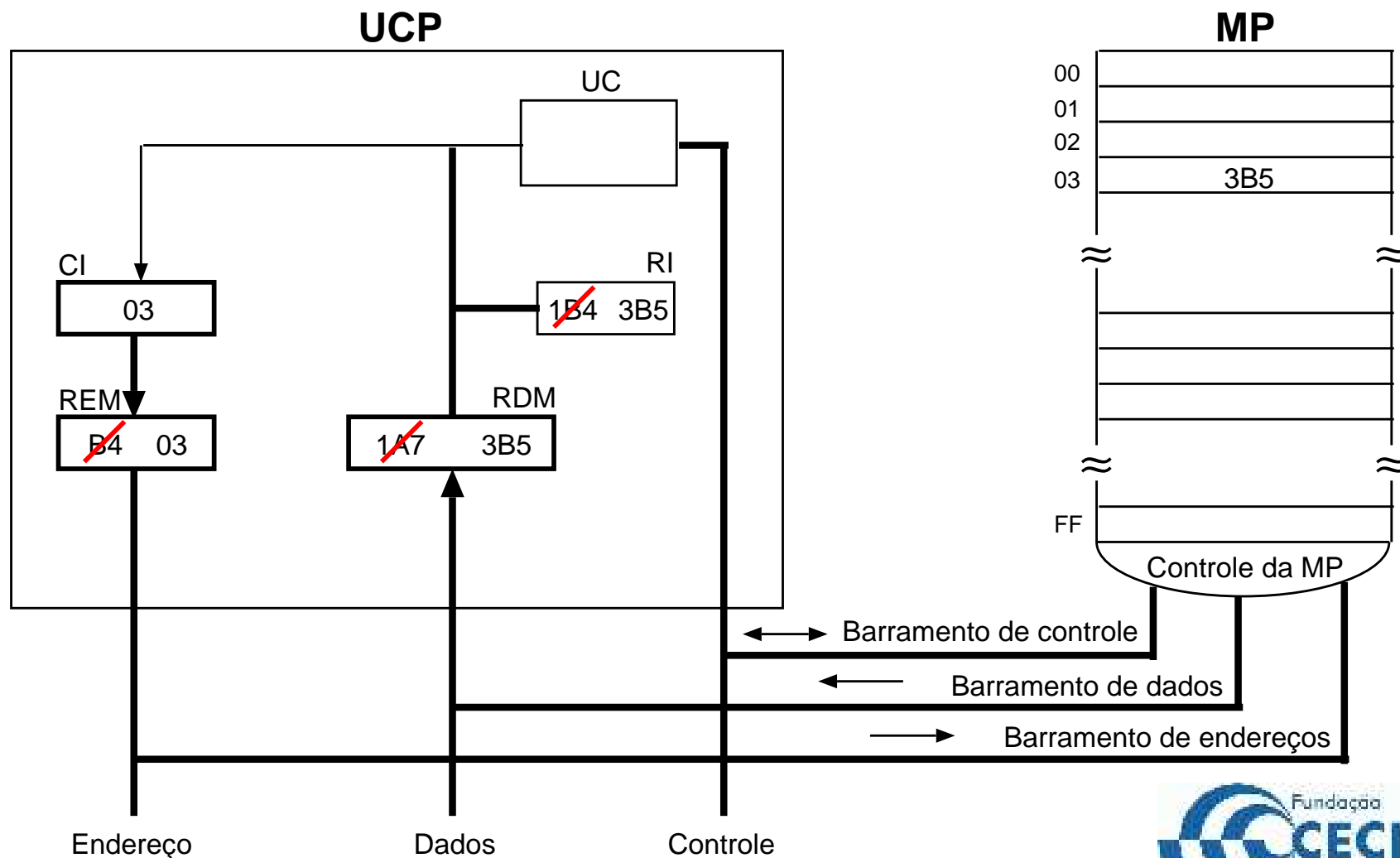
(Fig. 6.23 do livro texto)

Exemplo

Voltar

## Funcionamento da UCP: Execução de ADD

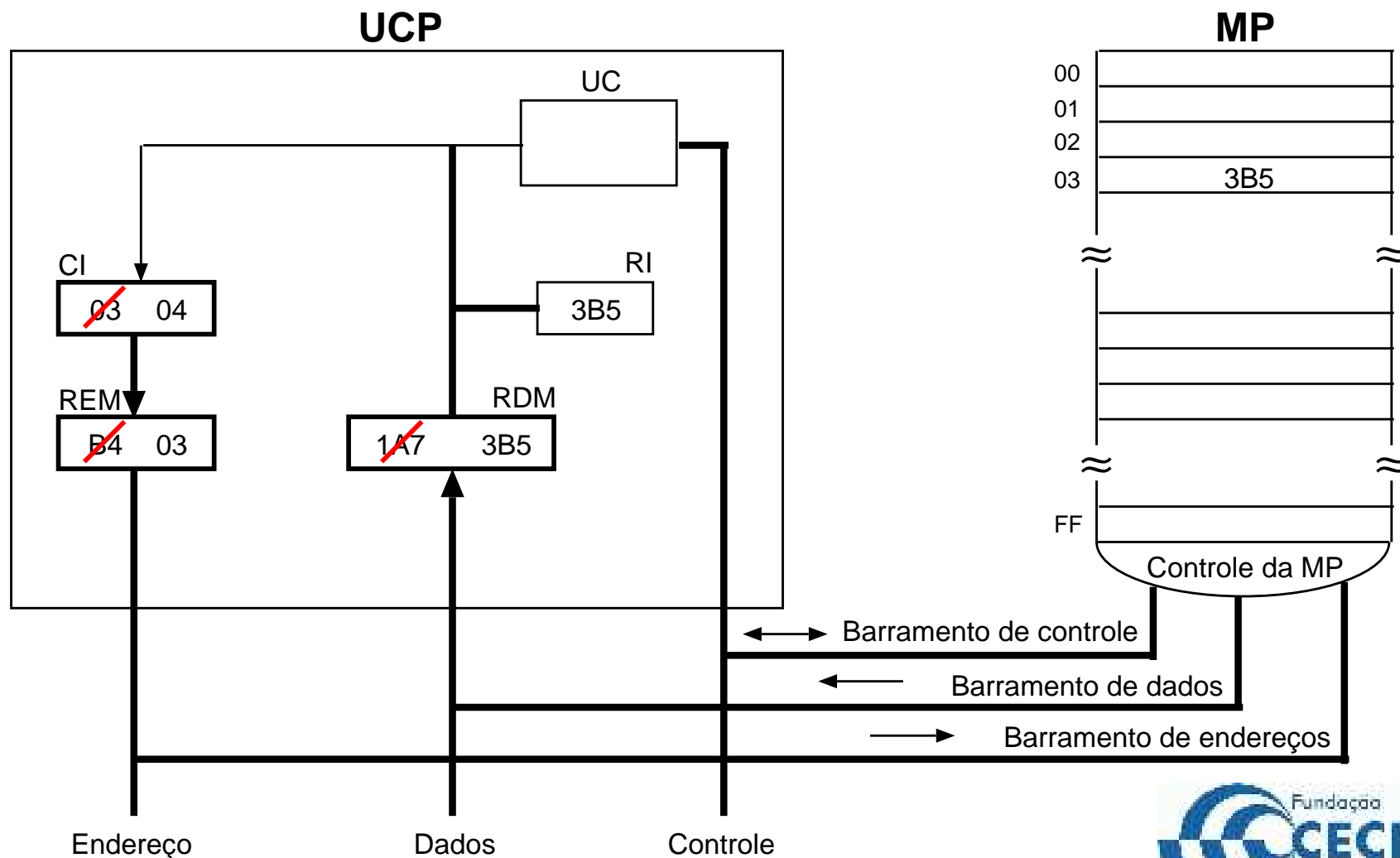
- $RI \leftarrow (CI)$



(Fig. 6.24 do livro texto)

## Funcionamento da UCP: Execução de ADD

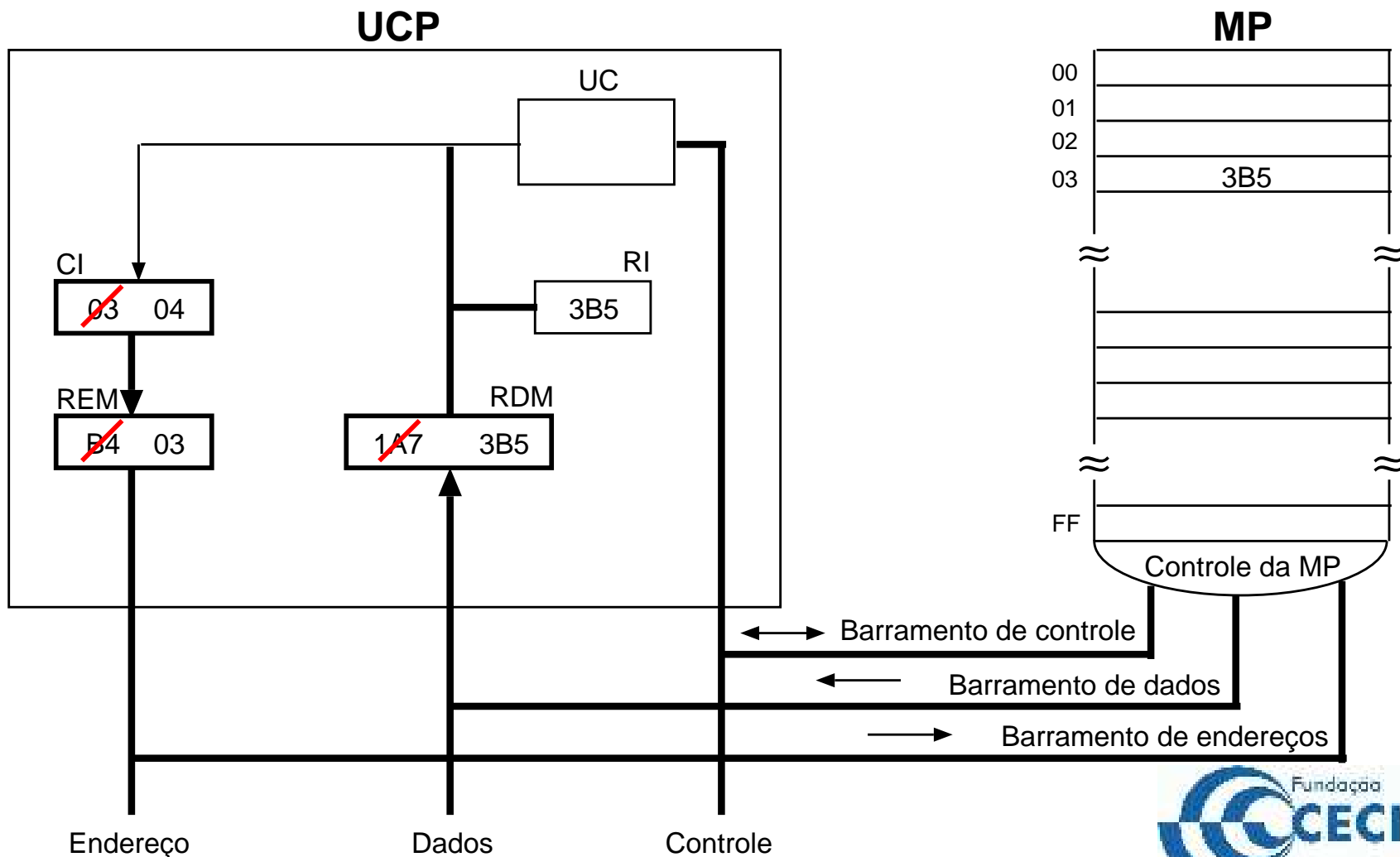
- $CI \leftarrow CI + 1$



(Fig. 6.24 do livro texto)

# Funcionamento da UCP: Execução de ADD

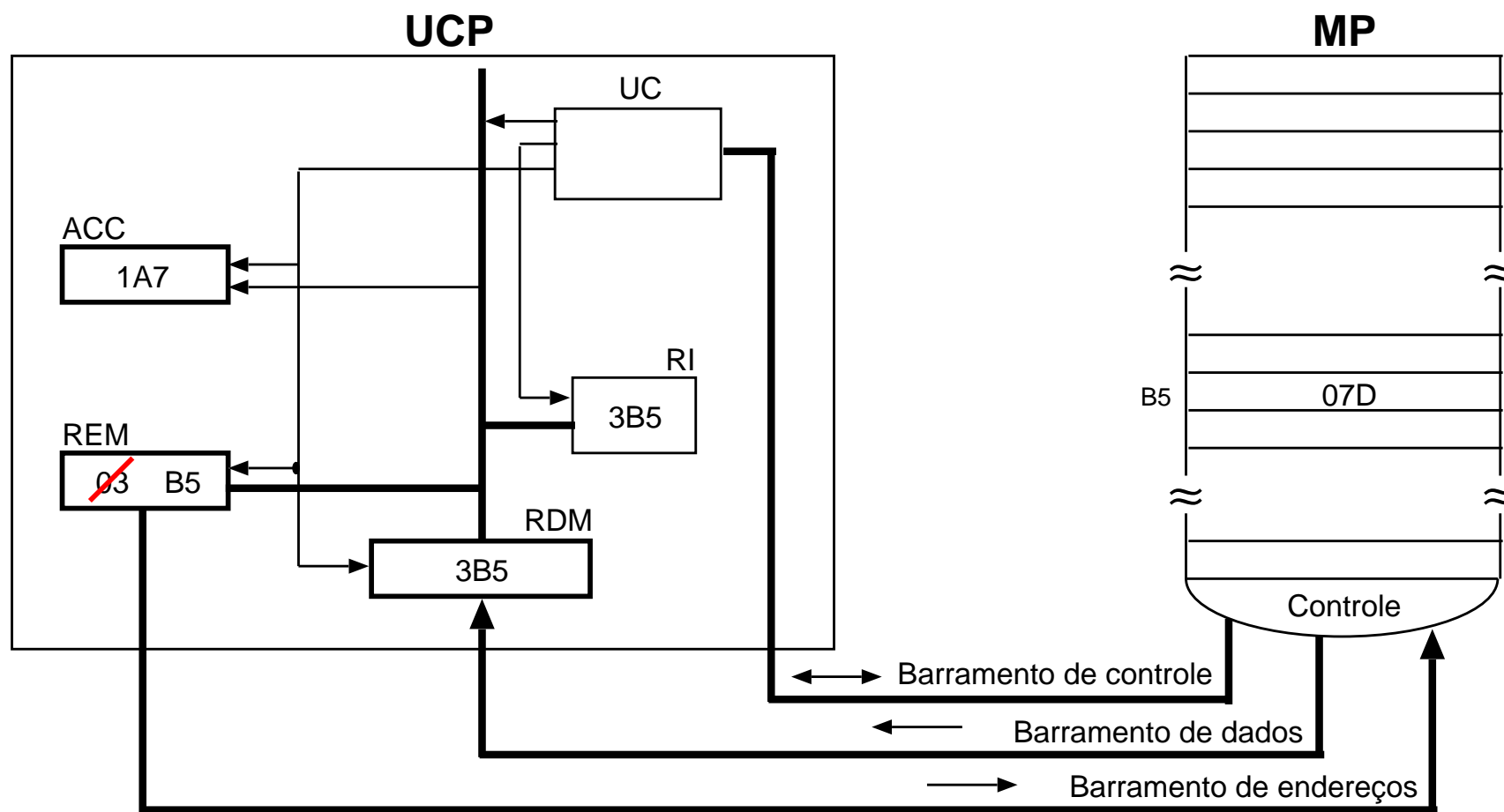
- Decodificação do código de operação
  - Decodificador recebe os 4 bits do código de operação
  - Decodificador gera sinais para que a unidade de controle produza os sinais para executar a operação definida pela instrução (soma)



(Fig. 6.24 do livro texto)

# Funcionamento da UCP: Execução de ADD

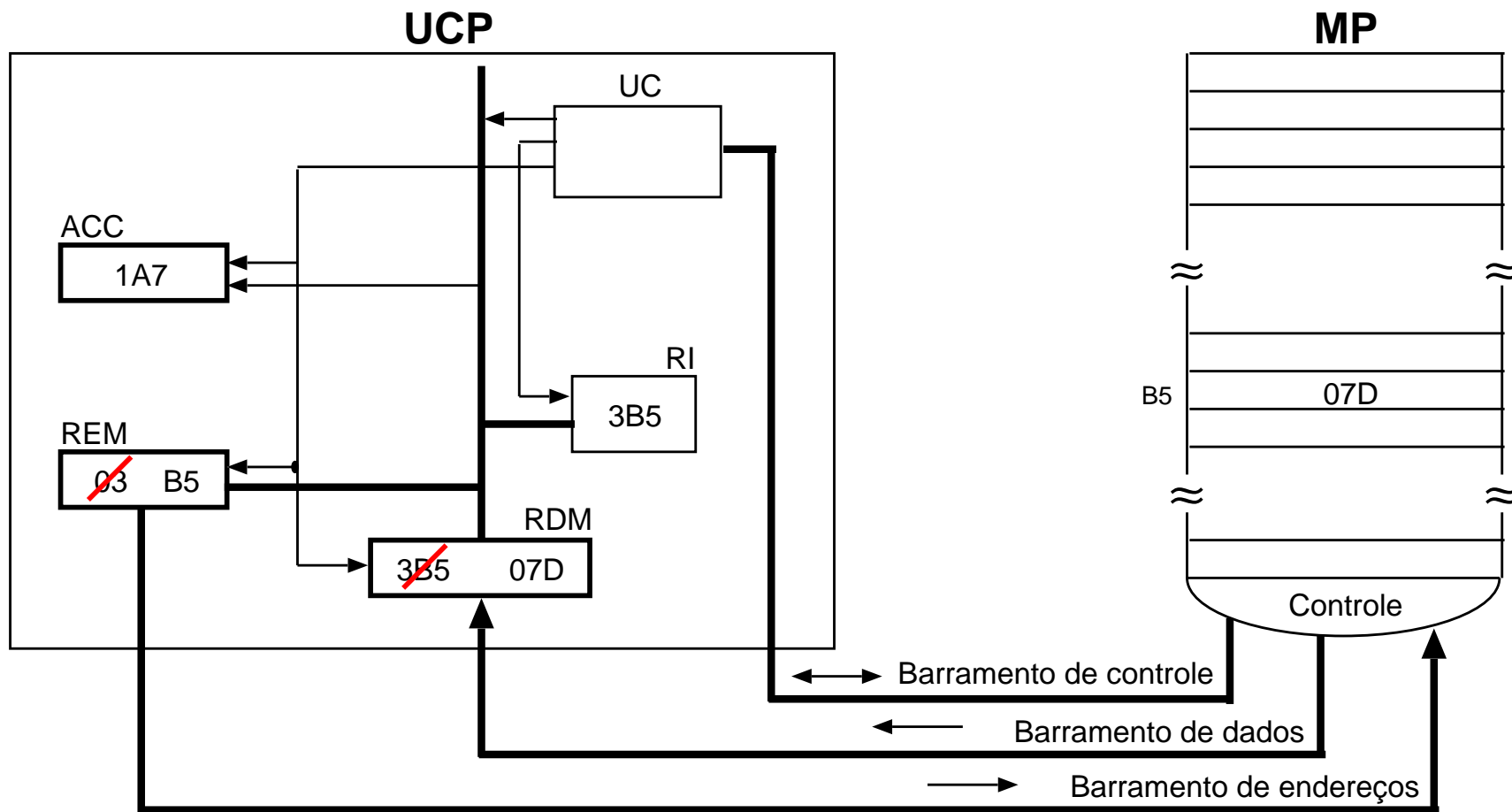
- Busca de operando na memória
  - A UC emite os sinais para que o valor do campo do operando (B5) seja transferido para o REM



(Fig. 6.25 do livro texto)

# Funcionamento da UCP: Execução de ADD

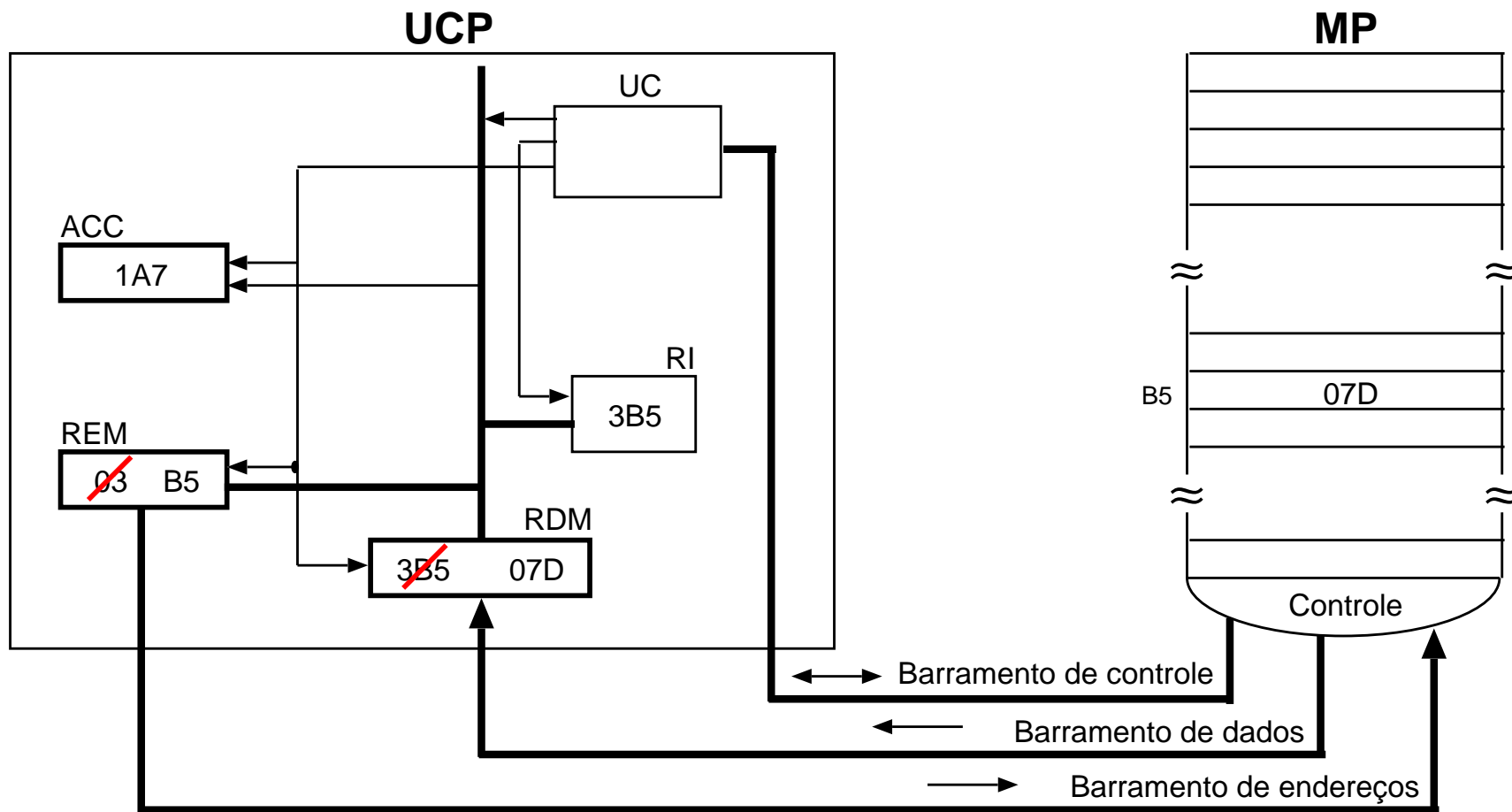
- Busca de operando na memória
  - A UC ativa a linha READ do barramento de controle
  - Conteúdo do endereço de memória B5 é transferido para o RDM



(Fig. 6.25 do livro texto)

# Funcionamento da UCP: Execução de ADD

- Busca de operando na memória
  - Dados a serem somados são transferidos para a UAL através do acumulador
  - $UAL \leftarrow ACC$

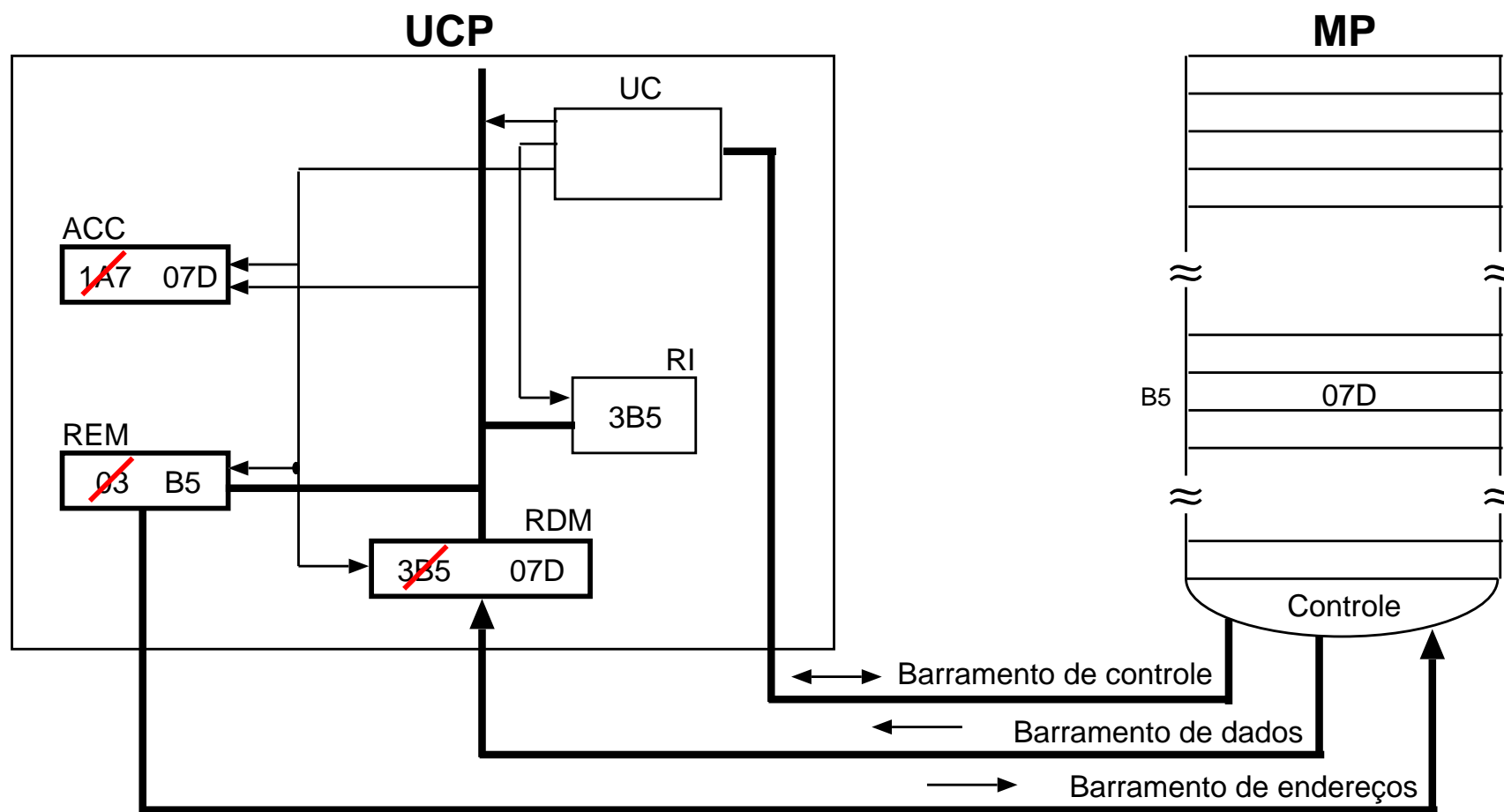


(Fig. 6.25 do livro texto)



# Funcionamento da UCP: Execução de ADD

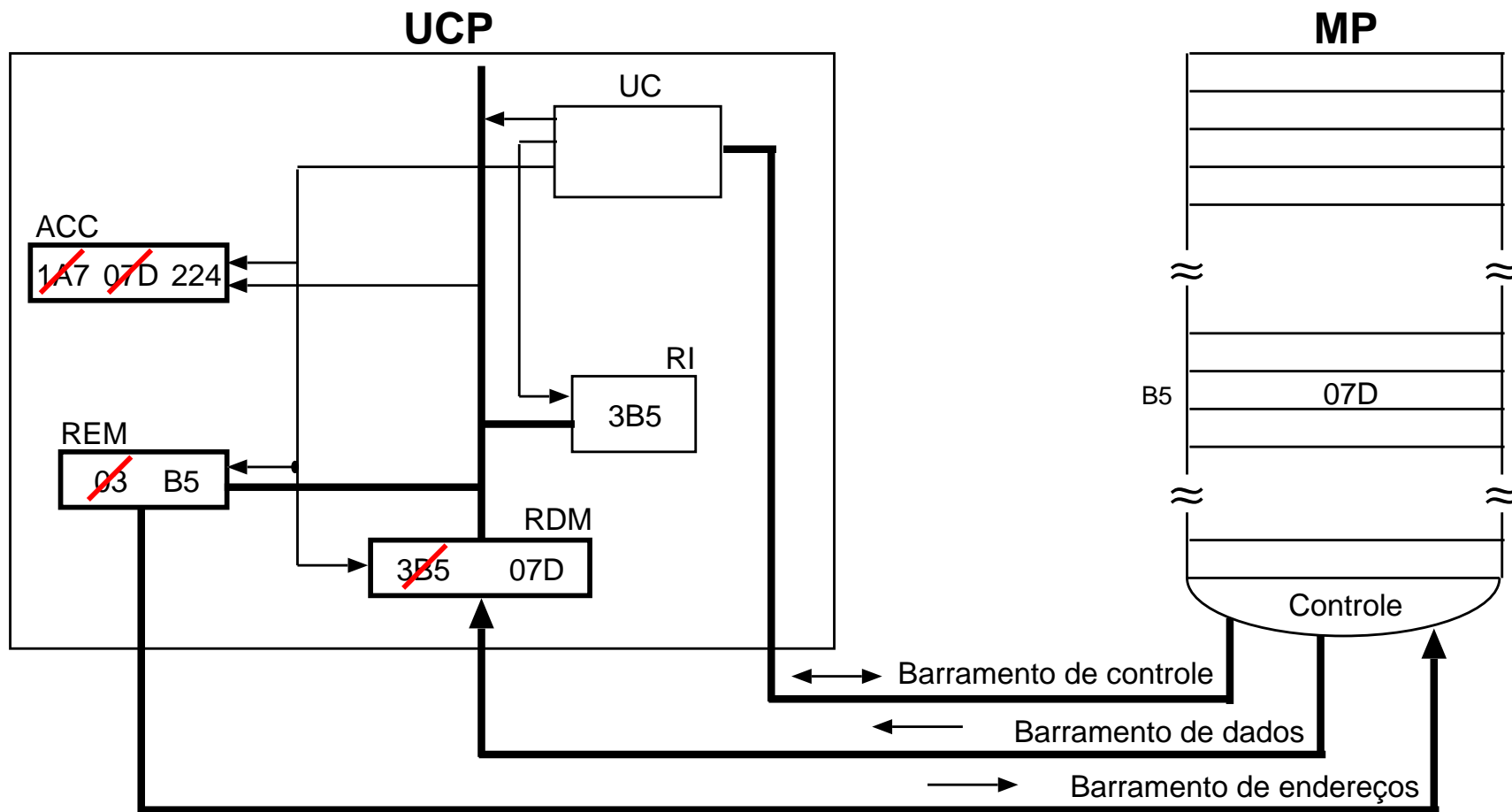
- Busca de operando na memória
  - $ACC \leftarrow RDM$
  - $UAL \leftarrow ACC$



(Fig. 6.25 do livro texto)

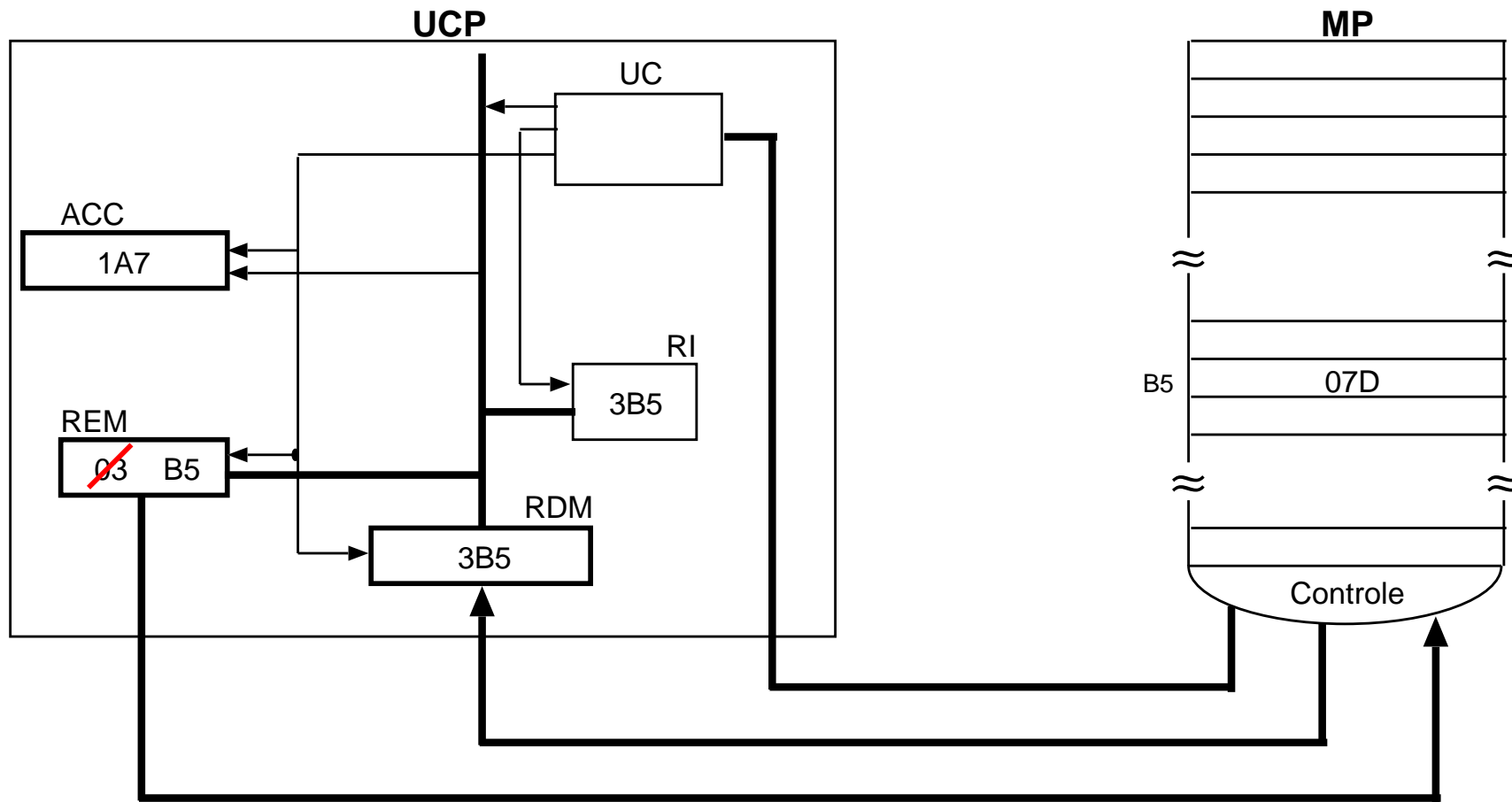
# Funcionamento da UCP: Execução de ADD

- Execução da operação
  - Valores são somados na UAL
    - $1A7 + 07D = 224$
  - Resultado armazenado no acumulador



(Fig. 6.25 do livro texto)

- 1 A UC emite os sinais para que o valor do campo do operando (B5) seja transferido para o REM
- 2 A UC ativa a linha READ do barramento de controle
- 3 Conteúdo do endereço de memória B5 é transferido para o RDM
- 4 Dados a serem somados são transferidos para a UAL através do acumulador
- 5  $UAL \leftarrow ACC$       6  $ACC \leftarrow RDM$       7  $UAL \leftarrow ACC$
- 8 Valores são somados na UAL -  $1A7 + 07D = 224$
- 9 Resultado armazenado no acumulador



## Linguagem de montagem

- A maneira mais direta de utilizar o hardware é através da linguagem de máquina (linguagem binária de 0s e 1s)
- Mais complicada e difícil de ser utilizada pelo programador

## Linguagem de montagem

- Exemplo de programa em Pascal

$X := A + B - C;$

- Programa convertido em linguagem de máquina

000100100011

001100100100

010000100101

001000100110

## Linguagem de montagem

- Valores binários podem ser substituídos por hexadecimais

123

324

425

226

## Linguagem de montagem

- Emprego de símbolos alfanuméricos ao invés de números

- $X := A + B - C;$

PROG SOMA

LDA A

ADD B

SUB C

STR X

## Linguagem de montagem

- Linguagem de símbolos alfabéticos denominada linguagem de montagem (Assembly)
  - Relação de 1:1 com linguagem de máquina
  - Mais facilidade de compreensão e manuseio
  - Necessita ser traduzida para a linguagem de máquina utilizando-se um montador (Assembler)



## Linguagem de montagem

- Linha de instrução da linguagem Assembly é composta geralmente de 4 campos

Rótulo	Operação	Operandos	Comentário
Soma	Proc C Value: Palavra		
	Mov	AX , 0	Inicializar, zerando

## Linguagem de montagem

- Linguagem utilizada para desenvolvimento de programas básicos (sistemas operacionais) e de controle
- Atualmente, linguagens de alto nível, como C, permitem manipulação de estruturas primitivas e possibilitam desenvolvimento de programas mais estruturados e menores

## Unidade Aritmética e Lógica

- UAL é constituída de circuitos dedicados a realizar:
  - operações aritméticas como soma, subtração, divisão e multiplicação
  - operações lógicas como AND, OR, NOT
  - deslocamentos dos bits de um número



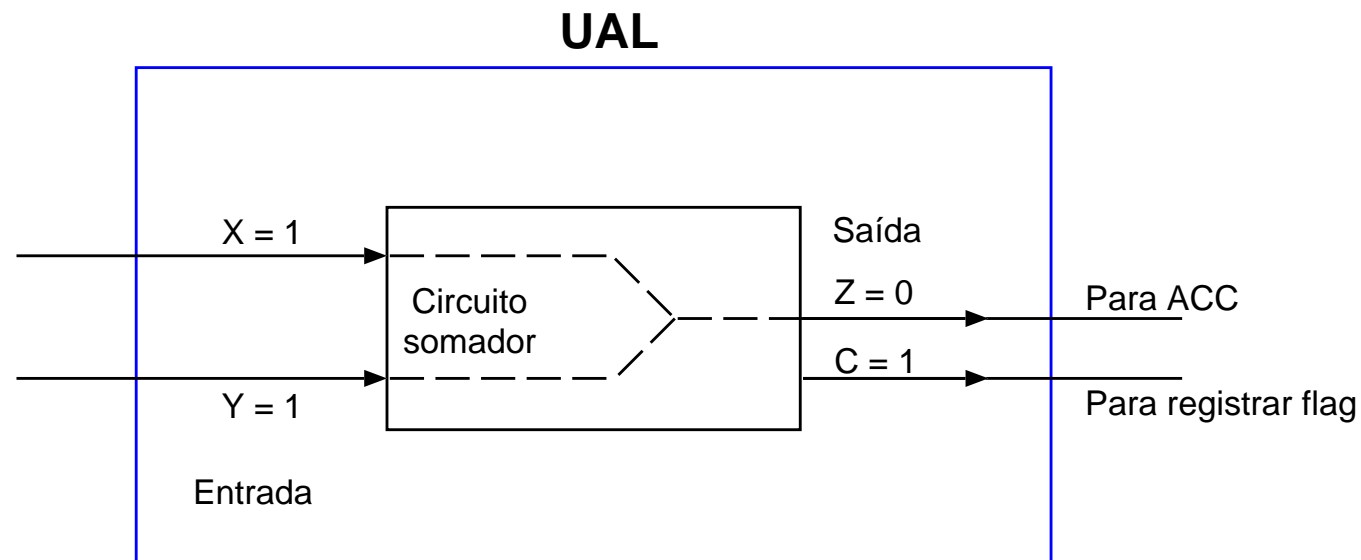
**Errata:** Deslocar o registrador na base 2 para a esquerda está multiplicando por dois e para a direita está dividindo por dois.

## Unidade Aritmética e Lógica

- Primeiros processadores possuíam um co-processador para efetuar operações com valores fracionários
- Co-processador de ponto flutuante
- Atualmente as UALs responsáveis pelas operações com números inteiros e fracionários estão integradas em uma mesma pastilha

# Unidade Aritmética e Lógica

- Funcionamento básico da UAL



X e Y - valores que serão somados ( $1 + 1 = 0$  e vai 1)  
Z - resultado da operação  $m = 0$   
C - bit "vai 1" = 1

(Fig. 6.31 do livro texto)

## Exercício 1

- Considere um computador com um conjunto de 128 instruções. Cada instrução é composta de 2 campos: o primeiro para indicar a operação e o segundo para indicar o endereço do operando. Supondo que sua memória tenha capacidade de armazenar 512 palavras e que cada instrução tem o tamanho de uma palavra e da célula de memória, pergunta-se:
  - Qual é o tamanho em bits do REM, RDM, RI e CI ?
  - Qual a capacidade de memória em bytes ?
  - Se o tamanho das instruções for alterado para 17 bits, mantendo inalterado o tamanho do REM, quantas novas instruções poderiam ser criadas ?

## Exercício 1 - Resposta

- Como sua memória tem capacidade de armazenar 512 palavras, o número de bits do endereço é  $\log_2 512 = \log_2 2^9 = 9$  bits
- O tamanho em bits do REM e do CI deve ser igual ao número de bits utilizado para o endereço, que é 9 bits
- Como o computador possui um conjunto de 128 instruções, o número de bits do código de operação é  $\log_2 128 = \log_2 2^7 = 7$  bits
- A instrução tem o tamanho igual ao número de bits do código de operação (7) somado ao número de bits do endereço (9), igual a 16 bits
- Como cada instrução tem o tamanho de uma palavra e da célula de memória, o tamanho da palavra é igual a 16 bits
- O tamanho em bits do RDM e do RI deve ser igual ao tamanho da palavra que é igual a 16
- A capacidade da memória em bytes é igual ao número de palavras multiplicado pelo tamanho da palavra:  $512 \times 16 \text{ bits} = 512 \times 2 \text{ bytes} = 1024 \text{ bytes}$  ou 1 Kbytes
- Se o tamanho das instruções for alterado para 17 bits, se mantendo inalterado o tamanho do REM, teremos um novo número de bits para o código de operação igual a  $17 - 9 = 8$  bits. Portanto podemos ter o dobro de instruções.

## Exercício 2

- Considere um sistema que possui uma memória com 256 células, sendo que cada célula pode armazenar 12 bits. Na figura abaixo são apresentados os endereços e conteúdos de algumas destas células

Endereço	Conteúdo
00	010
01	AFD
02	000
A4	123
A5	135
A6	B00
FD	440
FE	2F8
FF	0A5



## Exercício 2

- Qual a capacidade total de memória ?
- Supondo que, no início de um ciclo de instrução, o conteúdo do CI seja o hexadecimal A5 e que cada instrução ocupa uma única célula, qual instrução será executada, considerando o conjunto de instruções visto nesta aula ?
- Supondo que o conteúdo do REM tenha o valor hexadecimal FD e que um sinal de leitura seja enviado da UCP para a memória, qual deverá ser o conteúdo do RDM ao final do ciclo de leitura ?

## Exercício 2 - Resposta

- Como o computador possui 256 células com 12 bits cada uma, a capacidade total da memória é igual a  $256 \times 12 = 3072$  bits.
- Como o valor do CI é A5, a instrução a ser executada é aquela armazenada no endereço A5 que é igual a 135. Para podermos decodificar esta instrução convertamos para a base 2, obtendo 000100110101. Os 4 primeiros bits, 0001, indicam que a instrução é LDA. Os oito bits restantes indicam o endereço de memória do operando que é igual a 35 na base hexadecimal ou 53 na base decimal. Logo a instrução é LDA 53.
- Como o REM possui o valor FD, será lido o conteúdo da memória cujo endereço é FD. Este valor será colocado no RDM, logo RDM terá o conteúdo 440.

## Exercícios

- Capítulo 6 do livro texto

**11, 13, 14 e 17**

## **Aula 5**

### **Professores:**

Lúcia M. A. Drummond  
Simone de Lima Martins

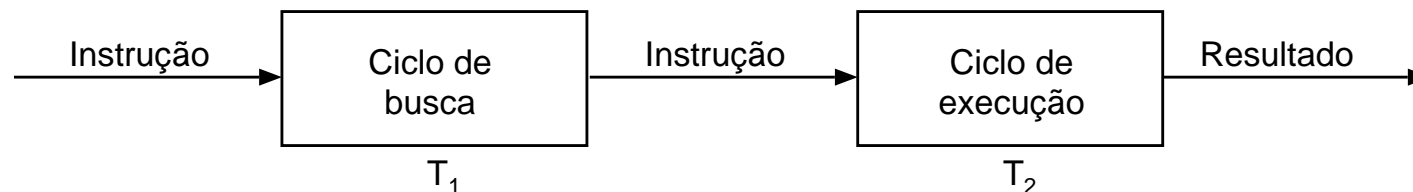
### **Conteúdo:**

#### **Unidade Central de Processamento 3**

- Linha de montagem ou pipelining
- Barramentos
- Implementação de controle

# Linha de montagem ou pipelining

- Etapas do ciclo de instrução são executadas de forma seqüencial
- Tempo total de execução da instrução é igual à soma dos tempos gastos em cada etapa



$T$  = tempo de execução da instrução =  $T_1 + T_2$

Ciclo de busca = leitura da instrução, incremento do CI

Ciclo de execução = decodificação, busca do operando, execução da operação

(Fig. 6.35 do livro texto)

# Linha de montagem ou pipelining

- Analogia com fabricação de um automóvel
  - Montagem da carroceria (T1)
  - Montagem do motor (T2)
  - Montagem das rodas (T3)
  - Instalação elétrica (T4)
  - Instalação de bancos e espelhos (T5)
  - Acabamento final (T6)
  - Tempo total  $T = T1 + T2 + T3 + T4 + T5 + T6$
  - Em um turno de X horas, um máximo de  $X/T$  carros podem ser montados

# Linha de montagem ou pipelining

- Metodologia inventada por Henry Ford no início do século XX
  - Dividir processo de fabricação em estágios independentes
  - Denominada linha de montagem (pipeline) pois um item pode iniciar sua fabricação antes que um item anterior termine sua montagem
  - Objetivo é possibilitar a fabricação de mais unidades por unidade de tempo
  - Não diminui o tempo de fabricação de uma unidade
  - Esta técnica é utilizada na arquitetura de unidades centrais de processadores

# Linha de montagem ou pipelining

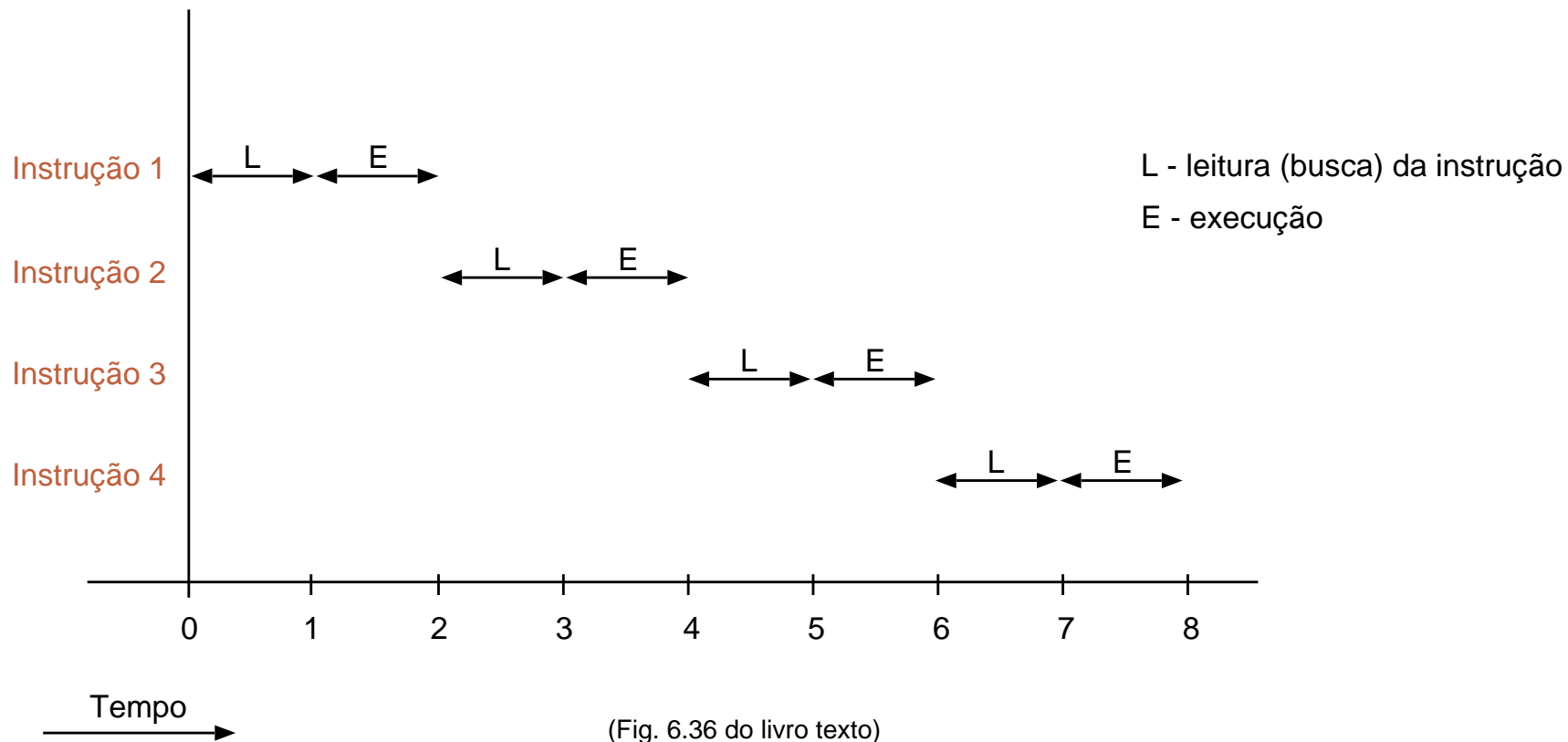
- Baseada em duas premissas básicas
  - O processo (fabricação de um automóvel ou execução de uma instrução) tem que ser dividido em estágios de realização independentes um do outro
  - Um novo produto (carro ou instrução) inicia seu processo de fabricação ou execução antes do anterior concluir seu processo



# Linha de montagem ou pipelining

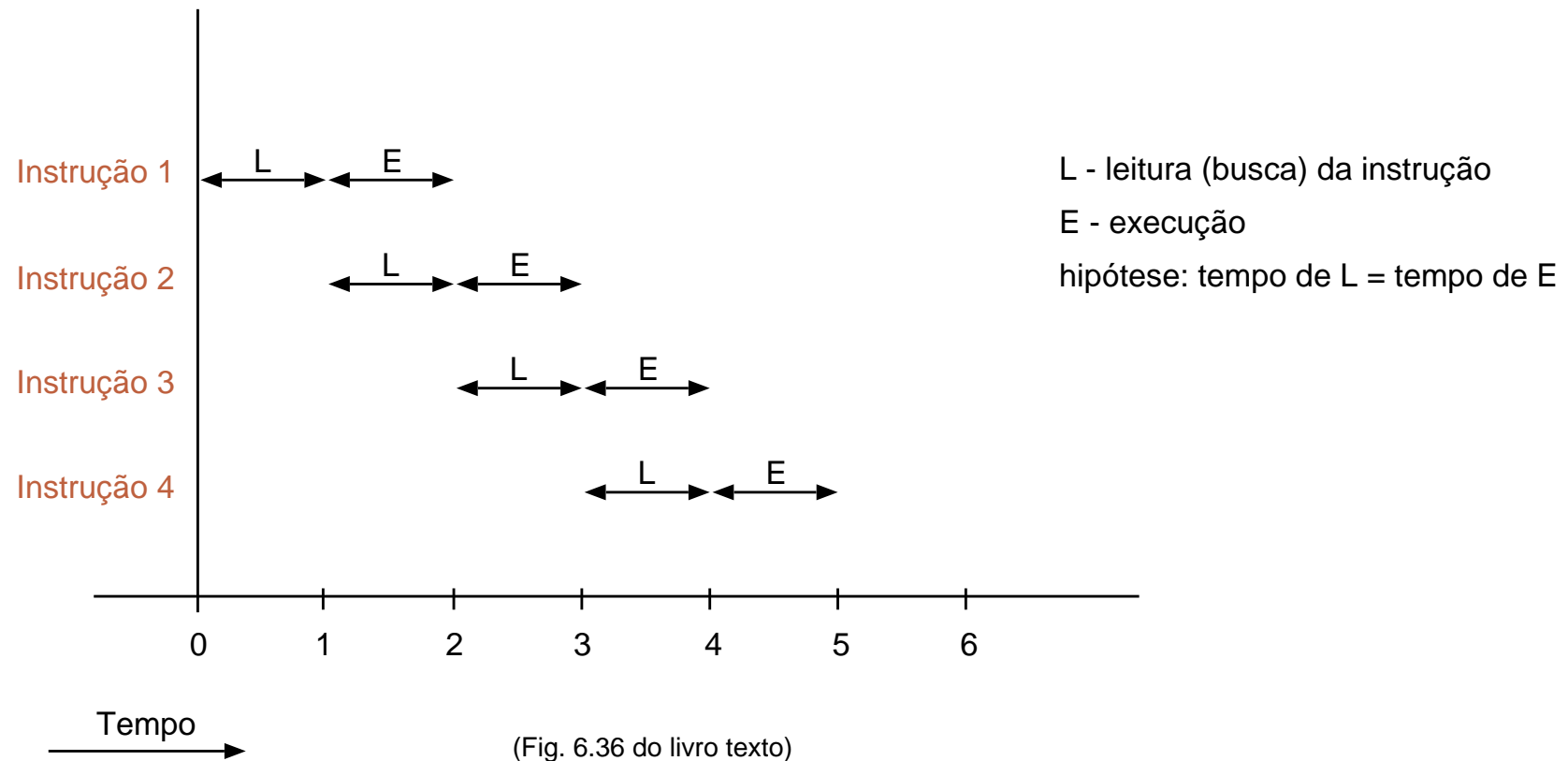
- Implementação de pipelining com divisão do ciclo de instrução em dois estágios
  - Busca da instrução
    - Acesso à memória
  - Execução da instrução
    - Nem sempre é necessário acessar a memória (decodificação ou execução da instrução)
- Quando não há atividade de acesso à memória no estágio de execução, pode-se ativar o estágio de busca da instrução

# Linha de montagem ou pipelining



- Supondo que cada estágio gaste o mesmo período de tempo  $T$ 
  - Para executar as 4 instruções de forma seqüencial seriam gastos  $8T$

# Linha de montagem ou pipelining



- Supondo que cada estágio gaste o mesmo período de tempo  $T$ 
  - Para executar as 4 instruções de forma seqüencial seriam gastos  $8T$
  - Na estrutura em pipelining se gasta  $5T$
  - Mas cada instrução gasta  $2T$

# Linha de montagem ou pipelining

- Razões para não haver aumento significativo de produtividade em pipelining com 2 estágios
- Tempo de realização do estágio de execução em geral é maior do que o tempo de busca (acesso à memória e execução de operações). Então o estágio de busca pode ter de esperar algum tempo para poder enviar a instrução para o estágio de execução
- A ocorrência de instruções de desvio condicional faz com que o endereço da próxima instrução a ser buscada seja desconhecido. Então estágio de busca tem que esperar a finalização do estágio de execução

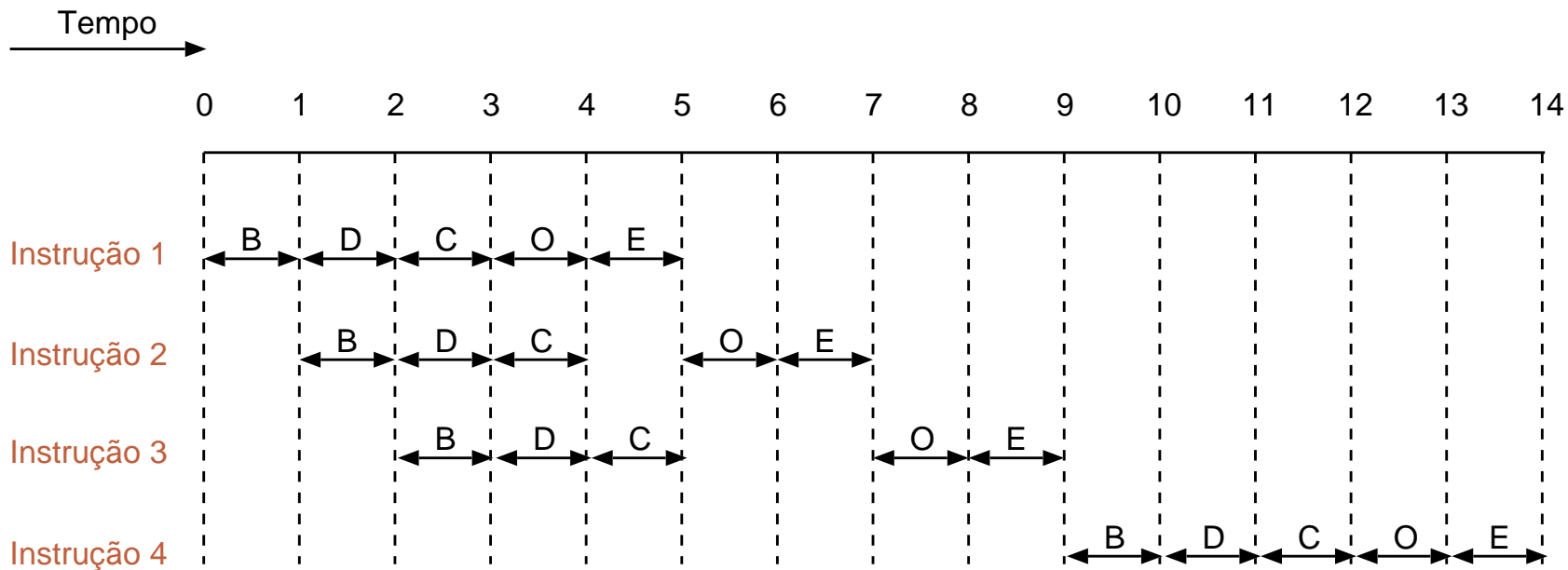
# Linha de montagem ou pipelining

- Para se obter maior produtividade o pipelining deve ser dividido em mais estágios, pois teremos uma maior probabilidade de obter tempos iguais em cada estágio
- Divisão de cada instrução em 5 estágios com duração igual
  - Busca da instrução (B)
  - Decodificação da instrução (D)
  - Cálculo do endereço de operandos (C)
  - Busca dos operandos (O)
  - Execução da operação (E)

# Linha de montagem ou pipelining

- Considerações sobre a arquitetura que implementa o pipelining
  - Somente um acesso à memória pode ser realizado de cada vez, ou seja, durante a execução do estágio de busca de instrução (B), o estágio de busca de operandos (O) não pode ser realizado
  - Sempre existe acesso à memória no estágio de execução
  - Todos os estágios são realizados em um período de tempo igual

# Linha de montagem ou pipelining



(Fig. 6.37 do livro texto)



- Tempo de execução seqüencial: 20 unidades de tempo
- Tempo de execução com pipelining: 14 unidades de tempo

# Linha de montagem ou pipelining

- Instrução de desvio condicional

Exemplo:

- JP OP
  - Se  $ACC > 0$ , então  $PC \leftarrow OP$
- Não há meios de se conhecer o endereço da próxima instrução antes do teste da condição definida na instrução
  - Se condição verdadeira, endereço definido na instrução
  - Se condição falsa, endereço é o seguinte



# Linha de montagem ou pipelining

- Soluções para diminuir o atraso no fluxo de processamento de instruções, devido à instrução de desvio condicional
- Estágio de busca obtém na memória a instrução armazenada após a instrução de desvio. Caso não haja desvio, nenhum tempo foi perdido. Caso contrário, descarta-se a instrução buscada e busca-se a nova instrução
- Busca da instrução armazenada após a instrução de desvio e da instrução armazenada no endereço do desvio

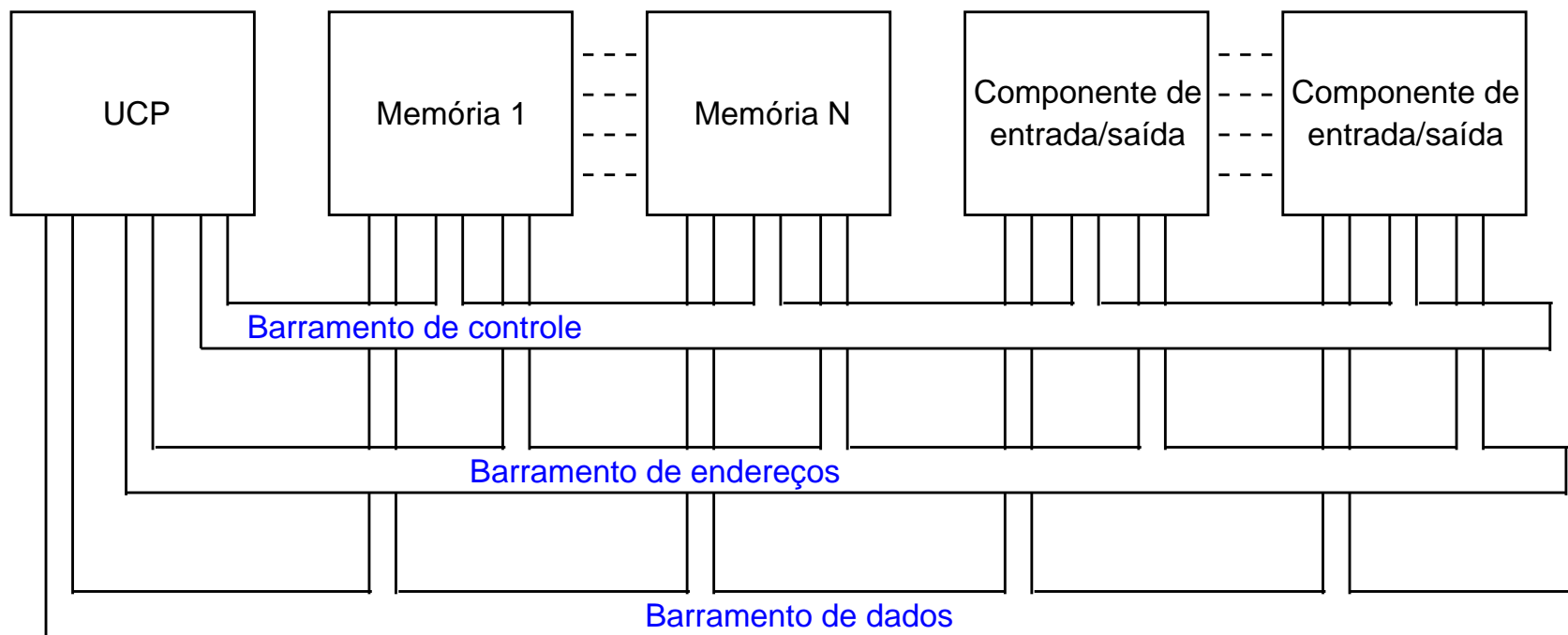
# Linha de montagem ou pipelining

- Soluções para diminuir o atraso no fluxo de processamento de instruções, devido à instrução de desvio condicional
- Sistema tenta prever se desvio vai ocorrer ou não
  - Prever que o desvio nunca será tomado
  - Prever que o desvio sempre será tomado
  - Prever se o desvio será tomado baseado no código de operação
  - Prever o desvio com base em uma tabela de histórico de desvios para cada instrução de desvio

# Barramento

- O barramento de um sistema de computação é o elemento responsável pela interligação dos demais componentes, conduzindo de modo sincronizado o fluxo de informações (dados, endereços e sinais de controle) entre eles de acordo com uma programação de atividades previamente definida pela unidade de controle

# Barramento



(Fig. 6.38 do livro texto)

# Barramento

- Barramento de dados
  - Múltiplas linhas condutoras sendo que cada uma permite a passagem de um bit de informação
  - Diferentes quantidades de bits: 8, 16, 32, 64 e 128
- Barramento de endereços
  - Utilizado pelo processador para indicar de onde um dado deve ser lido e onde um dado deve ser escrito
  - Quantidade de bits especifica a quantidade máxima de memória principal
  - O valor binário colocado neste barramento pode indicar um endereço de dispositivo de entrada e saída

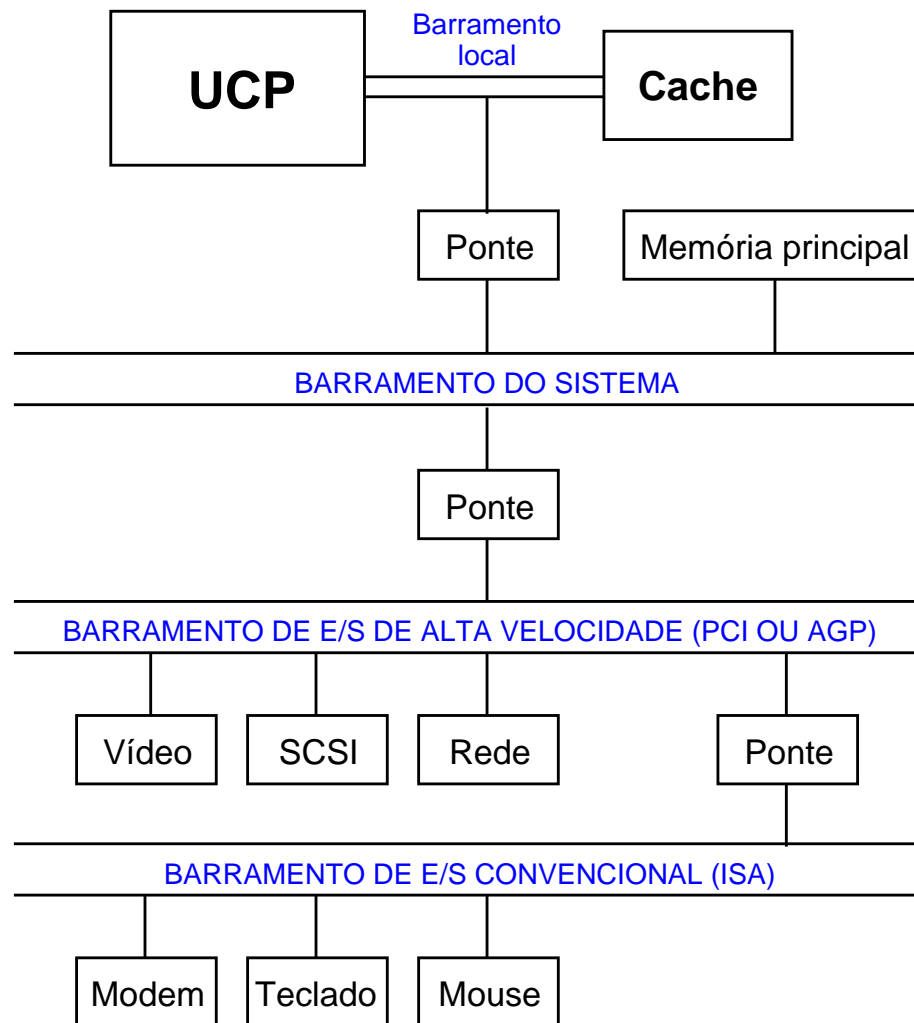
# Barramento

- Barramento de controle
  - Constituído de linhas por onde fluem sinais específicos da programação do sistema
    - Leitura de dados (memory read)
    - Escrita de dados (memory write)
    - Leitura de E/S (I/O read)
    - Escrita de E/S (I/O write)
    - Certificação de transferência de dados (ACK)
    - Pedido de interrupção (interrupt request)
    - Relógio (clock)

# Barramento

- Arquiteturas modernas não utilizam barramentos compartilhados por todos os componentes
- Características diferentes entre os diversos componentes levaram à criação de diversos tipos de barramento
- Cada barramento possui taxas de transferência de bits diferentes e apropriadas às velocidades dos componentes conectados, sendo organizados de forma hierárquica

# Barramento



(Fig. 6.40(b) do livro texto)



# Barramento

- Largura do barramento
  - Caracteriza a quantidade de informação (número de bits, em geral) que pode fluir simultaneamente no barramento
- Ciclo de tempo do barramento
  - Tempo requerido para mover um grupo de bits (cujo tamanho é definido pela largura do barramento) ao longo do barramento

# Barramento

- Barramentos são compartilhados pelos componentes
- Compartilhamento implica na necessidade de definição de regras explícitas de acesso ao barramento
  - Quando acessar
  - Como acessar
- Compartilhamento implica na necessidade de definição de regras explícitas de comunicação
  - Como interrogar um componente destinatário
  - Qual resposta deve ser enviada
  - Quanto dura a comunicação

# Barramento

- Protocolos de barramento foram criados para definir estas regras de acesso ao barramento
  - ISA - Industry Standard Adapter
    - Adotado para periféricos de baixa velocidade
  - PCI - Peripheral Component Interconnect
    - Permite várias larguras de barramento e velocidades de transmissão
    - Quase um padrão no mercado
  - USB - Universal Serial Bus
    - Pode-se conectar muitos dispositivos simultaneamente
  - AGP - Accelerated Graphics Port

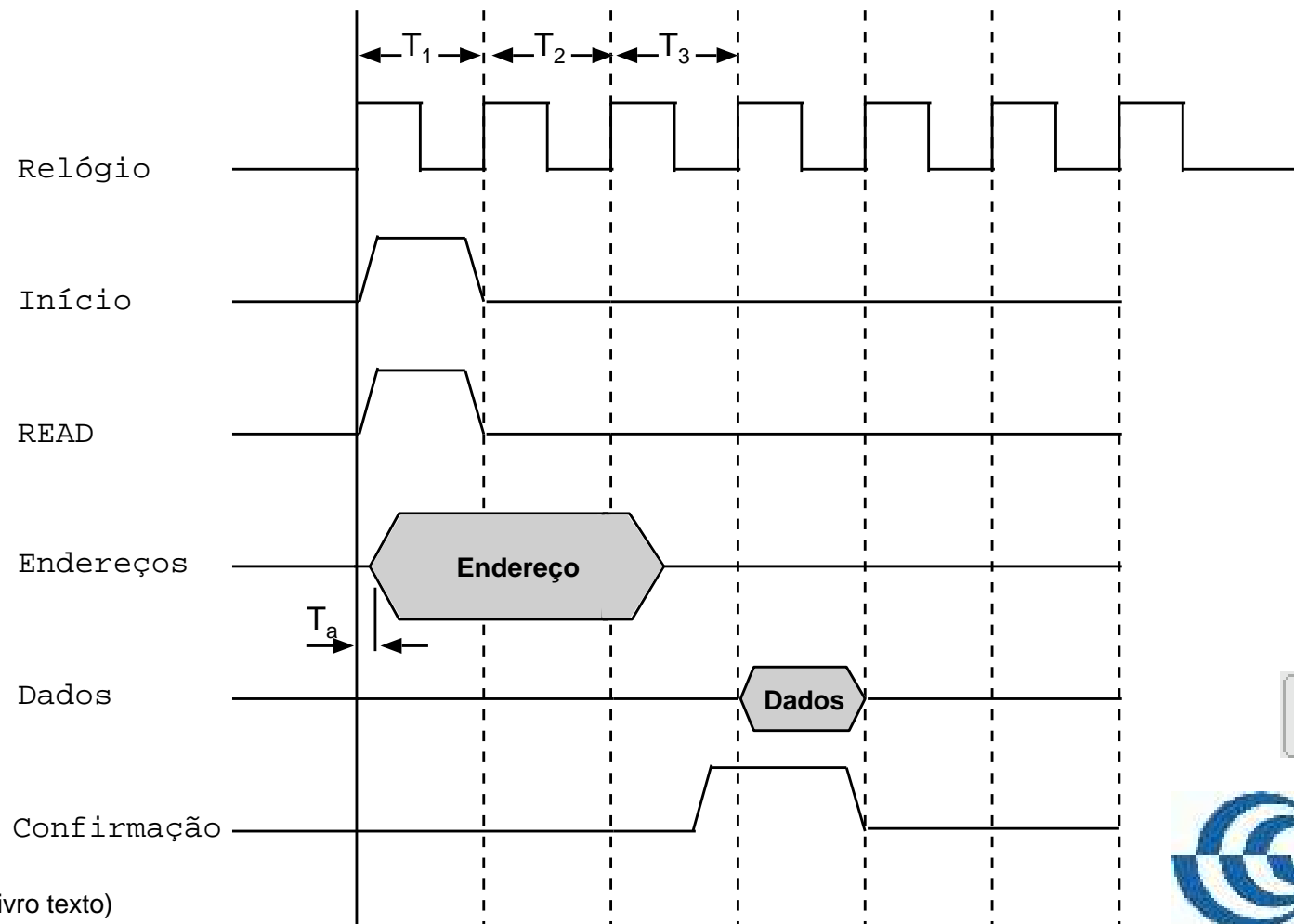
# Barramento

- Projeto do barramento
  - Método de controle do acesso ao barramento
    - Mestre/escravo
      - Um mestre é o único que pode acessar o barramento
      - Toda comunicação é realizada via mestre, que pode causar um gargalo no sistema
    - Acesso direto
      - Todos os componentes podem acessar o barramento
      - Requer maior complexidade nos circuitos de controle do barramento

# Barramento

- Projeto do barramento
  - Tipo de sincronização

## Operação síncrona

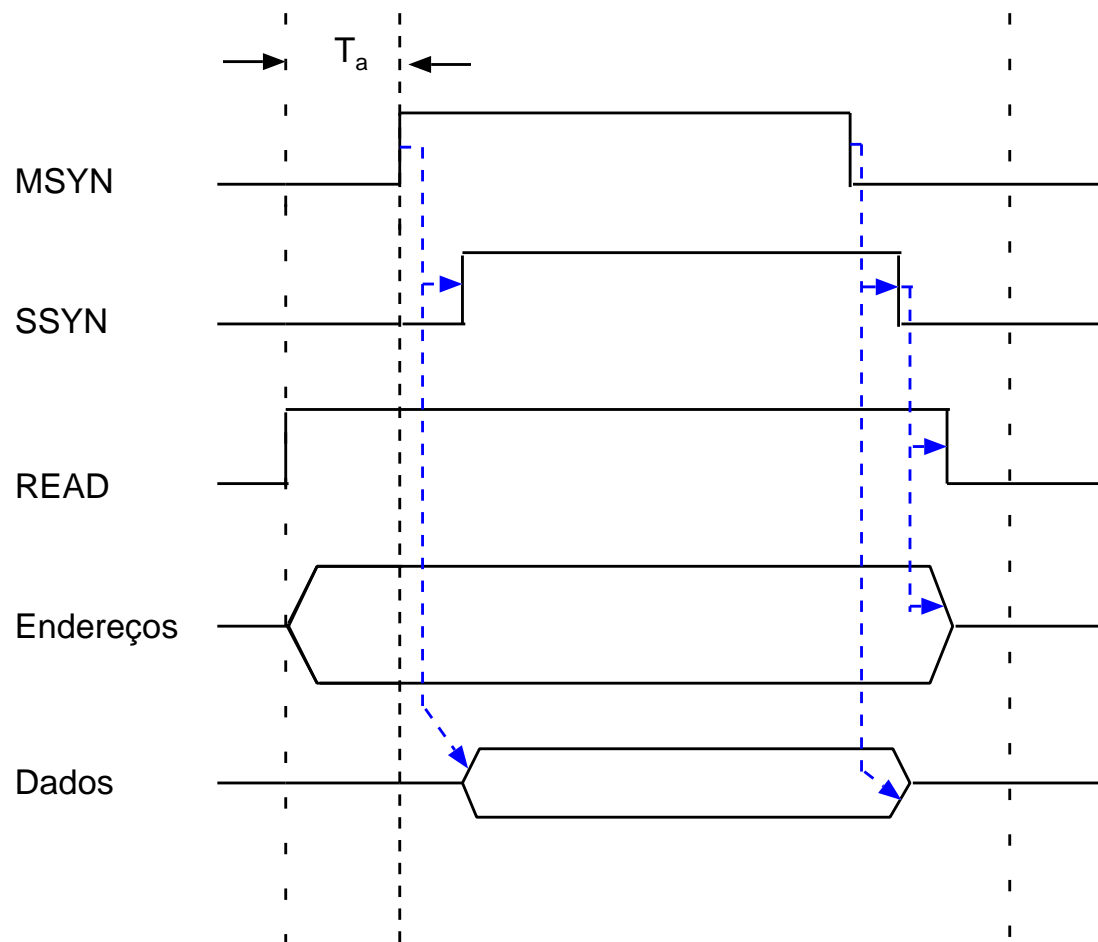


(Fig. 6.41 do livro texto)

# Barramento

- Projeto do barramento
  - Tipo de sincronização

Operação assíncrona

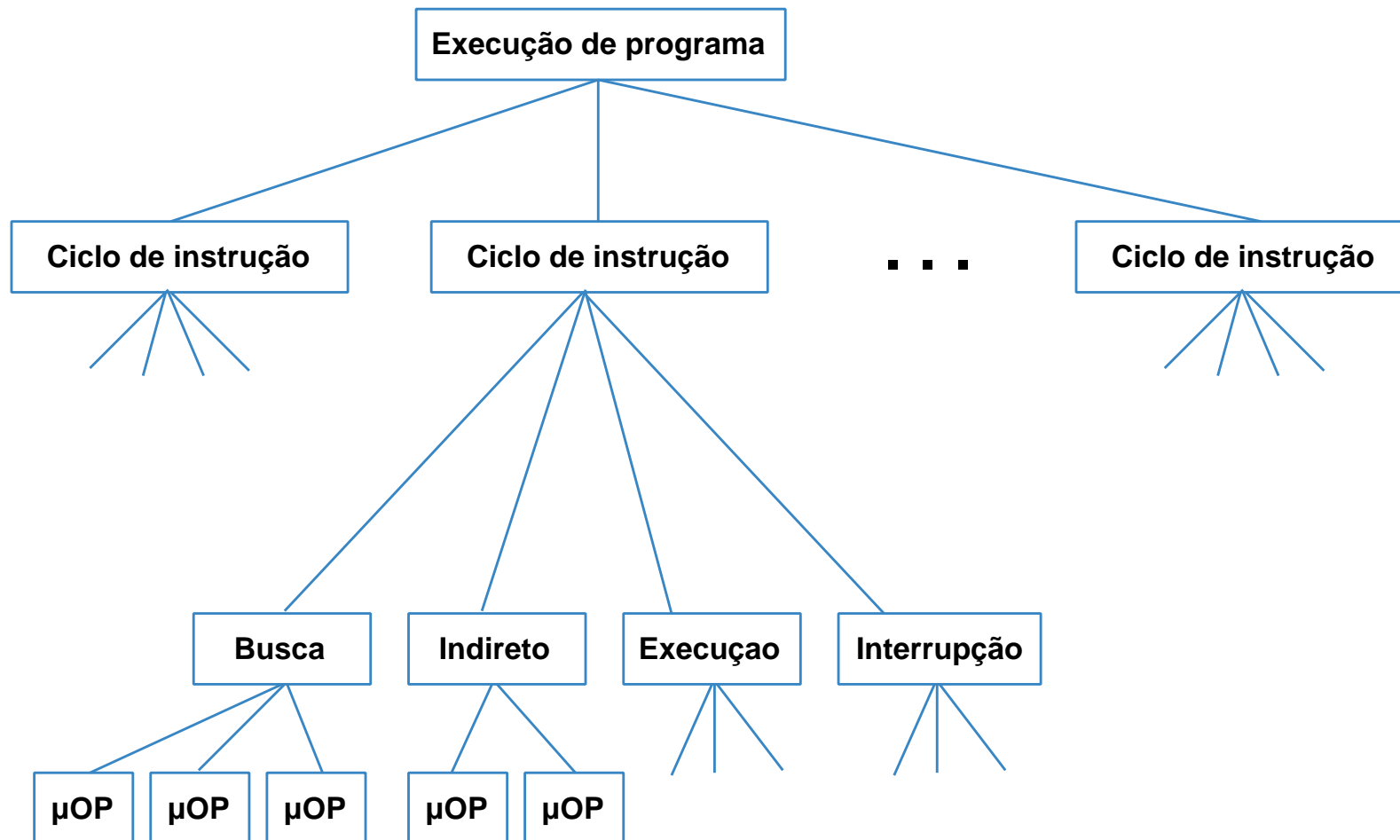


(Fig. 6.42 do livro texto)

# Barramento

- Projeto do barramento
  - Sincronização síncrona X assíncrona
    - Barramento síncrono é fácil de implementar e testar
    - Barramento síncrono pode ter problemas ao trabalhar com dispositivos com tempos de transferência diferentes
    - Barramento assíncrono pode interligar componentes de diversas velocidades e tecnologias

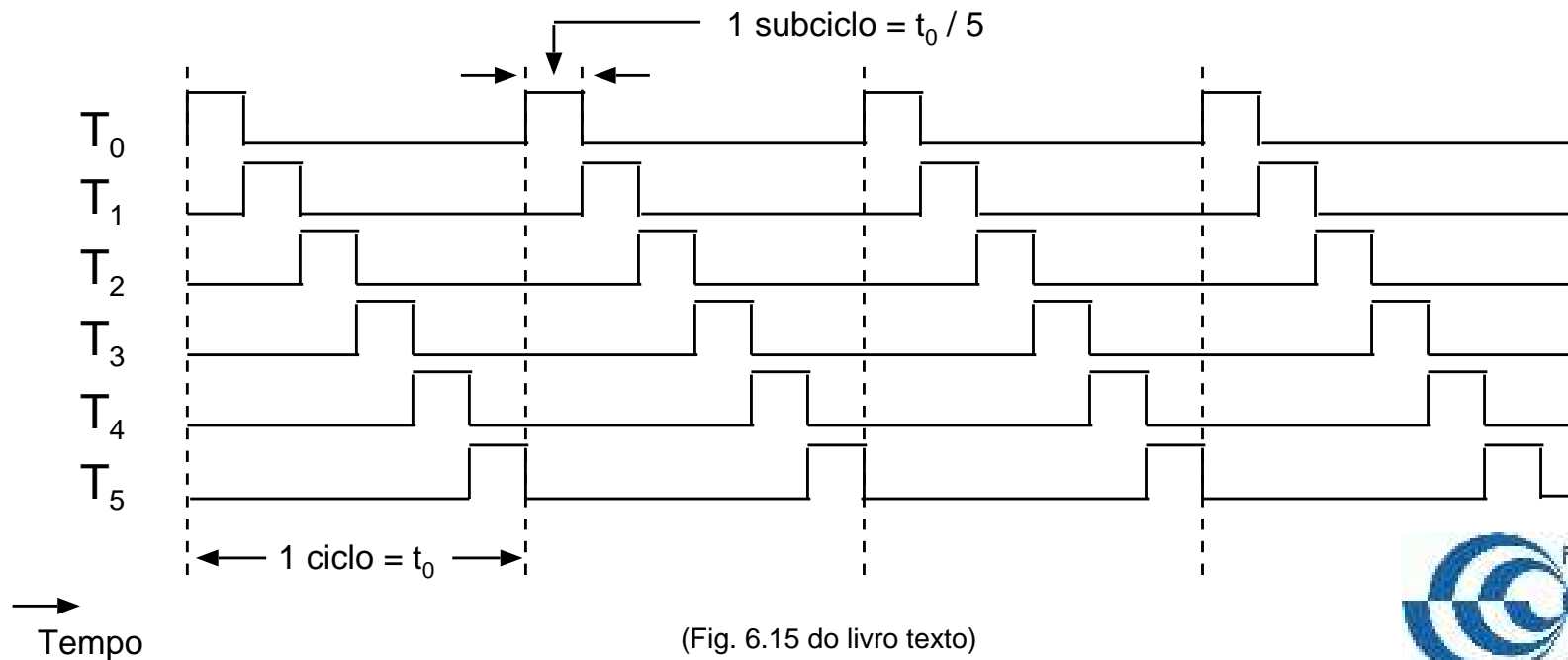
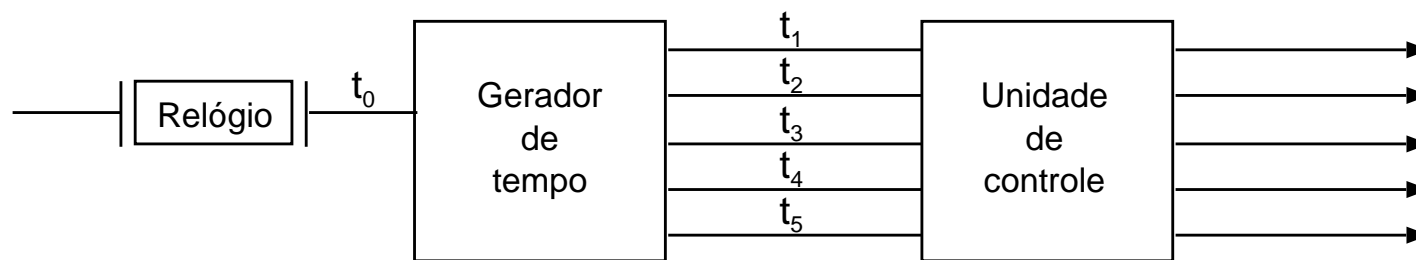
# Implementação de controle





# Implementação de controle

- O conjunto de  $\mu$ operações de cada subciclo do ciclo de instrução (busca, indireto, execução, interrupção) deve ser executado em um ciclo de relógio



(Fig. 6.15 do livro texto)

# Implementação de controle

- Ciclo de busca

t1: REM ← (CI)

t2: RDM ← Memória

CI ← CI+1

t3: RI ← (RDM)

# Implementação de controle

- Ciclo de execução
- ADD X (Soma do conteúdo do endereço X com o conteúdo do acumulador)

t1: REM  $\leftarrow$  (RI(Endereço))

t2: RDM  $\leftarrow$  Memória

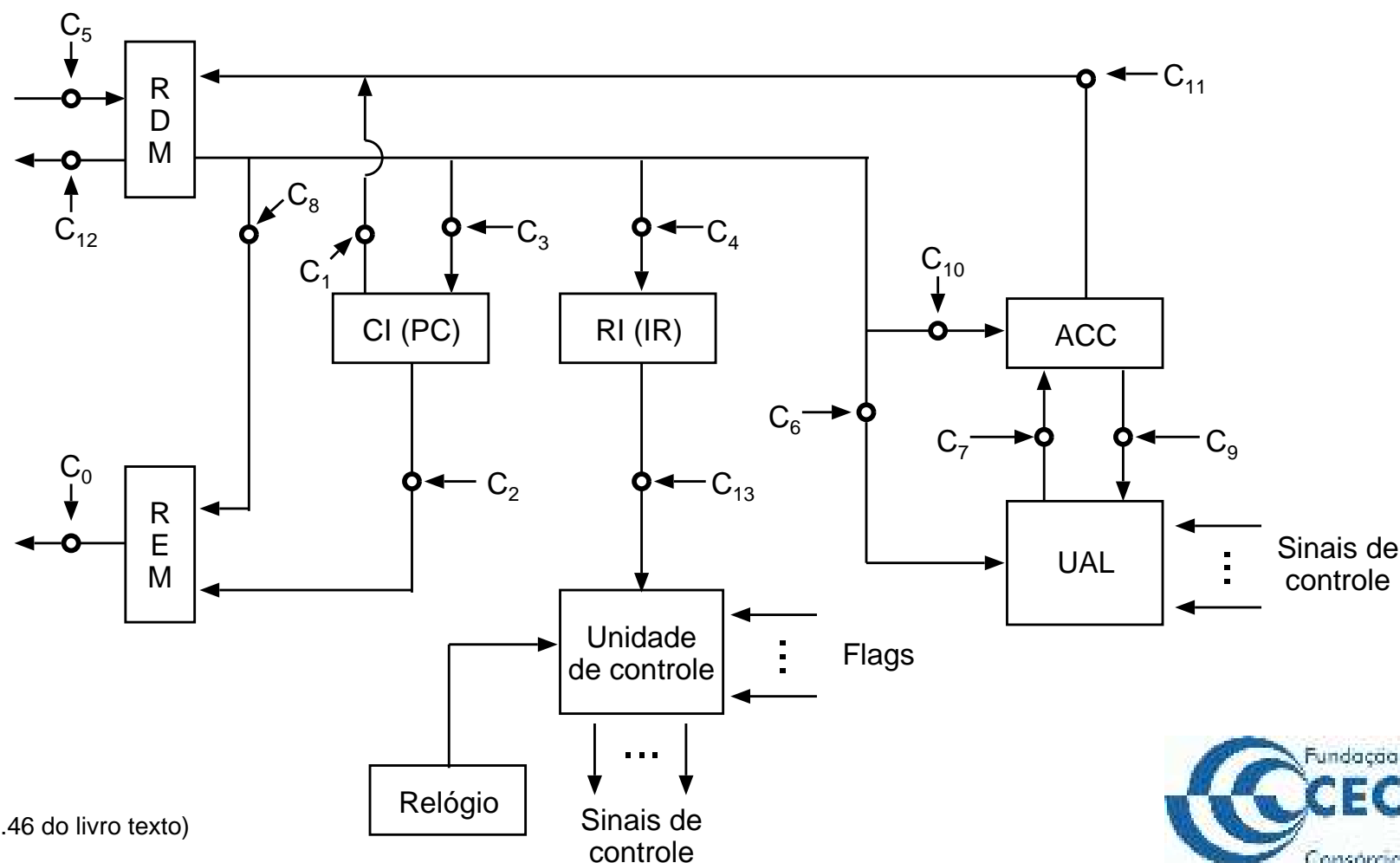
t3: ACC  $\leftarrow$  (ACC) + (RDM)

# Implementação de controle

- Unidade de controle deve ser construída contendo a programação da geração dos sinais de controle internos e externos à UCP conforme a instrução que será decodificada
- Duas maneiras utilizadas no projeto de uma UC
  - Controle programado diretamente no hardware (hardwired control)
  - Controle por microprogramação

# Implementação de controle

- Controle programado diretamente no hardware (hardwired control)
  - Unidade de controle é construída como um conjunto de circuitos logicamente combinados
  - Sinais de controle



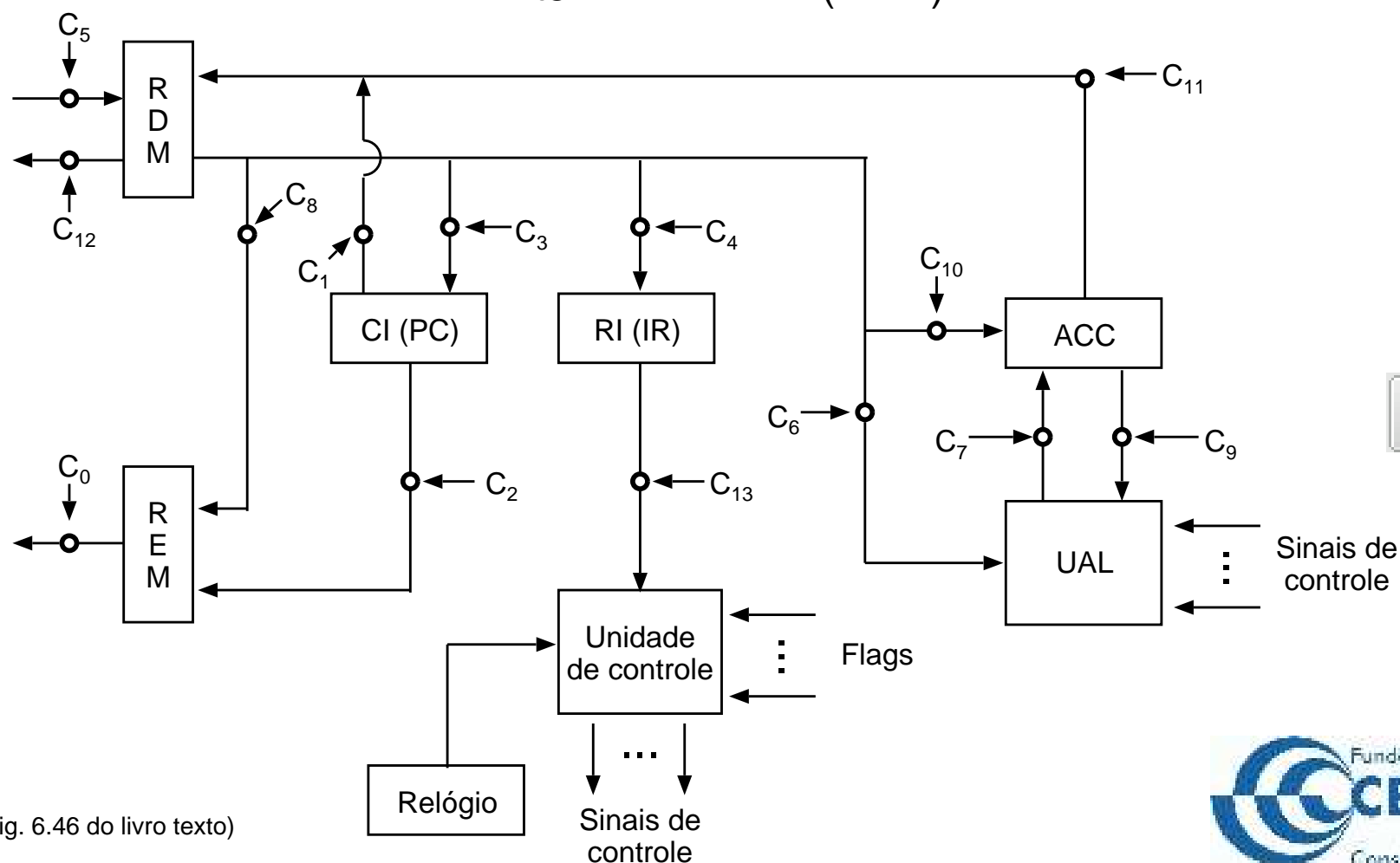
(Fig. 6.46 do livro texto)

# Implementação de controle

- Sinais de controle

- Ciclo de busca

t1: REM ← (CI)  
 t2: RDM ← Memória  
 CI ← CI+1  
 t3: RI ← (RDM)



(Fig. 6.46 do livro texto)

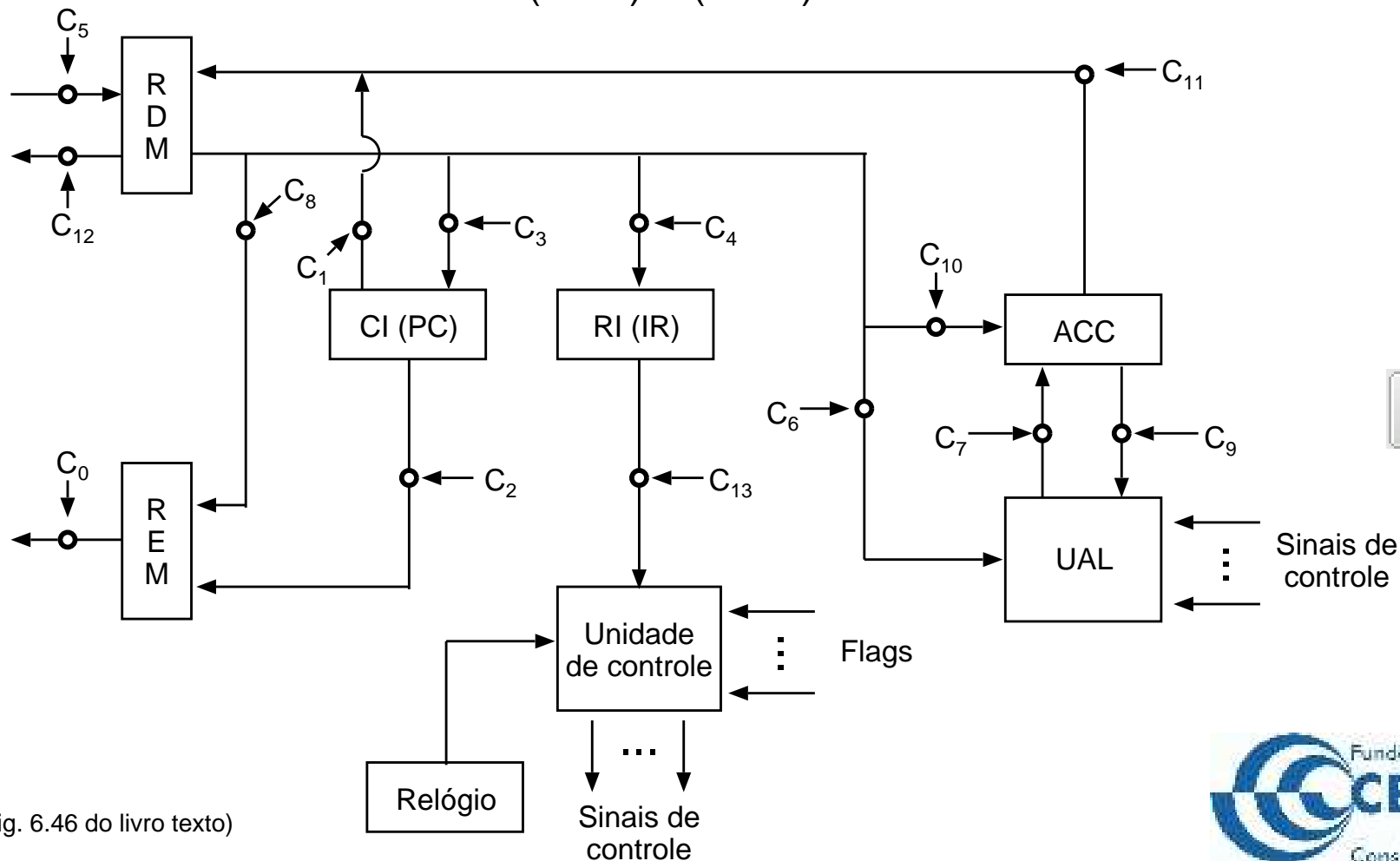
# Implementação de controle

- Sinais de controle
  - Ciclo de execução (instrução ADD X)

t1: REM  $\leftarrow$  (RI(Endereço))

t2: RDM  $\leftarrow$  Memória

t3: ACC  $\leftarrow$  (ACC) + (RDM)



(Fig. 6.46 do livro texto)

# Implementação de controle

Subciclo	Temporização	Sinais de controle ativos
Busca:	t1: $REM \leftarrow (CI)$ t2: $RDM \leftarrow Memória$ $CI \leftarrow CI+1$ t3: $RI \leftarrow (RDM)$	C2 C0, C5 e READ  C4
Execução:	t1: $REM \leftarrow (RI(endereço))$ t2: $RDM \leftarrow Memória$ t3: $ACC \leftarrow (ACC) + (RDM)$	C4, C8 C0, C5 e READ C6, C7, C9

Sinais PQ = 00 Subciclo de busca

PQ = 01 Subciclo indireto

PQ = 10 Subciclo de execução

PQ = 11 Subciclo de interrupção

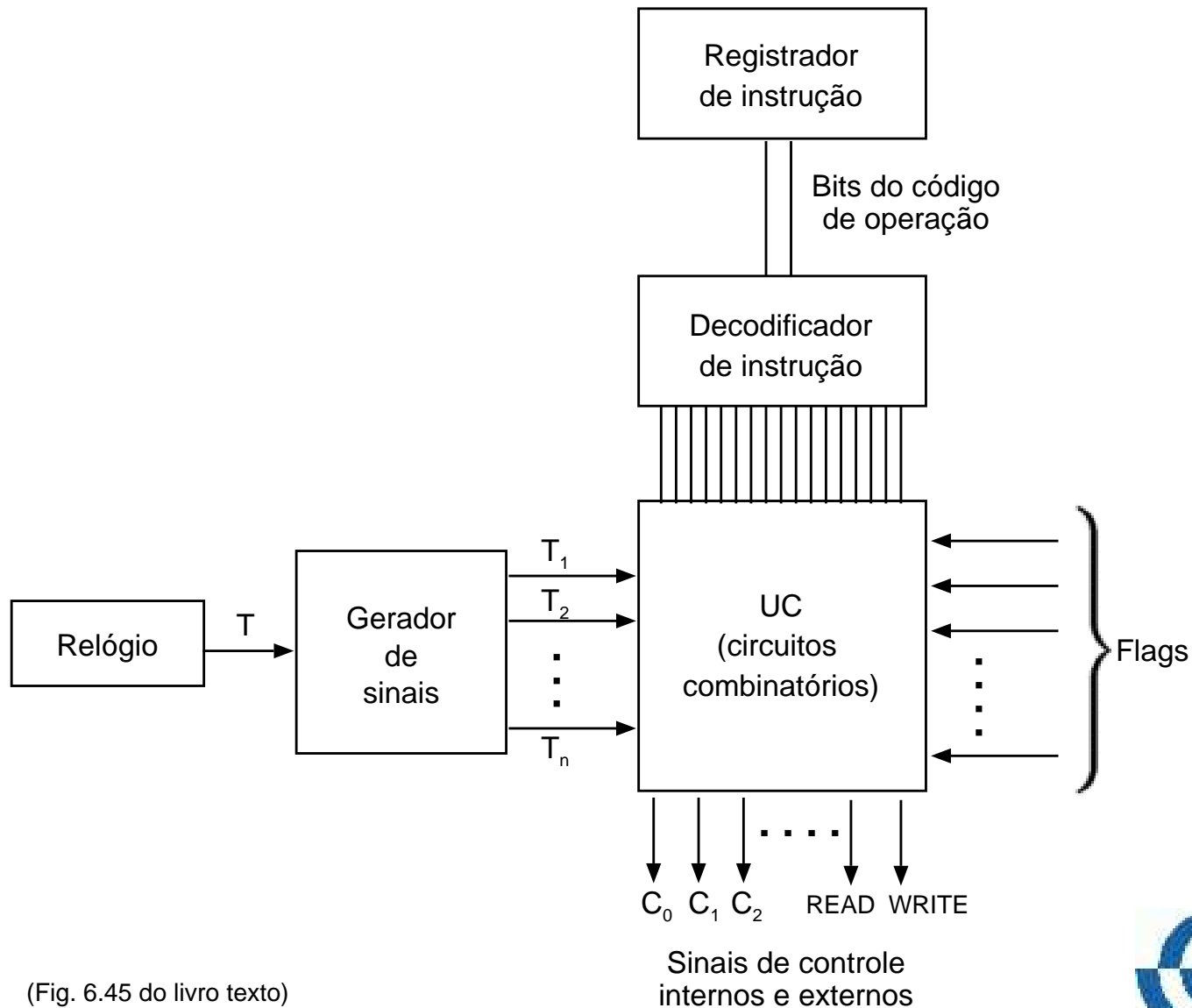
Sinal C5: ativo em Busca e  
Execução de ADD, em t2

$$C5 = \overline{P} \cdot \overline{Q} \cdot T2 + P \cdot \overline{Q} \cdot (LDA + ADD) \cdot T2$$



# Implementação de controle

- Esquema de um sistema de controle programado diretamente no hardware



(Fig. 6.45 do livro texto)

# Implementação de controle

- Controle programado diretamente no hardware
  - Número de equações booleanas requeridas pode ser muito grande
  - Implementar um circuito combinatório que satisfaça estas equações pode se tornar extremamente difícil
  - Abordagem utilizada para simplificar o projeto da unidade de controle: microprogramação

# Implementação de controle

- Controle por microprogramação
  - Conceito desenvolvido por Wilkes em 1951

Subciclo	Temporização	Sinais de controle ativos
Busca:	t1: $REM \leftarrow (CI)$ t2: $RDM \leftarrow Memória$ $CI \leftarrow CI+1$ t3: $RI \leftarrow (RDM)$	C2 C0, C5 e READ  C4
Execução:	t1: $REM \leftarrow (RI(endereço))$ t2: $RDM \leftarrow Memória$ t3: $ACC \leftarrow (ACC) + (RDM)$	C4, C8 C0, C5 e READ C6, C7, C9

- Cada microoperação é descrita em notação simbólica
- Esta notação é conhecida como linguagem de microprogramação

# Implementação de controle

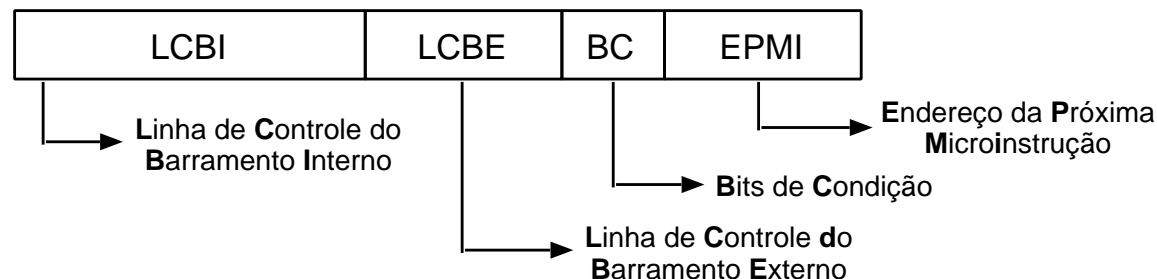
- Controle por microprogramação
  - Cada linha descreve um conjunto de microoperações que ocorrem ao mesmo tempo e é conhecida como microinstrução
  - Uma seqüência de microinstruções é um microprograma ou firmware

# Implementação de controle

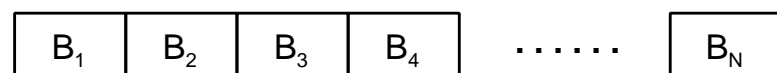
- Como o conceito de microprogramação pode ser utilizado para implementar a unidade de controle ?
  - Para cada microoperação, a unidade de controle ativa ou desativa um conjunto de sinais
  - Cada linha de controle é representada por um dígito binário
  - Uma palavra de controle é um conjunto de bits onde cada bit representa uma linha de controle
  - Cada microoperação pode ser representada por um padrão de 0s e 1s na palavra de controle

# Implementação de controle

## • Microinstruções horizontais



(a) Exemplo de um formato de microinstrução horizontal



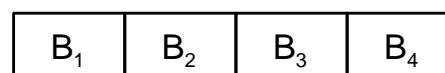
$B_1$  - porta lógica CI  $\rightarrow$  REM

$B_2$  - porta lógica ROM  $\rightarrow$  RI

$B_3$  - sinal relógio para CI

Bit  $B_1, B_2 \dots B_N$   $\left\{ \begin{array}{l} 1 - \text{ativa a linha} \\ 0 - \text{desativa a linha} \end{array} \right.$

(b) Exemplo de campo LCBI de uma microinstrução



$B_1$  - não desvia

$B_2$  - desvio incondicional

$B_3$  - desviar se reg. = 0

$B_4$  - desviar se OUL

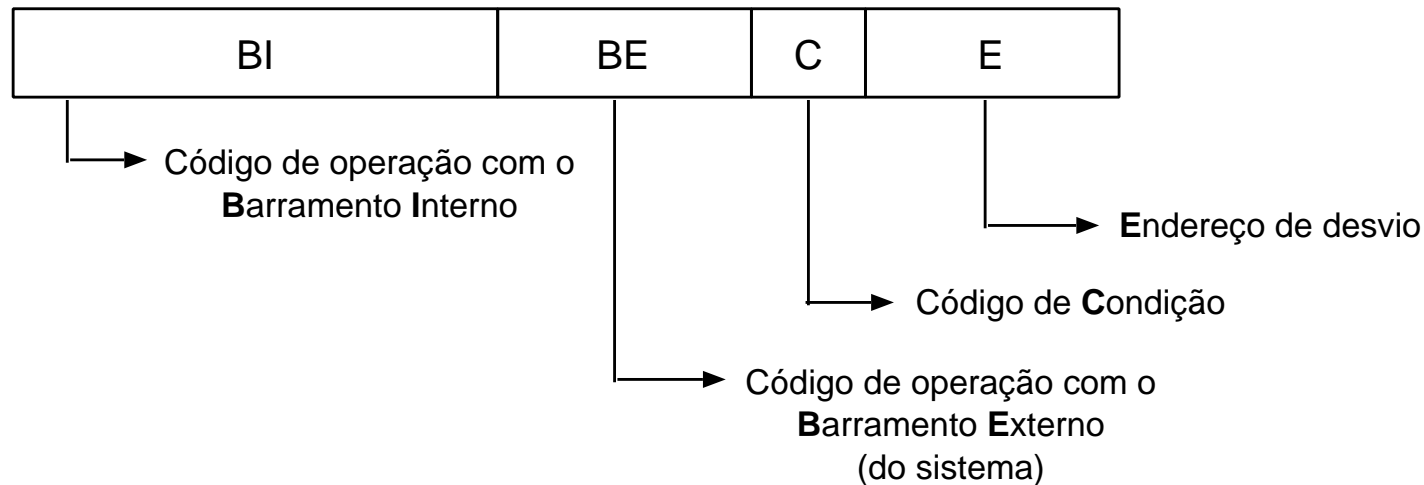
(c) Exemplo de campo BC de uma microinstrução

(Fig. 6.49 do livro texto)

- Linhas de controle ativam sinais de controle que fazem com que uma ou mais microoperações sejam executadas
- Se condição especificada no campo de condição for falsa, executa a próxima microinstrução da seqüência
- Se condição verdadeira, o endereço da próxima microinstrução será dado pelo campo Endereço da Próxima Microinstrução

# Implementação de controle

- Microinstruções verticais

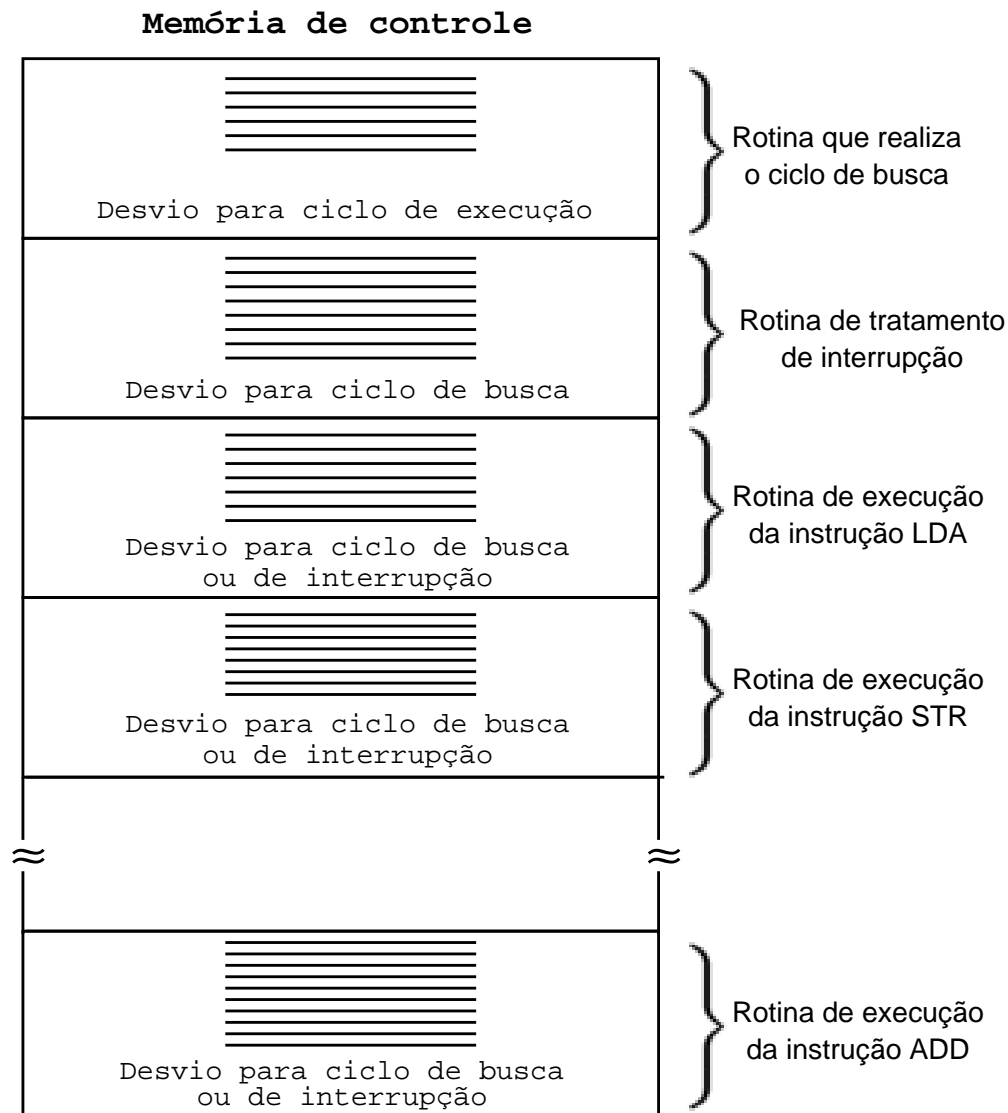


(Fig. 6.50 do livro texto)

- Os bits das linhas de controle não acessam diretamente os sinais de controle
- O campo BI contém um código de um grupo de ações que deve ser decodificado para gerar os sinais de controle de forma correta
- Utilizado para diminuir o número de bits das microinstruções

# Implementação de controle

- Organização da memória de controle

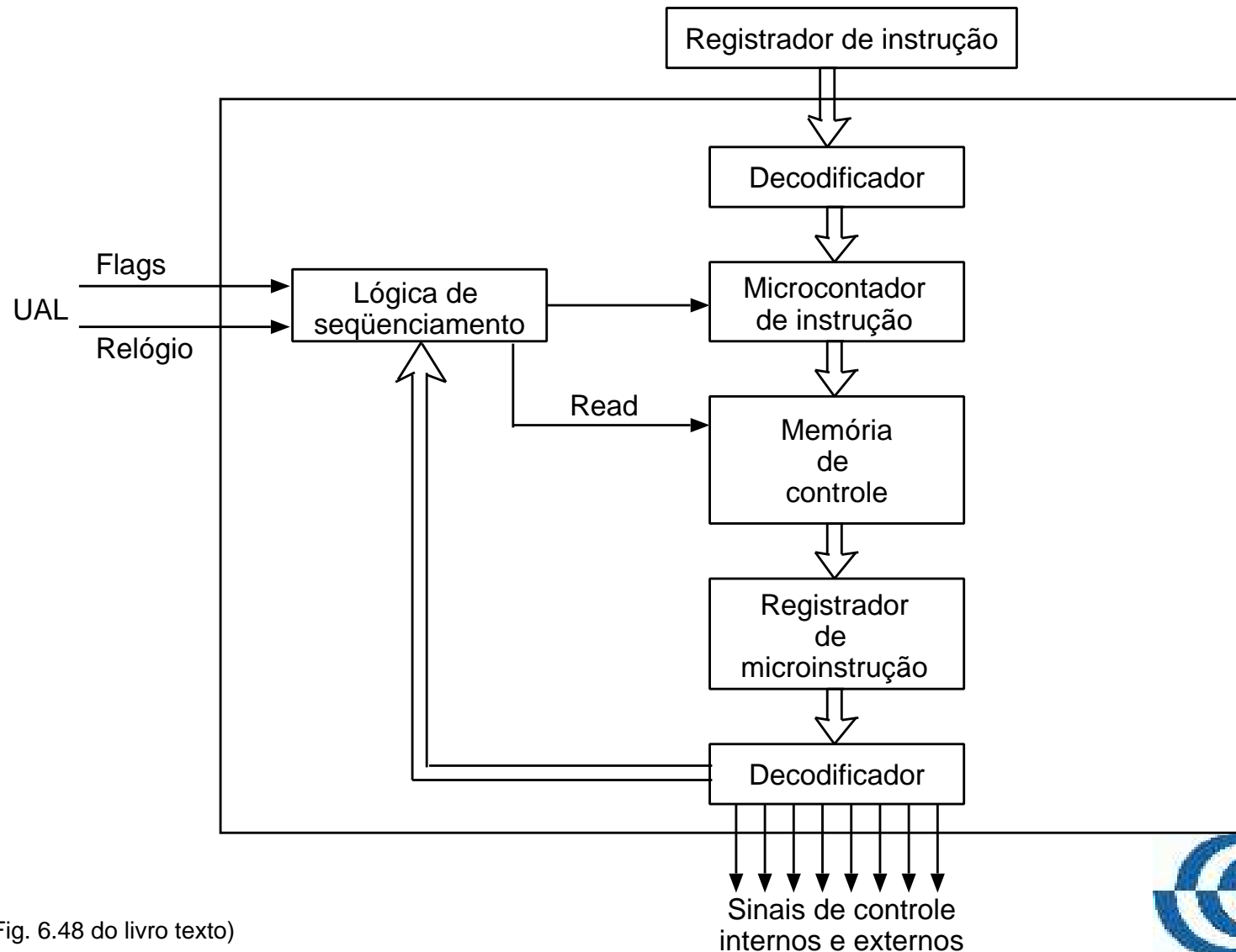


(Fig. 6.51 do livro texto)



# Implementação de controle

- Projeto de uma unidade de controle microprogramada



(Fig. 6.48 do livro texto)

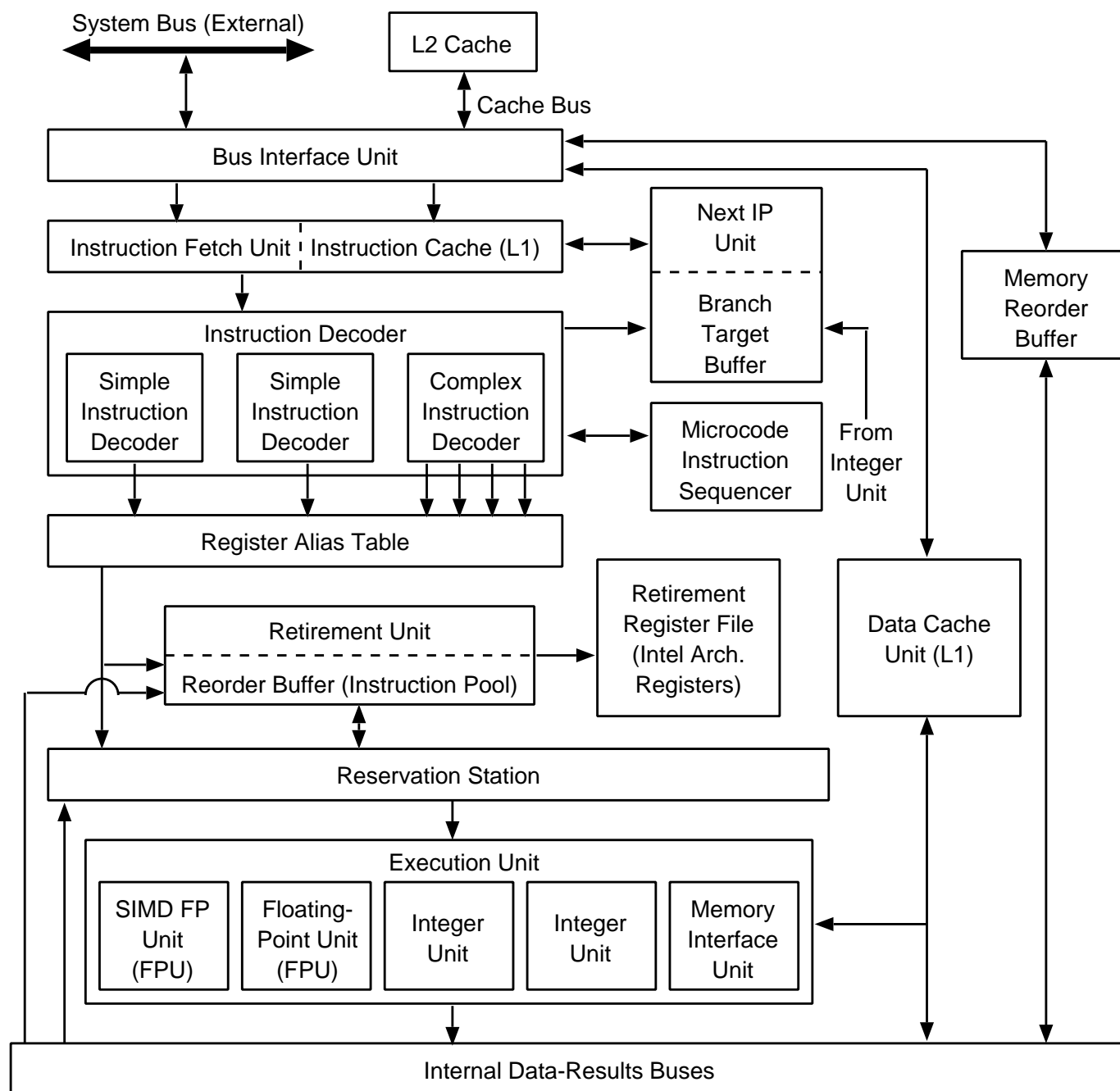
# Implementação de controle

- Vantagens de uma unidade de controle microprogramada
  - Simplifica o projeto da unidade de controle
  - Implementação mais barata e menos sujeita a erro
  - Lógica de seqüenciamento e dos decodificadores possuem uma lógica mais simples que a lógica exigida na implementação por hardware

# Implementação de controle

- Desvantagens de uma unidade de controle microprogramada
  - Mais lenta que a unidade implementada por hardware
  - Unidades de controle de máquinas CISC implementadas por microprogramação, devido à facilidade de implementação
  - Os processadores RISC possuem formato de instrução mais simples e as unidades de controle são tipicamente implementadas por hardware

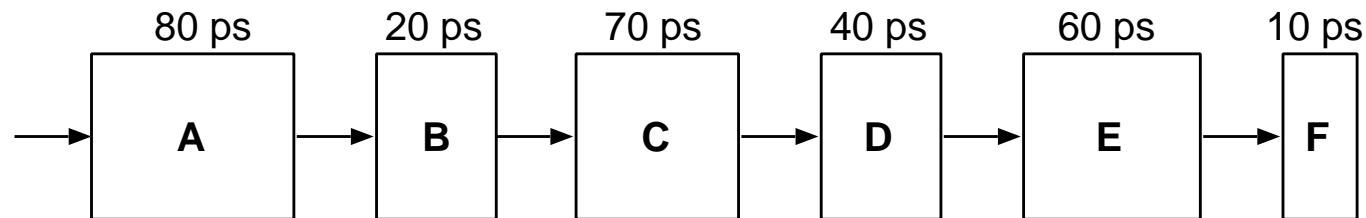
# Diagrama em blocos dos processadores da família P6



(Fig. 6.61 do livro texto)

# Exercício

- Analisando-se uma unidade central de processamento, verificou-se que suas instruções podem ser separadas em uma seqüência de 6 blocos (A-F), que apresentam os seguintes tempos de execução: 80, 20, 70, 40, 60 e 10 ps, onde  $p=10^{-12}$ , como mostrado na figura abaixo:

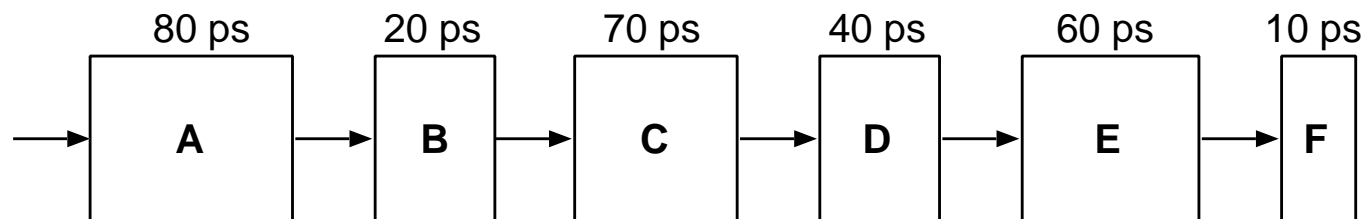


# Exercício

- Existem várias maneiras de se criar uma arquitetura utilizando-se pipeline para esta máquina. Indique como você implementaria esta arquitetura para cada um dos casos abaixo, indicando como as operações seriam divididas em estágios de pipeline. Para cada caso indique também o tempo de execução de uma seqüência de 10 instruções.
  - a) Pipeline em 2 estágios
  - b) Pipeline em 3 estágios
  - c) Pipeline em 4 estágios
  - d) Pipeline em 5 estágios

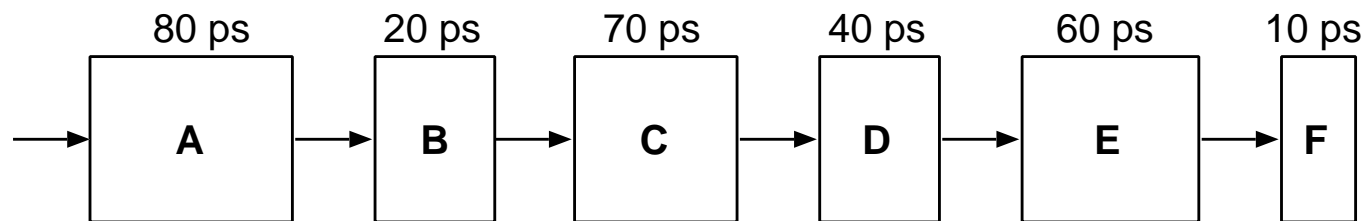
# Exercício

- Para cada caso, deve-se agrupar os blocos em cada estágio, de modo que os estágios possuam tempos de execução parecidos.
  - Pipeline em dois estágios
  - Os blocos A, B e C devem ficar no primeiro estágio com um tempo de execução de 170 ps e os blocos D, E e F no segundo estágio com tempo de execução de 110 ps. Logo a máquina deverá ser projetada de modo que cada estágio possua o maior dos tempos de execução dentre os estágios, que no caso é 170 ps. O tempo de execução de 10 instruções será :  $340 + 9 \times 170 = 1870$  ps



# Exercício

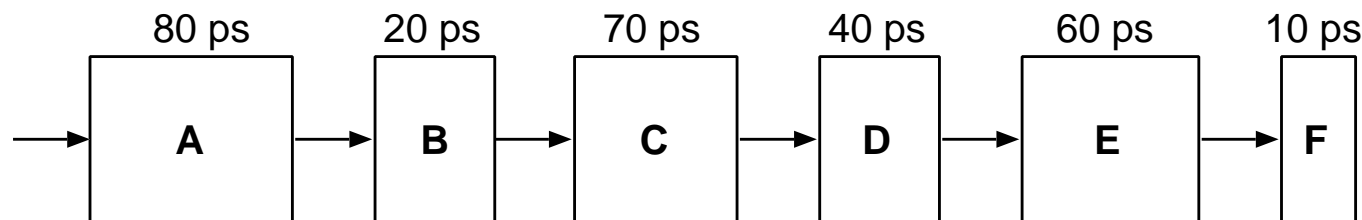
- Pipeline em três estágios
- Os blocos A e B devem ficar no primeiro estágio com um tempo de execução de 100 ps, os blocos C e D no segundo com tempo de execução de 110 ps e os blocos E e F no terceiro com tempo de execução de 70 ps. Logo a máquina deverá ser projetada de modo que cada estágio possua o maior dos tempos de execução dentre os estágios, que no caso é 110 ps. O tempo de execução de 10 instruções será :  $330 + 9 \times 110 = 1320$  ps





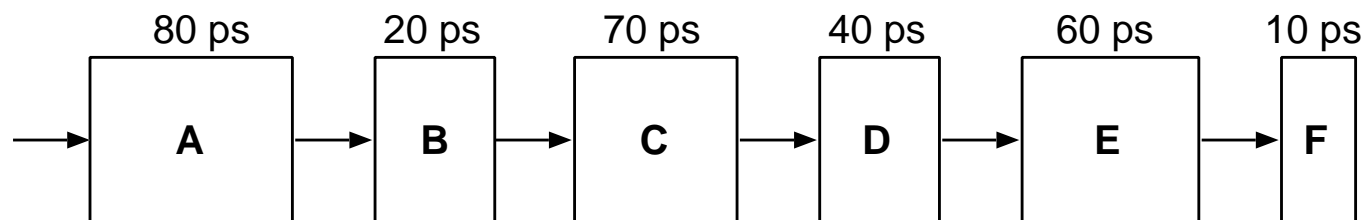
# Exercício

- Pipeline em quatro estágios
- O bloco A deve ficar no primeiro estágio com um tempo de execução de 80 ps, os blocos B e C no segundo com tempo de execução de 90 ps, o bloco D no terceiro estágio com duração de 40 ps e os blocos E e F no quarto com tempo de execução de 70 ps. Logo a máquina deverá ser projetada de modo que cada estágio possua o maior dos tempos de execução dentre os estágios, que no caso é 90 ps. O tempo de execução de 10 instruções será :  $360 + 9 \times 90 = 1170$  ps



# Exercício

- Pipeline em cinco estágios
- O bloco A deve ficar no primeiro estágio com um tempo de execução de 80 ps, o bloco B no segundo com duração igual a 20 ps, C no terceiro com tempo de execução de 70 ps, o bloco D no quarto estágio com duração de 40 ps e os blocos E e F no quinto com tempo de execução de 70 ps. Logo a máquina deverá ser projetada de modo que cada estágio possua o maior dos tempos de execução dentre os estágios, que no caso é 80 ps. O tempo de execução de 10 instruções será :  $400 + 9 \times 80 = 1120$  ps



# Exercícios

- Capítulo 6 do livro texto
  - 23, 24 e 25

## Aula 6

### Professores:

Lúcia M. A. Drummond  
Simone de Lima Martins

### Conteúdo:

#### Representação de Dados

- Tipos de dados
- Tipo caractere
- Tipo numérico

# Representação de dados

- Instruções de máquina realizam operações sobre dados
- Dados e instruções são armazenados internamente como uma seqüência de bits
- As classes de dados abordadas nesta aula:
  - Caracteres
  - Números

# Representação de dados

- Existem diversas formas de representação de dados que são utilizadas e compreendidas pelo hardware dos sistemas
- Cada dado é representado internamente por um número de bits
- A quantidade de bits utilizada afeta o tamanho e capacidade de inúmeros componentes do sistema
  - Tamanho do barramento de dados
  - Capacidade dos registradores
  - Capacidade da UAL

# Tipos de dados

- O programador deve definir para o sistema como cada dado será manipulado
- Cada dado declarado no programa deve possuir um tipo associado
- Exemplo de declaração de tipo em Pascal:
  - VAR QUANT: INTEGER;
  - VAR QUANT: REAL;
- O tipo associado indica como os bits devem ser organizados para representar o dado e como devem ser realizadas as operações sobre o dado

# Tipos de dados

## Exemplo

- Somar 103 e 258 representados como valores inteiros

$$\begin{array}{r} 01 \\ 103 \\ + 258 \\ \hline 361 \end{array}$$

Somar

Voltar

- Somar 103 e 258 representados em notação científica

$$103 = 0,103 \times 10^{+3}$$

$$258 = 0,258 \times 10^{+3}$$

$$\begin{array}{r} 001 \\ 0,103 \\ + 0,258 \\ \hline 0,361 \end{array}$$

Resultado:  $0,361 \times 10^{+3}$

Somar

Voltar



# Tipos de dados

- Tipos primitivos mais utilizados
  - Tipo caractere
  - Tipo numérico
- PASCAL
  - CHAR: caractere
  - INTEGER e REAL: numérico

# Tipo caractere

- Como representar todos os caracteres alfabéticos (maiúsculos e minúsculos), algarismos decimais, sinais de pontuação utilizando somente dois símbolos (0 e 1) ?
- Cada caractere é representado por uma seqüência distinta de bits
- Caracteres são codificados para o correspondente grupo de bits de acordo com o código utilizado

# Tipo caractere

- Códigos
  - BCD (Binary Coded Decimal)
    - Grupo de 6 bits/caractere
    - Permite a codificação de até 64 caracteres
  - EBCDIC (Extended Binary Coded Decimal Interchange Code)
    - Grupo de 8 bits/caractere
    - Permite a codificação de até 256 caracteres
  - ASCII (American Standard Code for Information Exchange)
    - Grupo de 7 bits/caractere mais um bit de paridade
    - Permite a codificação de até 128 caracteres
    - Versões estendidas utilizam 8 bits
  - UNICODE
    - Grupo de 16 bits/caractere
    - Permite a codificação de até 65.536 caracteres

# Utilizando a tabela ASCII

Decimal →	32	48	64	80	96	112
Hex →	20 (space)	30 0	40 @	50 P	60 `	70 p
	33	49	65	81	97	113
	21 !	31 1	41 A	51 Q	61 a	71 q
	34	50	66	82	98	114
	22 "	32 2	42 B	52 R	62 b	72 r
	35	51	67	83	99	115
	23 #	33 3	43 C	53 S	63 c	73 s
	36	52	68	84	100	116
	24 \$	34 4	44 D	54 T	64 d	74 t
	37	53	69	85	101	117
	25 %	35 5	45 E	55 U	65 e	75 u
	38	54	70	86	102	118
	26 &	36 6	46 F	56 V	66 f	76 v
	39	55	71	87	103	119
	27 '	37 7	47 G	57 W	67 g	77 w
	40	56	72	88	104	120
	28 (	38 8	48 H	58 X	68 h	78 x
	41	57	73	89	105	121
	29 )	39 9	49 I	59 Y	69 i	79 y
	42	58	74	90	106	122
	2A *	3A :	4A J	5A Z	6A j	7A z
	43	59	75	91	107	123
	2B +	3B ;	4B K	5B [	6B k	7B {
	44	60	76	92	108	124
	2C ,	3C <	4C L	5C \	6C l	7C
	45	61	77	93	109	125
	2D —	3D =	4D M	5D ]	6D m	7D }
	46	62	78	94	110	126
	2E .	3E >	4E N	5E ^	6E n	7E ~
	47	63	79	95	111	127
	2F /	3F ?	4F O	5F _	6F o	7F DEL

**VAR N: INTEGER;**

**56 41 52 20 4E 3A 49 4E 54 45 47 45 52 3B**

**N:= 5;**

**4E 3A 3D 35 3B**

# Tipo numérico

- Forma mais eficiente de representar números é utilizar representação binária
- Deve-se levar em consideração:
  - A representação do sinal de um número
  - A representação da vírgula (ou ponto) que separa a parte inteira da parte fracionária de um número não-inteiro
  - A quantidade limite de algarismos possível de ser processada pela UAL de um processador

# Representação em ponto fixo

- Consiste na determinação de uma posição fixa para a vírgula (ou ponto)
- Todos os dados representados em ponto fixo possuem a mesma quantidade de algarismos inteiros e fracionários
  - 1101,101    1110,001    0011,110
- As posições mais adotadas para a vírgula são:
  - Na extremidade esquerda do número (número é totalmente fracionário)
  - Na extremidade direita do número (número é inteiro)

# Representação em ponto fixo

- Na maioria dos processadores atuais, esta representação é utilizada para representar números inteiros
- Como representar números inteiros sem sinal ?
  - Converte para a base 2
  - Considere que podemos utilizar 5 bits para representar inteiros sem sinal
  - A representação do número  $(24)_{10}$  será 11000

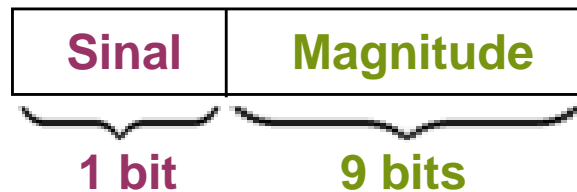
# Representação em ponto fixo

- Como representar números inteiros com sinal ?
  - Sinal e magnitude
  - Complemento a 2



# Representação em ponto fixo

- Sinal e magnitude
  - A representação em sinal magnitude de um número com n bits é obtida utilizando-se o bit mais à esquerda para indicar o sinal e os n-1 bits para indicar a magnitude do número
- Bit de sinal 0 indica número positivo e 1, negativo



0000010101  
+ 21

1000010101  
- 21

# Representação em ponto fixo

- O maior número que pode ser representado em sinal e magnitude utilizando-se  $n$  bits

$$\underbrace{0}_{1} \underbrace{1111 \dots 1111}_{n-1 \text{ bits}} \quad +(2^{n-1} - 1)$$

- O menor número que pode ser representado em sinal e magnitude utilizando-se  $n$  bits

$$\underbrace{1}_{1} \underbrace{1111 \dots 1111}_{n-1 \text{ bits}} \quad -(2^{n-1} - 1)$$

- Faixa de valores  $-(2^{n-1} - 1)$  a  $+(2^{n-1} - 1)$

# Representação em ponto fixo

- Características da representação sinal e magnitude
  - Duas representações para o número 0
    - +0 (00000...000) e -0 (10000...000)
    - Mais complicado de testar se um valor é igual a 0
  - A mesma quantidade de números positivos e negativos é representada

# Representação em ponto fixo

- Aritmética em sinal e magnitude
  - Algoritmo de soma:
    - Verificam-se os sinais dos números e efetua-se uma comparação entre eles
    - Se ambos possuem o mesmo sinal, somam-se as magnitudes e o sinal do resultado é o mesmo das parcelas
    - Se os números possuem sinais diferentes:
      - identifica-se a maior das magnitudes e registra-se o seu sinal
      - subtrai-se a magnitude menor da maior
      - o sinal do resultado é igual ao sinal de maior magnitude

# Representação em ponto fixo

- Soma utilizando-se representação sinal e magnitude e 6 bits

$$\begin{array}{r}
 \phantom{+} 10 \\
 + \phantom{+} 15 \\
 \hline
 +25
 \end{array}$$

$$\begin{array}{r}
 110 \\
 001010 \\
 001111 \\
 \hline
 01001
 \end{array}$$

Somar

Voltar

$$\begin{array}{r}
 +15 \\
 + -4 \\
 \hline
 +11
 \end{array}$$

$$\begin{array}{r}
 001111 \\
 100100 \\
 \hline
 001011
 \end{array}$$

Somar

Voltar

# Representação em ponto fixo

- Aritmética em sinal e magnitude
  - Algoritmo de subtração:
    - Troca-se o sinal do subtraendo
    - Executa algoritmo de soma
    - **Exemplo:**  $-18 - (+12) = -18 + (-12)$

		0 0 0 0
- 1 8	1 1 0 0 1 0	
+ - 1 2	1 0 1 1 0 0	
<hr/>	<hr/>	
- 3 0	1 1 1 1 1 0	

Somar

Voltar

# Representação em ponto fixo

- Complemento a 2
  - Números positivos são representados como na representação em sinal e magnitude
  - Números negativos
    - Invertem-se os bits da representação positiva
    - Soma-se 1
  - Números em complemento a 2 com 8 bits

+3 = 00000011

+2 = 00000010

+1 = 00000001

0 = 00000000

-1 = inv(00000001) + 1 = 11111110 + 1 = 11111111

-2 = 11111110

-3 = 11111101

# Representação em ponto fixo

- Inteiro com sinal codificado por um vetor com  $w$  bits  $[x_{w-1}, x_{w-2}, \dots, x_0]$  utilizando representação complemento a 2
  - $N = -x_{w-1} 2^{w-1} + \sum_{i=0}^{w-2} x_i \times 2^i$
  - $0101 = -0 \times 2^3 + 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 = +5$
  - $1101 = -1 \times 2^3 + 1 \times 2^0 + 0 \times 2^1 + 1 \times 2^2 = -3$
- Bit mais significativo é o bit de sinal: 1 para negativos, 0 para positivos



# Representação em ponto fixo

- Intervalo de valores a ser representado em complemento a 2:

- Menor número

$$\underbrace{1}_{1} \underbrace{0000 \dots 0000}_{n-1 \text{ bits}} - (2^{n-1})$$

- Maior número

$$\underbrace{0}_{1} \underbrace{1111 \dots 1111}_{n-1 \text{ bits}} + (2^{n-1} - 1)$$

- Faixa de valores:  $(2^{n-1})$  a  $+(2^{n-1} - 1)$

# Representação em ponto fixo

- Vantagens da utilização de representação complemento a 2:
  - uma única representação para o zero
  - necessita apenas um circuito somador para operações de soma e subtração
  - operações simples para encontrar a representação de números negativos (inverte e soma 1)
- Assimetria na quantidade de números representados
  - $-2^{n-1}$  a  $+(2^{n-1}-1)$

# Representação em ponto fixo

- Extensão de sinal em complemento a 2
  - Dado um inteiro  $x$  representado com sinal utilizando-se  $w$  bits, deseja-se representá-lo com  $k+w$  bits
  - Faça  $k$  cópias do bit de sinal
  - $X' = \underbrace{x_{w-1}, \dots, x_{w-1}}_{K \text{ cópias de BMS}}, x_{w-1}, x_{w-2}, \dots, x_0$
  - $0011 \rightarrow 00000011$
  - $1100 \rightarrow 11111100$

# Representação em ponto fixo

- Adição em complemento a 2
  - Soma em binário normal
  - Monitora o bit de sinal para saber se houve estouro (overflow)
- Subtração em complemento a 2
  - Obtém a representação em complemento a 2 do subtraendo e soma ao minuendo
    - $a - b = a + (-b)$
  - Somente necessita de circuitos de soma e de inversão
  - Monitora o bit de sinal para saber se houve estouro (overflow)

# Representação em ponto fixo

- Adição em complemento a 2

- +11 + (+20)

	0 0 0 0 0
+1 1	0 0 1 0 1 1
+ +2 0	0 1 0 1 0 0
<hr/>	<hr/>
+31	0 1 1 1 1 1

Somar

Voltar

- 25 + (-12)

	0 0 1 0 0
-2 5	1 0 0 1 1 1
+ -1 2	1 1 0 1 0 0
<hr/>	<hr/>
-37	0 1 1 0 1 1

Somar

Voltar

# Representação em ponto fixo

- Subtração em complemento a 2
  - $+11 - (+21) = +11 + (-21)$

				0 1 0 1 1	
+ 1 1	0 0 1 0 1 1			0 0 1 0 1 1	
+ - 2 1	0 1 0 1 0 1	1 0 1 0 1 0 + 1	=	1 0 1 0 1 1	
<hr/>				<hr/>	
- 1 0				1 1 0 1 1 0	

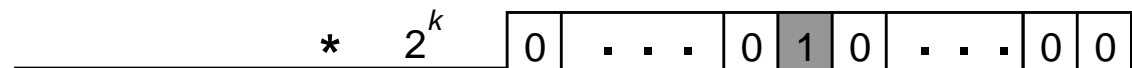
Somar

Voltar

# Representação em ponto fixo

- Operação de multiplicação por potência de 2
  - $u \ll k$  fornece  $u * 2^k$
  - Para números com e sem sinal

Operandos:  $w$  bits



Produto verdadeiro:  $w+k$  bits.



Descarta  **$k$**  bits:  **$w$**  bits

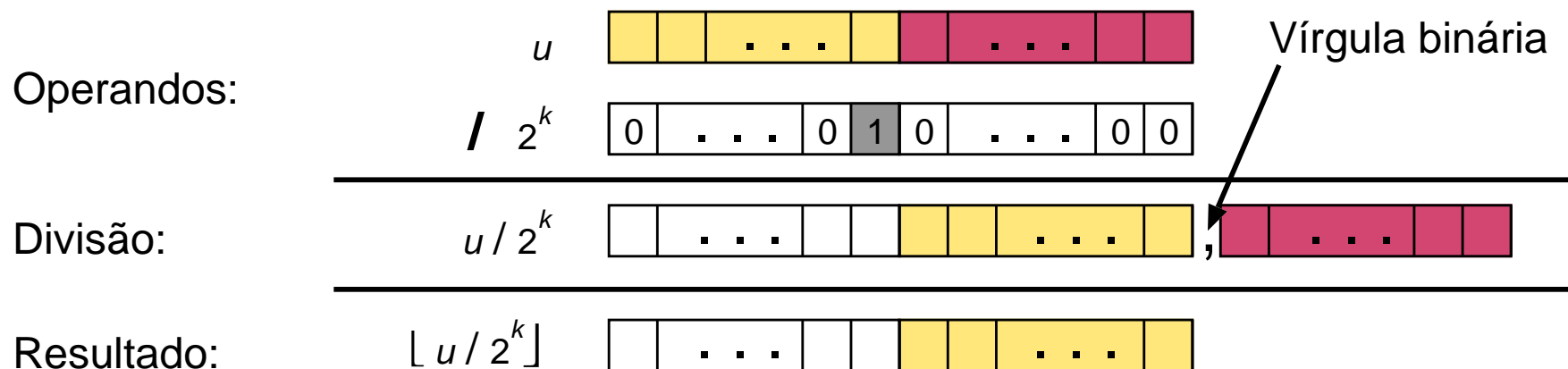


- Exemples

- $u \ll 3 == u * 8$
- $u \ll 5 - u \ll 3 == u * 24$
- Maioria das máquinas desloca e soma mais rápido que multiplica
  - Compilador gera o código automaticamente

# Representação em ponto fixo

- Quociente da divisão de número sem sinal por potência de 2
  - $u \gg k$  fornece  $\lfloor u / 2^k \rfloor$
  - Utiliza deslocamento lógico

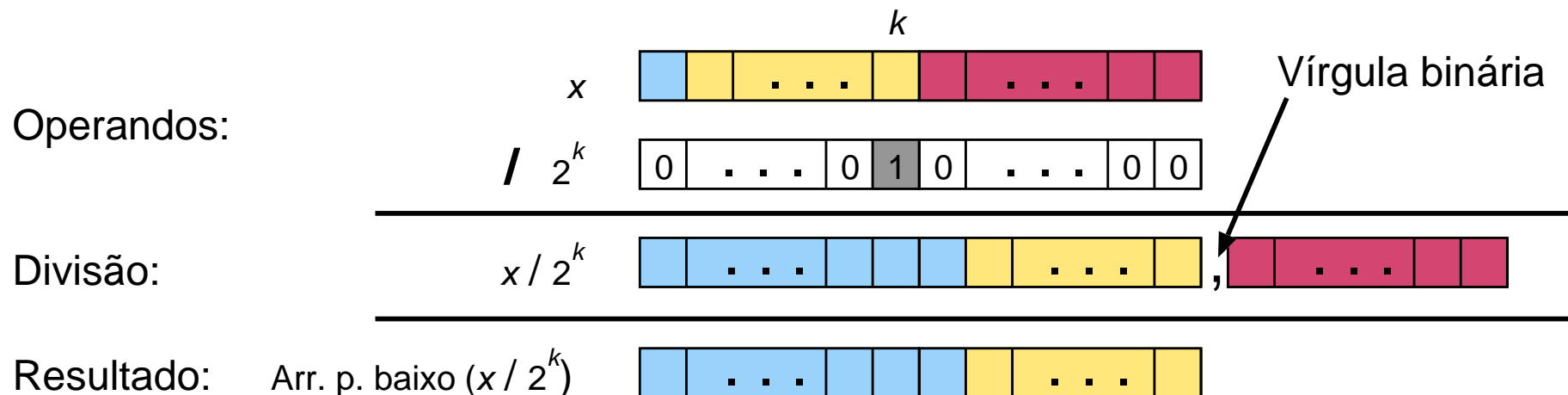


	Divisão	Calculada	Hexa	Binário
x	15213	15213	3B 6D	00111011 01101101
$x \gg 1$	7606,5	7606	1D B6	<b>000</b> 11101 10110110
$x \gg 4$	950,8125	950	03 B6	<b>0000</b> 0011 10110110
$x \gg 8$	59,4257813	59	00 3B	<b>00000000</b> 00111011



# Representação em ponto fixo

- Quociente da divisão de número com sinal por potência de 2
  - $x \gg k$  fornece  $\lfloor x / 2^k \rfloor$
  - Utiliza deslocamento aritmético
  - Arredonda para direção errada quando  $u < 0$



# Representação em ponto fixo

- Quociente da divisão de número com sinal por potência de 2

	Divisão	Calculado	Hexa	Binário
y	-15213	-15213	C4 93	11000100 10010011
y >> 1	-7606,5	-7607	E2 49	11100010 01001001
y >> 4	-950,8125	-951	FC 49	11111100 01001001
y >> 8	-59,4257813	-60	FF C4	11111111 11000100

# Representação em ponto flutuante

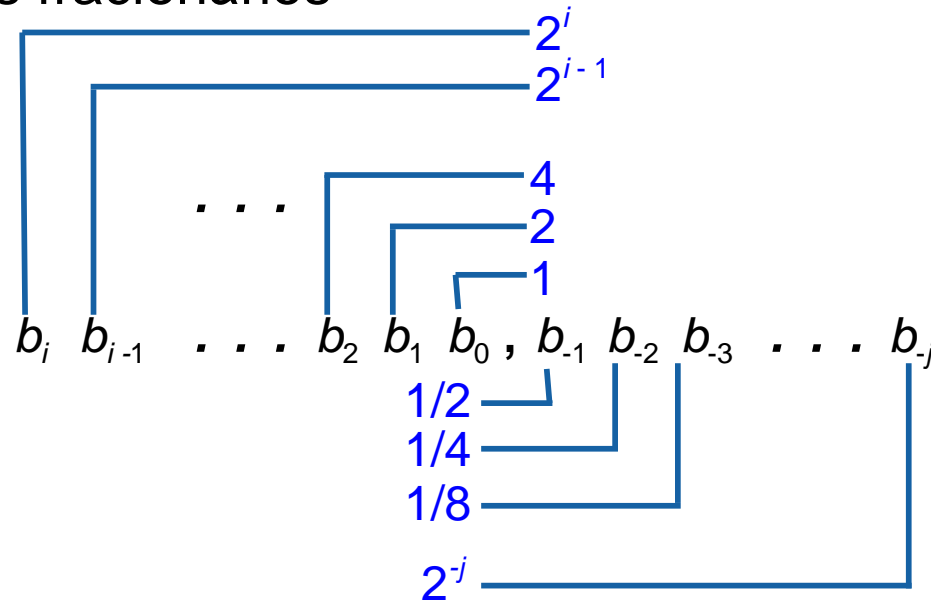
- Como representar números fracionários ?
  - Vírgula poderia ficar fixa em algum lugar
  - Limite no número de bits para parte inteira e fracionária
- Notação científica normalizada na base 10
  - $1.000.000.000 = 1,0 \times 10^9$
  - $0,00000000000017 = 1,7 \times 10^{-11}$

# Representação em ponto flutuante

- Representação IEEE Standard 754
  - Estabelecido em 1985 como padrão uniforme para aritmética em ponto flutuante
  - A maioria das CPUs suporta este padrão
- Foi projetado para criar um padrão com facilidades para operações numéricas

# Representação em ponto flutuante

- Números binários fracionários



- Representação

- Bits à direita da "vírgula binária" representam potências fracionárias de 2

- Representa o número: 
$$\sum_{k=-j}^i b_k \cdot 2^k$$

# Representação em ponto flutuante

- Representação IEEE 754
  - Forma numérica
    - $-1^s M 2^E$ 
      - Bit de sinal  $s$  determina se o número é negativo ou positivo
      - O significando  $M$  é um valor fracionário na faixa  $[1.0, 2.0)$ .
      - O expoente  $E$  é um número inteiro com sinal

- Codificação



- MSB é o bit de sinal
- O campo exp codifica  $E$
- O campo frac codifica  $M$

# Representação em ponto flutuante

- Representação IEEE 754
  - Tamanhos
    - **Precisão simples: 8 bits para exp, 23 bits para frac**
      - 32 bits no total
    - **Precisão dupla: 11 bits para exp, 52 bits para frac**
      - 64 bits no total
  - Precisão estendida: 15 bits para exp, 63 bits para frac
    - Somente em máquinas compatíveis com Intel
    - 80 bits no total
      - » 1 bit não utilizado

# Representação em ponto flutuante

- Valores numéricos normalizados
  - $\text{exp} \neq 000\dots 0$  e  $\text{exp} \neq 111\dots 1$
  - Representação em excesso de  $n$  para expoente
 
$$E = \text{Exp} - \text{Bias}$$
    - $\text{Exp}$  : inteiro sem sinal representado por  $\text{exp}$
    - $\text{Bias}$  : Valor de  $n$ 
      - » Precisão simples: 127 ( $\text{Exp}$ : 1...254,  $E$ : -126...127)
      - » Precisão dupla: 1023 ( $\text{Exp}$ : 1...2046,  $E$ : -1022...1023)
      - » Em geral:  $\text{Bias} = 2^{e-1} - 1$ , onde  $e$  é o número de bits do expoente
  - Significando codificado com 1 implícito antes da vírgula
 
$$M = 1,xxx\dots x_2$$
    - $xxx\dots x$ : bits do campo  $\text{frac}$
    - Mínimo quando 000...0 ( $M = 1,0$ )
    - Máximo quando 111...1 ( $M = 2,0 - \epsilon$ )



# Representação em ponto flutuante

- Valor

Float  $F = 15213.0$ ;

$$15213_{10} = 11101101101101_2 = 1,1101101101101_2 \times 2^{13}$$

- Significando

$$M = 1,\underline{1101101101101}_2$$

$$\text{frac} = \underline{1101101101101}0000000000_2$$

- Expoente

$$E = 13$$

$$\text{Bias} = 127$$

$$\text{Exp} = 140 = 10001100_2$$

## Representação em ponto flutuante

Hexa: 4 6 6 D B 4 0 0

Binário: 0100 0110 0110 1101 1011 0100 0000 0000

# Representação em ponto flutuante

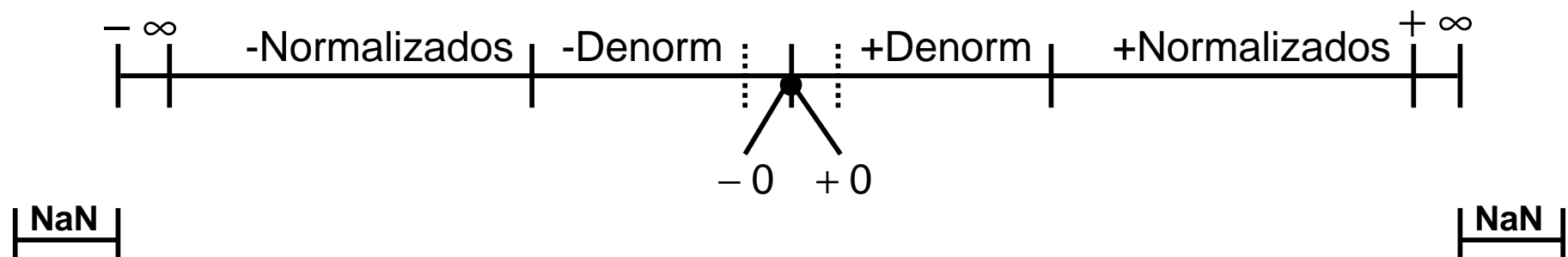
- Valores denormalizados
  - $\text{exp} = 000\dots 0$
  - Valor
    - Valor do expoente  $E = -\text{Bias} + 1$
    - Valor do significando  $M = 0, \text{xxx}\dots\text{x}_2$ 
      - $\text{xxx}\dots\text{x}$ : bits do campo frac
  - Casos
    - $\text{exp} = 000\dots 0, \text{frac} = 000\dots 0$ 
      - Representa o valor 0
      - Existem duas representações: +0 and -0
    - $\text{exp} = 000\dots 0, \text{frac} \neq 000\dots 0$ 
      - Números muito perto de 0,0

# Representação em ponto flutuante

- Valores especiais
  - $\text{exp} = 111\dots 1$
  - Casos
    - $\text{exp} = 111\dots 1, \text{frac} = 000\dots 0$ 
      - Representa o valor  $\infty$  (infinito)
      - Operações em que ocorrem overflows
      - Positivo e negativo
      - Ex.,  $1,0/0,0 = -1,0/-0,0 = +\infty$ ,  $1,0/-0,0 = -\infty$
    - $\text{exp} = 111\dots 1, \text{frac} \neq 000\dots 0$ 
      - Not-a-Number (NaN)
      - Representa o caso quando não se pode determinar um valor numérico
      - Ex.,  $\text{sqrt}(-1)$ ,  $\infty - \infty$

# Representação em ponto flutuante

- Visualização da codificação de números reais em ponto flutuante



# Representação em ponto flutuante

- Exemplos de valores

Descrição	exp	frac	Valor numérico
Zero	00...00	00...00	0,0
Menor Pos. Denorm.	00...00	00...01	$2^{-\{23,52\}} \times 2^{-\{126,1022\}}$
Simples $\approx 1,4 \times 10^{-45}$			
Dupla $\approx 4,9 \times 10^{-324}$			
Maior Pos. Denorm.	00...00	11...11	$(1,0 - \epsilon) \times 2^{-\{126,1022\}}$
Simples $\approx 1,18 \times 10^{-38}$			
Dupla $\approx 2,2 \times 10^{-308}$			
Menor Pos. Norm.	00...01	00...00	$1,0 \times 2^{-\{126,1022\}}$
Maior que o maior denormalizado			
Um	01...11	00...00	1,0
Maior Pos. Norm.	11...10	11...11	$(2,0 - \epsilon) \times 2^{\{127,1023\}}$
Simples $\approx 3,4 \times 10^{38}$			
Dupla $\approx 1,8 \times 10^{308}$			

# Representação em ponto flutuante

- Soma em ponto flutuante

$$(-1)^{s1} M1 2^{E1}$$

$$(-1)^{s2} M2 2^{E2}$$

- Assuma  $E1 > E2$

- Resultado exato

$$(-1)^s M 2^E$$

- Sinal  $s$ , significando  $M$ :
  - Resultado de alinhamento com sinal & soma
- Expoente  $E$ :  $E1$

- Ajuste

- Se  $M \geq 2$ , desloca  $M$  para a direita, incrementa  $E$
- Se  $M < 1$ , desloca  $M$  para a esquerda  $k$  posições, decrementa  $E$  de  $k$
- Overflow se  $E$  fora da faixa
- Arredonda  $M$  para ajuste ao campo frac

$$|\leftarrow E1 - E2 \rightarrow|$$

$$(-1)^{s1} M1$$

$$(-1)^{s2} M2$$

+

$$(-1)^s M$$

# Exercícios

- Capítulo 7 do livro texto
  - 4, 5, 9, 12, 13, 15 e 27

## Aula 7

### Professores:

Lúcia M. A. Drummond  
Simone de Lima Martins

### Conteúdo:

#### Representação de instruções

- Introdução
- Quantidade de operandos
- Modos de endereçamento



# Introdução

- Conjunto de instruções: instruções que o processador pode realizar diretamente
- Exemplo de instrução de máquina:

Instrução de máquina de um operando

Cód. de Operação	Operando
------------------	----------

Cód. de Operação - indica o tipo da operação a ser realizada  
Operando - endereço do dado

(Fig. 8.1 do livro texto)

# Introdução

- Processadores projetados para realizar instruções muito simples
- Código de operação:
  - código que após ser decodificado durante o ciclo de execução permitirá que a UC emita os sinais necessários para se efetivar a seqüência de passos de realização da operação indicada
  - Quantidade de bits neste campo define o limite máximo de instruções que o processador pode executar
  - **Exemplo:** código de 6 bits, processador pode executar 64 instruções

# Introdução

- Operando: contém informação sobre o dado - tipo da informação sobre o dado - seu valor ou o endereço de memória onde localizá-lo. Podem existir um ou mais campos.
- Exemplo:

C. Op.	Op. 1	Op. 2	Op. 3
--------	-------	-------	-------

ADD Op. 1, Op. 2, Op. 3

(a) Instrução de 3 operandos (com memória)

C. Op.	Op. 1	Op. 2
--------	-------	-------

MOVE Op. 1, Op. 2

(b) Instrução de 2 operandos (com memória)

C. Op.	R	Op.
--------	---	-----

ADC R, Op.

(c) Instrução de 2 operandos, sendo um registrador

C. Op.	Op.
--------	-----

JCXZ Op.

(d) Instrução de 1 operando (desvio para memória)

(Fig. 8.2 do livro texto)

# Introdução

- Representação de Instruções:
  - Quantidade de operandos
  - Modo de interpretação do valor armazenado no campo operando (modo de endereçamento do dado)

# Quantidade de Operandos

- Computadores com conjunto de instruções dos mais variados tipos e formatos
- Exemplo do formato de instrução do sistema SEAC de 1949.

C. Op.	Operando 1	Operando 2	Operando 3	End. próx. instrução
--------	------------	------------	------------	----------------------

Operando n - endereço do dado ou o próprio valor do dado

(Fig. 8.3 do livro texto)

# Quantidade de Operandos

- Computador com memória de 2K células e código de operação de 6 bits
- Endereço de 11 bits
- $4(\text{operandos}) \times 11(\text{bits de endereço}) + 6(\text{C. Op}) = 50 \text{ bits}$
- Exemplo: ADD X,Y,Z,P
- $(Z) \leftarrow (X) + (Y)$ , onde P é o endereço da próxima instrução
- Essa instrução permitiria a execução de  $C = A + B$
- Em linguagem Assembly: ADD A,B,C,P

# Quantidade de Operandos

## **Vantagens de muitos operandos:**

- Completeza: instrução possui todos os operandos, dispensando até instruções de desvio incondicional
- Menor quantidade de instruções em um programa

## **Desvantagens:**

- Grande ocupação de espaço de memória
- Nem toda instrução precisa de 3 operandos (Ex: Load, desvio)

# Quantidade de Operandos

- Tamanho de instruções:
  - Tamanho da memória
  - Tamanho e organização das células da MP
  - Velocidade de acesso
  - Organização do barramento de dados
- Economia de espaço x conjunto completo de instruções
- RISC
- Introdução do registrador CI (PC - program counter)



# Quantidade de Operandos

## Instruções com 3 operandos

Formato Básico:

C. Op.	Operando 1	Operando 2	Operando 3
--------	------------	------------	------------

(Fig. 8.4 do livro texto)

As **instruções aritméticas** podem ser do tipo:

<b>ADD</b> A,B,X	$(X) \leftarrow (A) + (B)$
<b>SUB</b> A,B,X	$(X) \leftarrow (A) - (B)$
<b>MPY</b> A,B,X	$(X) \leftarrow (A) \times (B)$
<b>DIV</b> A,B,X	$(X) \leftarrow (A) / (B)$

# Quantidade de Operandos

## Instruções com 3 operandos

Exemplo:

$$X = A * (B + C * D - E / F)$$

### **Assembly correspondente:**

MPY C,D,T1	; multiplicação de C e D, resultado em T1
DIV E,F,T2	; divisão de E por F, resultado em T2
ADD B,T1,X	; soma de B com T1, resultado em X
SUB X,T2,X	; subtração entre X e T2, resultado em X
MPY A,X,X	; multiplicação de A por X

# Quantidade de Operandos

## Instruções com 3 operandos

### **Considerações:**

- Operandos com endereços iguais - desperdício de memória
- Número de instruções igual ao número de operações
- Raramente encontradas em conjunto de instruções dos processadores atuais
- Pelo exemplo anterior, a maioria das instruções tem 2 endereços (o outro é repetido)

# Quantidade de Operandos

## Instruções com 2 operandos

ADD Op1, Op2	$(Op1) \leftarrow (Op1) + (Op2)$
SUB Op1, Op2	$(Op1) \leftarrow (Op1) - (Op2)$
MPY Op1, Op2	$(Op1) \leftarrow (Op1) \times (Op2)$
DIV Op1, Op2	$(Op1) \leftarrow (Op1) / (Op2)$

O conteúdo de Op1 será destruído com o armazenamento do resultado da operação.

Salvamento de variável:

MOVE A,B     $(A) \leftarrow (B)$

# Quantidade de Operandos

## Instruções com 2 operandos

Exemplo:

$$X = A * (B + C * D - E / F)$$

### **Assembly correspondente:**

MPY C,D	; multiplicação de C e D, resultado em C
DIV E,F	; divisão de E por F, resultado em E
ADD B,C	; soma de B com C, resultado em B
SUB B,E	; subtração de B e E, resultado em B
MPY A,B	; multiplicação de A por B, resultado em A
MOVE X,A	; armazenamento do resultado final, A, em X

# Quantidade de Operandos

## Instruções com 1 operando

ADD Op	$ACC \leftarrow ACC + (Op)$
SUB Op	$ACC \leftarrow ACC - (Op)$
MPY Op	$ACC \leftarrow ACC * (Op)$
DIV Op	$ACC \leftarrow ACC / (Op)$

O acumulador (ACC) é empregado como operando implícito e para guardar o resultado da operação.

Duas novas instruções:

LDA Op	$ACC \leftarrow (Op)$
STA Op	$(Op) \leftarrow ACC$

# Quantidade de Operandos

## Instruções com 1 operando

Para não destruir valores de variáveis:

$$X = A*(B+C*D-E/F)$$

**Assembly correspondente:**

```
LDA C
MPY D
STA X
LDA E
DIV F
STA T1
LDA B
ADD X
SUB T1
MPY A
STA X
```

# Quantidade de Operandos

## Tamanho e Consumo de Tempo de Execução de Instruções de 3, de 2 e de 1 Operando

**Instrução de 3 operandos** - C. Op. = 8 bits + 3 operandos de 20 bits cada um = 68 bits

Ciclos de memória = 4 (um para buscar a instrução e  
3 para os operandos)

**Instrução de 2 operandos** - C. Op. = 8 bits + 2 operandos de 20 bits cada um = 48 bits

Ciclos de memória = 4 (um para buscar a instrução e  
3 para os operandos)

**Instrução de 1 operando** - C. Op. = 8 bits + 1 operando de 20 bits = 28 bits

Ciclos de memória = 2 (um para buscar a instrução e  
1 para o operando)

(Tabela 8.1 do livro texto)



# Quantidade de Operandos

**Programas Assembly para Solucionar a Equação:  $X = A * (B + C * D - E / F)$**

Com instruções de 3 operandos	Com instruções de 2 operandos (sem salvamento)	Com instruções de 2 operandos (com salvamento)	Com instruções de 1 operando
MPY      C, D, T1 DIV      E, F, T2 ADD      B, T1, X SUB      X, T2, X MPY      A, X, X	MPY      C, D DIV      E, F ADD      B, C SUB      B, E MPY      A, B MOVE     X, A	MOVE     X, C MPY      X, D MOVE     T1, E DIV      T1, F ADD      X, B SUB      X, T1 MPY      X, A	LDA      C MPY      D STA      X LDA      E DIV      F STA      T1 LDA      B ADD      X SUB      T1 MPY      A STA      X
Espaço: 340 bits Tempo: 20 acessos	Espaço: 288 bits Tempo: 24 acessos	Espaço: 336 bits Tempo: 28 acessos	Espaço: 308 bits Tempo: 22 acessos

(Tabela 8.2 do livro texto)

# Quantidade de Operandos

## Considerações:

- Instruções de poucos operandos ocupam menos espaço de memória e tornam o projeto do processador mais simples
- O programa gerado com poucos operandos pode ser maior
- Instruções de 1 operando são simples e baratas. Mas o uso de 1 registrador pode reduzir a flexibilidade e velocidade de processamento
- Instruções com mais de 1 operando podem usar endereços de memória e registradores em seus formatos

# Modos de Endereçamento

## Considerações:

- O endereçamento de instrução é realizado pelo contador de instruções (CI) e o ciclo de instrução é iniciado pela transferência da instrução para o registrador de instrução (RI)
- Toda instrução consiste em uma ordem codificada para a UCP realizar uma operação sobre um dado (valor numérico, caractere alfabético ou endereço)
- Localização dos dados pode estar explicitamente indicada na própria instrução através de *CAMPO DE OPERANDO*, ou implicitamente (armazenado no ACC)

# Modos de Endereçamento

- Existem várias formas de localizar o dado: *modos de endereçamento*
  - Imediato
  - Direto
  - Indireto
  - Por registrador
  - Indexado
  - Base mais deslocamento

# Modos de Endereçamento

## Modo Imediato

- Método mais simples: indicar o próprio valor no campo operando
- O dado é transferido da memória juntamente com a instrução
- Útil para inicialização de contadores, operação com constantes

# Modos de Endereçamento

## Modo Imediato

### **Desvantagens:**

- Limitação do campo de operando reduz o valor máximo do dado a ser manipulado
- Programas repetidamente executados com valores diferentes de variáveis em cada execução, acarretariam o trabalho de alteração do campo operando a cada execução

# Modos de Endereçamento

## Modo Imediato

### Exemplo 8.1:

C. Op. / Operando  
(4 bits) / (8bits)

JMP Op.

C. Op. = 1010 = hexadecimal A

Instrução: 101000110101 ou A35 (C. Op. = A e Operando = 35)

Armazenar o valor 35 no CI

# Modos de Endereçamento

## Modo Imediato

### Exemplo 8.2:

C. Op. / R / Operando  
(4 bits) / (4 bits) / (8 bits)

MOV R, Op.

C. Op. = 0101 = hexadecimal 5

Instrução: 0101001100000111 ou 5307 (C. Op. = 5, R = 3 e Operando = 07)  
Armazenar o valor 07 no registrador 3 (R3)



# Modos de Endereçamento

## Modo Direto

Operando indica o endereço de memória onde se localiza o dado

### **Considerações:**

- Modo simples - requer apenas 1 acesso à memória
- Mais lento que o modo imediato
- Com o aumento do tamanho de memória, são necessários muitos bits para endereçamento direto

# Modos de Endereçamento

## Modo Direto

### Exemplo 8.3:

a) C. Op. / Operando  
(4 bits) / (8 bits)

LDA Op.

C. Op. = 7

Instrução: 73B (C. Op. = 7 e Operando = 3B)

Após a execução da instrução, o ACC conterá o valor 05A

	MP
	//
	//
	//
3B	05A
	//
	//
	//
5C	<del>103</del> 15D
	//
	//

# Modos de Endereçamento

## Modo Direto

### Exemplo 8.3:

**b)** C. Op. / Op.1 / Op.2  
(4 bits) / (8 bits) / (8 bits)

ADD Op.1, Op.2

C. Op. = B

Instrução: B5C3B (C. Op. = B, Op.1 = 5C e Op. 2 = 3B)

Após a execução da instrução, o endereço 5C conterà o valor 15D

	MP
	//
	//
	//
3B	05A
	//
	//
	//
5C	<del>103</del> 15D
	//
	//

# Modos de Endereçamento

## Modo Indireto

- O operando representa o endereço de uma célula, cujo conteúdo é outro endereço de célula
- Requer dois acessos à memória para se obter o dado
- Útil para acessar dados armazenados contiguamente na memória, como vetores.

# Modos de Endereçamento

## Modo Indireto

### Exemplo 8.4:

C. Op. / Operando  
(4 bits) / (8 bits)

LDA Op.

C. Op. = 4

Instrução: 474 (C. Op. = 4, Op.= 74)

Após a execução da instrução, o valor 1A4 estará armazenado no ACC

	MP
74	05D
	''
5D	1A4
	''

# Modos de Endereçamento

## Observações

- 1) Há dois métodos de indicação do modo de endereçamento das instruções:
  - O código de operação estabelece não só o tipo da instrução como também o modo de endereçamento
  - A instrução possui um campo específico para indicar o modo de endereçamento

C. Op.		Modo end.		Operando	
4		2		10	
				00 - Direto (DI) 01 - Imediato (IM) 10 - Indireto (ID) 11 - Não usado	
C. Op. = A	LDA IM	———	A13B	101000100111011	
	LDA DI	———	A43B	101010000111011	
	LDA ID	———	A83B	101100000111011	

(Fig. 8.6 do livro texto)

# Modos de Endereçamento

## Observações

Modo de endereçamento	Definição	Vantagens	Desvantagens
<b>Imediato</b>	O campo operando contém o dado.	Rapidez na execução da instrução.	Limitação do tamanho do dado. Inadequado para o uso com dados de valor variável.
<b>Direto</b>	O campo operando contém o endereço do dado.	Flexibilidade no acesso a variáveis de valor diferente em cada execução do programa.	Perda de tempo, se o dado é uma constante.
<b>Indireto</b>	O campo operando contém o endereço do endereço do dado.	Manuseio de vetores (quando o modo indexado não está disponível). Uso como "ponteiro".	Muitos acessos a MP para execução.

(Fig. 8.7 do livro texto)

# Modos de Endereçamento

## Observações

2) Em relação aos 3 modos de endereçamento temos:

- O modo imediato não requer acesso à memória principal para buscar o dado, o modo direto requer um acesso, o modo indireto requer dois acessos
- Em relação ao tempo, modo imediato é mais rápido, seguido de modo direto e por último o modo indireto



# Modos de Endereçamento

## Endereçamento por Registrador

- Semelhante aos modos direto e indireto, mas o endereço de memória é substituído por um endereço de registrador da UCP
- Menor número de bits na instrução
- Dado armazenado em um meio mais rápido

# Modos de Endereçamento

## Endereçamento por Registrador

**Comparação:**

**Do** I = 1 **to** 100

Read A,B

$X = A + B$

End

# Modos de Endereçamento

## Endereçamento por Registrador

Modo de endereçamento direto:

GET L	; ler valor do "loop" (no exemplo o valor é igual a 100)
LDI 0	; $ACC \leftarrow 0$
SUBM L	; $ACC \leftarrow ACC - (L)$ (no exemplo valor é 100)
In STA I	; $(I) \leftarrow ACC$ (inicialmente $I = -100$ )
JZ Fim	; se $ACC = 0$ vá para Fim
GET A	; ler valor do dado para o endereço A
GET B	; ler valor do dado para o endereço B
LDA A	; $ACC \leftarrow (A)$
ADD B	; $ACC \leftarrow ACC + (B)$
STR X	; $(X) \leftarrow ACC$
LDA I	; $ACC \leftarrow (I)$
INC	; $ACC \leftarrow ACC + 1$ (no exemplo estamos fazendo $I+1$ )
JMP In	; vá para In
Fim HLT	; parar

# Modos de Endereçamento

## Endereçamento por Registrador

Modo de endereçamento direto:

**Gastam-se 200 ciclos de memória apenas para carregar o acumulador com A e armazenar o valor do acumulador em X**

# Modos de Endereçamento

## Endereçamento por Registrador

Modo de endereçamento por registrador:

LDI R1, 0	; $R1 \leftarrow 0$
LDI R2, 100	; $R2 \leftarrow 100$
In SUBR R1, R2	; $(R1) \leftarrow (R1) - (R2)$
JZ R, Fim	; se $R1 = 0$ vá para Fim
GET A	; ler valor do dado para o endereço A
GET B	; ler valor do dado para o endereço B
LDA A	; $ACC \leftarrow (A)$
ADD B	; $ACC \leftarrow ACC + (B)$
STR X	; $(X) \leftarrow ACC$
INC R1	; $(R1) \leftarrow (R1) + 1$ (no exemplo estamos fazendo $I+1$ )
JMP In	; vá para In
Fim HLT	; parar

# Modos de Endereçamento

## Endereçamento por Registrador

As instruções:

```
LDI R1,1,  
LDI R2, 100  
SUBR R1, R2  
INC R1
```

Não acessam a memória principal

Não há vantagem em fazer  $MP \leftarrow R \leftarrow UAL$  nos casos dos valores A e B (não reduz ciclos de memória)

# Modos de Endereçamento

## Endereçamento por Registrador

Economia de bits

Instruções com uso de registradores:

C.Op. / R1 / R2

8 bits / 4 bits / 4 bits = 16 bits

Instruções com acesso a 64K células de MP:

C. Op. / Op. 1 / Op.2

8 bits / 16 bits / 16 bits = 40 bits

# Modos de Endereçamento

## Endereçamento por Registrador

Há duas maneiras de empregar o modo de endereçamento por registrador:

- Modo por **registrador direto**
- Modo por **registrador indireto**

Ocorre dificuldade em se definir quais dados serão armazenados em registradores e quais permanecerão em MP



# Modos de Endereçamento

## Endereçamento por Registrador

Dois tipos de instruções que usam o modo registrador:

C. Op.	R1	R2
(a)		
(a) ADD	R1, R2	(R1) ← (R1) + (R2)
(b) ADR	R, Op.	(R) ← (R) + (Op.) ou (R) (R) ← (R) + Op.
(a) LDR	R1, R2	(R1) ← ((R2))

(Fig. 8.8 do livro texto)

# Modos de Endereçamento

## Modo Indexado

- Permite manipular endereços para facilitar acesso a vetores
- O endereço do dado (elemento do array) relaciona-se com o seu índice
- O endereço do dado é a soma do valor do campo operando (fixo) e de um valor armazenado em um dos registradores da UCP (registrador índice)

# Modos de Endereçamento

## Modo Indexado

**Do I = 1 To 100**

$C(I) = A(I) + B(I)$

# Modos de Endereçamento

## Modo Indexado

Usando o Modo Direto sem alterar os bits que descrevem as instruções

```
LDA A(1)
ADD B(1)
STA C(1)
LDA A(2)
ADD B(2)
STA C(2)
-----
-----
LDA A(100)
ADD B(100)
STA C(100)
HLT
```

(Fig. 8.9 do livro texto)

Instruções para cada uma das 100 operações de soma a serem executadas

# Modos de Endereçamento

## Modo Indexado

Usando o Modo Direto com alteração dinâmica do conteúdo de instruções

Programa Assembly	Programa em linguagem de máquina
<b>T LDA A(1)</b>	<b>11A00</b>
<b>1 ADD B(1)</b>	<b>21A64</b>
<b>2 STA C(1)</b>	<b>31AC8</b>
<b>INC T</b>	<b>8103A</b>
<b>INC 1</b>	<b>8103B</b>
<b>INC 2</b>	<b>8103C</b>
<b>DCR N</b>	<b>919FF</b>
<b>JNZ T</b>	<b>D103A</b>
<b>END</b>	<b>F0000</b>

(Fig. 8.10 do livro texto)

# Modos de Endereçamento

## Modo Indexado

Usando o Modo Direto com alteração dinâmica do conteúdo de instruções

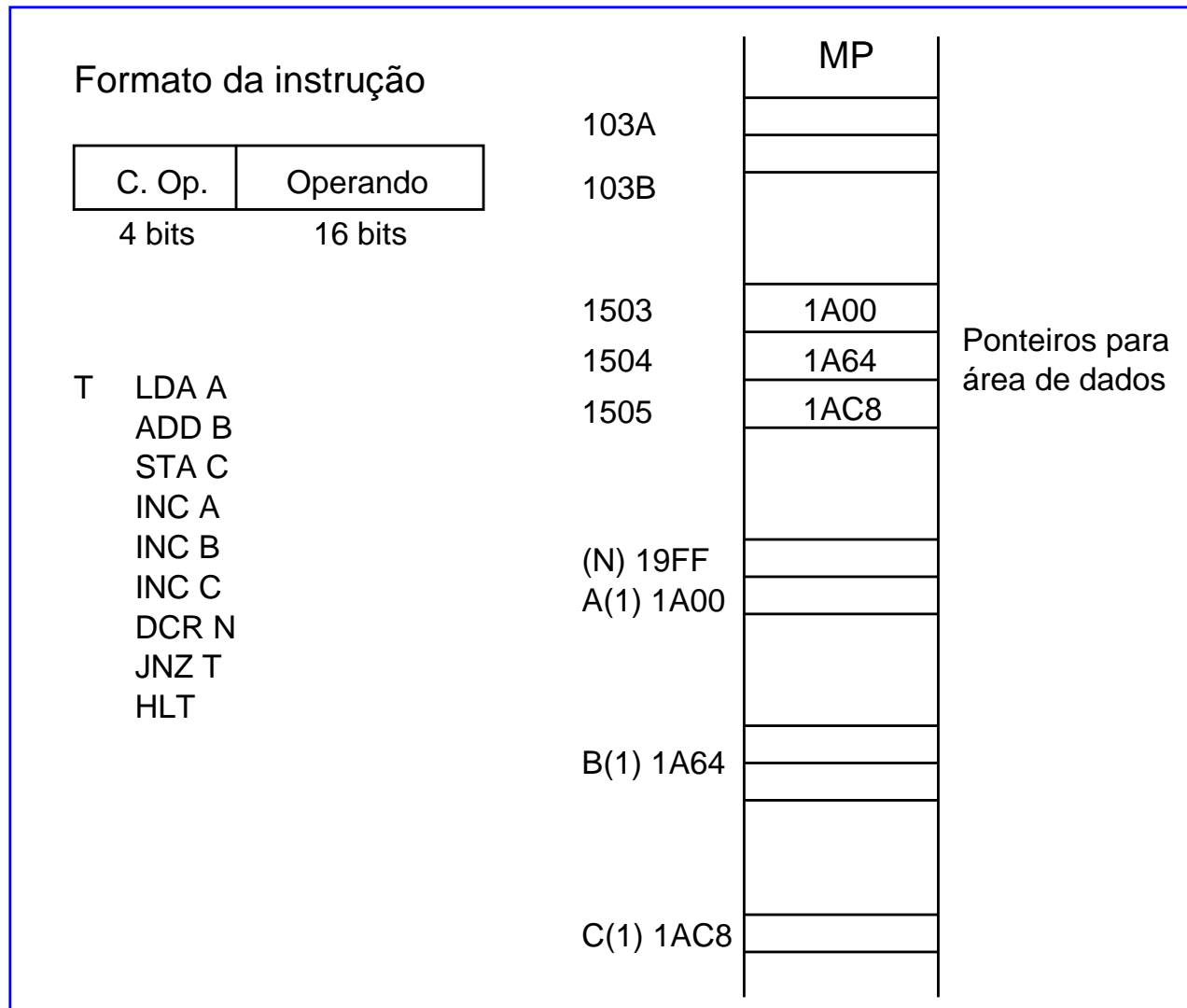
Formato da instrução			MP
C. Op.	Operando	103A	11A00
		103B	21A64
4 bits	16 bits	Cont. do trecho do progr.	
LDA Op. - C. Op. = 1		(N) 19FF	
ADD Op. - C. Op. = 2		A(1) 1A00	
STA Op. - C. Op. = 3		A(2) 1A01	
-----		A(3) 1A02	
INC Op. - C. Op. = 8		" "	
DCR Op. - C. Op. = 9			
-----			
JMP Op. - C. Op. = C		B(1) 1A64	
JNZ Op. - C. Op. = D		B(2) 1A65	
HLT - C. Op. = F		" "	
		C(1) 1AC8	
		C(2) 1AC9	
		" "	

(Fig. 8.11 do livro texto)

# Modos de Endereçamento

## Modo Indexado

Usando o Modo Indireto



(Fig. 8.12 do livro texto)

# Modos de Endereçamento

## Modo Indexado

Usando o Modo Indexado

```

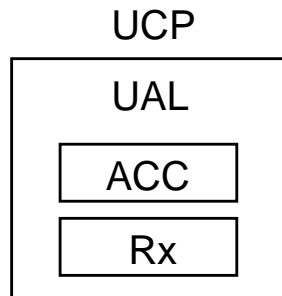
MVI (R4), 1
MVI (R2), 100
T LDA (R4), 19FF
ADD (R4), 1A63
STA (R4), 1AC7
INC (R4)
DCR (R2)
JZR (R2), T
HLT
```

(Fig. 8.13 do livro texto)



# Modos de Endereçamento

## Modo Indexado



Assumimos que um dos 16 reg. da UCP será usado como Rx. Seu endereço consta do campo R da instrução.

C. Op.	R	Operando
4 bits	4	16 bits

0 - MVI Rx, Op.	$(Rx) \leftarrow Op.$
1 - LDA Rx, Op.	$ACC \leftarrow (Op. + (Rx))$
2 - STA Rx, Op.	$(Op. + (Rx)) \leftarrow ACC$
3 - ADD Rx, Op.	$ACC \leftarrow ACC + (Op. + (Rx))$
4 - JMP Op.	$CI \leftarrow Op.$
5 - JZR Rx, Op.	$CI \leftarrow Op., \text{ se } (Rx) = 0$
6 - DCR Rx	$(Rx) \leftarrow (Rx) - 1$
7 - INC Rx	$(Rx) \leftarrow (Rx) + 1$
F - END	Parar

	MP
103A	
	"
	"
	"
A(1) 1A00	
	"
B(1) 1A64	
	"
	"
	"
C(1) 1AC8	

(Fig. 8.14 do livro texto)

# Modos de Endereçamento

## Modo Base Mais Deslocamento

- Tem características semelhantes ao modo indexado -  
 $\text{endereço} = \text{deslocamento} + \text{registrador base}$
- Registrador base fixo e variação do deslocamento
- Redução do tamanho da instrução

# Modos de Endereçamento

## Modo Base Mais Deslocamento

Sua escolha decorre de:

- Durante a execução de uma grande quantidade de programas, as referências as células de memória, normalmente são seqüenciais
- A maioria dos programas ocupa um pequeno espaço da MP disponível
- Redução do tamanho da instrução

# Modos de Endereçamento

## Modo Base Mais Deslocamento

### Exemplo:

- Campo de registrador base de 4 bits (16 registradores) e campo de deslocamento de 12 bits = 16 bits
- Processador pode endereçar 16 M – 24 bits
- Por registrador base pode-se endereçar 4096 bytes (4 K)
- Economia de 8 bits

# Modos de Endereçamento

## Modo Base Mais Deslocamento

Diferenças com o modo indexado:

- Indexação é empregada quando se deseja acessar diferentes dados com alteração do endereço por incremento ou decremento do registrador índice
- Quando a modificação do endereço é realizada para relocação do programa, basta uma única alteração no registrador base

# Modos de Endereçamento

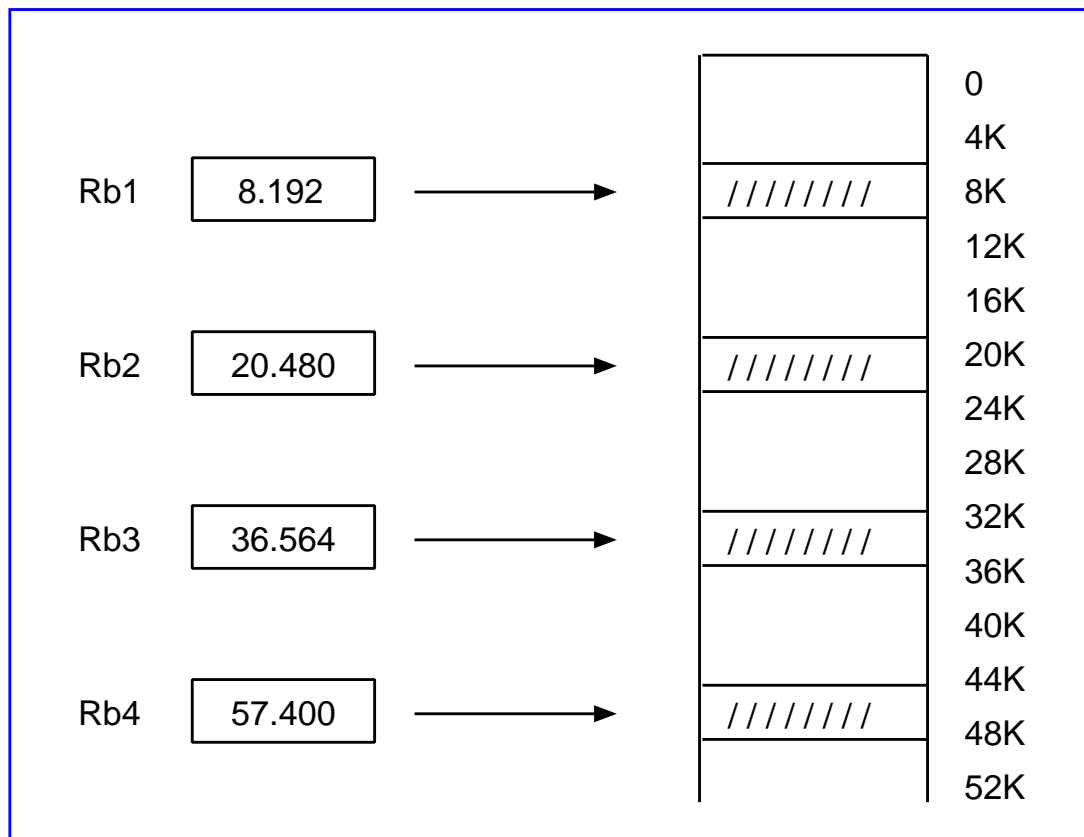
## Modo Base Mais Deslocamento

Diferenças com o modo indexado:

- Vários dados são acessados com diversos valores de registrador índice e um único valor de campo operando
- Vários dados são acessados com único valor de registrador base e valores diferentes no campo deslocamento

# Modos de Endereçamento

## Modo Base Mais Deslocamento



(Fig. 8.15 do livro texto)

# Representação de Instruções

## Exercícios

Todos os exercícios do Capítulo 8 do Livro Texto



## Aula 8

### Professores:

Lúcia M. A. Drummond  
Simone de Lima Martins

### Conteúdo:

#### Execução de Programas

- Introdução
- Linguagens de programação
- Montagem e Compilação
- Ligação ou Linkedição
- Execução de programas em código de máquina

# Introdução

- Programa de computador: conjunto de instruções ou comandos organizados em uma certa seqüência para realizar uma tarefa específica
- Atualmente, é raro escrever um programa em linguagem de máquina: problemas de manutenção e dificuldade de organizar as instruções sem erro.

# Linguagens de Programação

- Linguagem de programação: Linguagem criada para instruir um computador a realizar tarefas
- Código: programa completo, escrito em uma linguagem de programação

# Linguagem de Programação

## Linguagem de máquina:

- Tipo mais primitivo de linguagem de programação, com instruções diretamente executadas pela UCP
- Um programa de máquina é uma longa seqüência de algarismos binários

0010	0100	1001	0001	2 4 9 1
0100	0100	1001	1111	4 4 9 F
0100	0100	1001	0011	4 4 9 3
0001	0100	1001	0010	1 4 9 2
1000	0100	1001	1000	8 4 9 8
1110	0100	1001	1001	E 4 9 9
0011	0100	1001	0101	3 4 9 5
0100	0100	1001	1110	4 4 9 E
1111	0100	1001	1010	F 4 9 A
0000	0000	0000	0000	0 0 0 0

(a) Programa em linguagem binária      (b) Programa em hexadecimal

(Fig. 9.1 do livro texto)

# Linguagem de Programação

## Dificuldades de utilização de linguagem de máquina:

- O programador deverá conhecer todas as instruções disponíveis para aquela máquina - códigos de operação e formatos, registradores da UCP, endereços de células de memória onde estão os dados e instruções
- Programa poderá conter milhares de instruções - tarefa tediosa e difícil

# Linguagens de Programação

## Linguagem Assembly:

- Códigos de operação como 0101 são mais fáceis de ser lembrados como ADD
- O programador não precisa mais guardar os endereços reais em memória onde dados e instruções estão armazenados - usa símbolos
- Linguagem de máquina: 1110 01001001 1001 ou E499  
Linguagem Assembly: ADD SALARIO

# Linguagens de Programação

## Assembler (montador)

- Traduz o programa assembly em código de máquina para ser executado pela UCP
- Lê cada instrução em linguagem de montagem e cria uma instrução em linguagem de máquina

ORG	ORIGEM
LDA	SALARIO - 1
ADD	SALARIO - 2
ADD	SALARIO - 3
SUB	ENCARGO
STA	TOTAL
HLT	
DAD	SALARIO - 1
DAD	SALARIO - 2
DAD	SALARIO - 3

(Fig. 9.2 do livro texto)

# Linguagens de Programação

## Linguagem Assembly:

- Escrever programas ainda é uma tarefa árdua, tediosa e complexa
- Ainda é mais atraente do que escrever um programa em linguagem de máquina



# Linguagens de Programação

## Linguagens de alto nível:

- Refletem os procedimentos utilizados na solução de problemas, sem preocupação com o tipo de UCP ou de memória onde o programa será executado
- Mais simples, menos instruções do que Assembly
- Estruturadas de acordo com a compreensão e intenção do programador - linguagem de alto nível, nível afastado da máquina

# Linguagens de Programação

Linguagem	Data	Observações
FORTTRAN	1957	FORMula TRANslation - primeira linguagem de alto nível. Desenvolvida para realização de cálculos numéricos.
ALGOL	1958	ALGOrithm Language - linguagem desenvolvida para uso em pesquisa e desenvolvimento, possuindo uma estrutura algorítmica.
COBOL	1959	COMmon Business Oriented Language - primeira linguagem desenvolvida para fins comerciais.
LISP	1960	Linguagem para manipulação de símbolos e listas.
PL/I	1964	Linguagem desenvolvida com o propósito de servir para emprego geral (comercial e científico). Fora de uso.
BASIC	1964	Desenvolvida em Universidade, tornou-se conhecida quando do lançamento do IBM-PC, que veio com um interpretador da linguagem, escrito por Bill Gates e Paul Allen.
PASCAL	1968	Primeira linguagem estruturada - designação em homenagem ao matemático francês Blaise Pascal que, em 1642, foi o primeiro a planejar e construir uma máquina de calcular.
C	1967	Linguagem para programação de sistemas operacionais e compiladores.
ADA	1980	Desenvolvida para o Departamento de Defesa dos EUA.
DELPHI	1994	Baseada na linguagem Object Pascal, uma versão do Pascal orientada a objetos.
JAVA	1996	Desenvolvida pela Sun, sendo independente da plataforma onde é executada. Muito usada para sistemas Web.

(Tabela 9.1 do livro texto)

# Linguagens de Programação

DETAIL PARAGRAPH.

READ CARD-FILE AT END GO TO END-PARAGRAPH.

MOVE CORRESPONDING CARD-IN TO LINE-OUT.

ADD CURRENT-MONTH-SALES IN CARD-IN, YEAR-TO-DATE-SALES IN  
CARD-IN GIVING TOTAL-SALES-OUT IN LINE-OUT.

WRITE LINE-OUT BEFORE ADVANCING 2 LINES AT EOP

PERFORM HEADER-PARAGRAPH

**(a)** Trecho de programa em COBOL.

```
detail_procedure ( )
{
    int file_status, current_line, id_um;
    char *name;
    float current_sales, ytd_sales, total_sales;

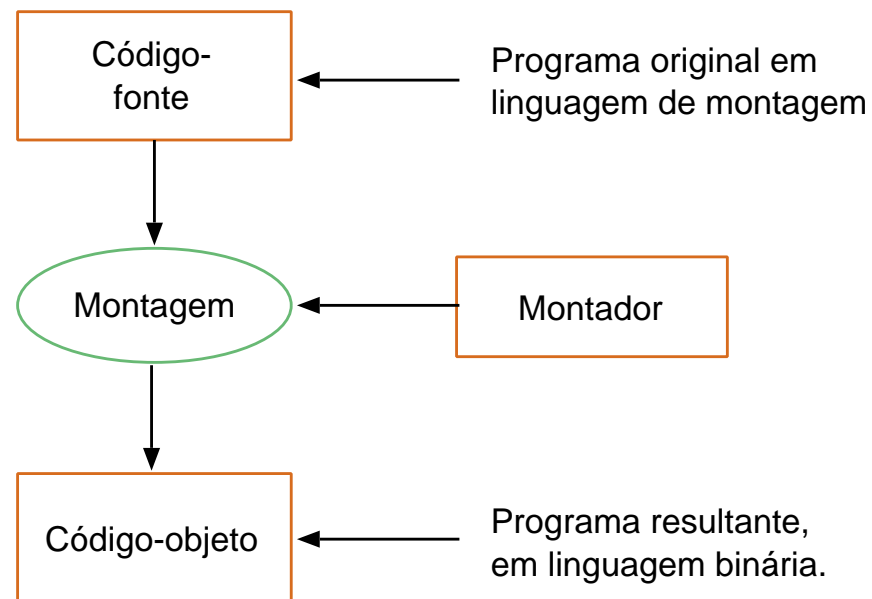
    current_line=64;
    file_status=fscanf (card_file, "%d%s%f%f", &id_num, &name, &current_sales,
        &ytd_sales);
    while (file_status !=EOF) {
        if (current_line > 63) {
            header_procedure ( );
            current_line=3;
        }
        file_status=fscanf (card_file, "%d%s%f%f", &id_num, &name, &current_sales,
            &ytd_sales);
        total_sales = current_sales + ytd_sales;
        printf ("%4d%30s%6.2f%7.2f\n\n", id_num, name, current_sales,
            ytd_sales, total_sales);
        current_line ++; current_line ++;
    }
end_procedure ( );
```

**(b)** Trecho de programa em C.

(Fig. 9.3 do livro texto)

# Montagem

- Tradução mais rápida e simples
- Realizada pelo Montador - Assembler
- Utilizada para traduzir um programa em linguagem de montagem (assembly) para seu equivalente em binário



(Fig. 9.4 do livro texto)

# Montagem

## Funções de um montador:

- Substituir códigos de operação simbólicos por valores numéricos.  
**Ex:** LOAD - 00101101, STR - 100010, ADD - 00001110,  
MOV - 11111100
- Substituir nomes simbólicos de endereços por valores numéricos dos endereços.  
**Ex:** ADD SOMA por 00001110 1011100011010
- Converter valores de constantes para valores binários.
- Examinar a correção de cada instrução. **Ex:** LDA pode ser uma instrução correta, mas LDB não. O montador não pode gerar código de operação inexistente

# Montagem

0000		ADDS	PROC	NEAR
0000	03 C3		ADD	Ax, Bx
0002	03 C1		ADD	Ax, Cx
0004	03 C2		ADD	Ax, Dx
0006	C3		RET	
0007		ADDS	ENDP	

(Fig. 9.5 do livro texto)

C. Op.	Descrição		
HLT	Parar		
INC	ACC	←	ACC + 1
DCR	ACC	←	ACC - 1
LDA Op.	ACC	←	M (Op.)
STR Op.	M (Op.)	←	ACC
ADD Op.	ACC	←	ACC + M (Op.)
SUB Op.	ACC	←	ACC - M (Op.)
JMP Op.	CI	←	Op.
JP Op.	Se ACC > 0, então: CI ← Op.		
JZ Op.	Se ACC = 0, então: CI ← Op.		

(a) Subconjunto de instruções

Início : ORG ZERO ; Origem do programa. Endereço relativo Ø.  
LDA CONTADOR ; Carregar valor do contador no ACC.  
JZ FIM ; Se contador = 0, então PARAR (desvia para FIM).  
LDA Parcela 1  
ADD Parcela 2 } ; Realizar operação aritmética com dados.  
STR Resultado  
LDA Contador ; Ler valor do contador para ACC.  
DCR Zero ; Substituir 1 do contador.  
JMP Início ; Voltar para início do loop.  
Fim : HLT ; Parar.  
DAD Parcela 1  
DAD Parcela 2  
DAD Resultado  
DAD Contador

(b) Programa em linguagem de montagem

(Fig. 9.6 do livro texto)

# Montagem

## Dois tipos básicos de símbolos:

- Códigos de operação. **Ex:** LDA, STR, JZ
- Endereços de dados ou de instruções. **Ex:** INICIO, FIM, CONTADOR, PARCELA 1



# Montagem

## Montador de 2 passos:

- O programa é examinado instrução por instrução duas vezes
- **Primeiro passo:** detecta erros, monta tabela de símbolos de endereços
- **Segundo passo:** cria o código objeto

# Montagem

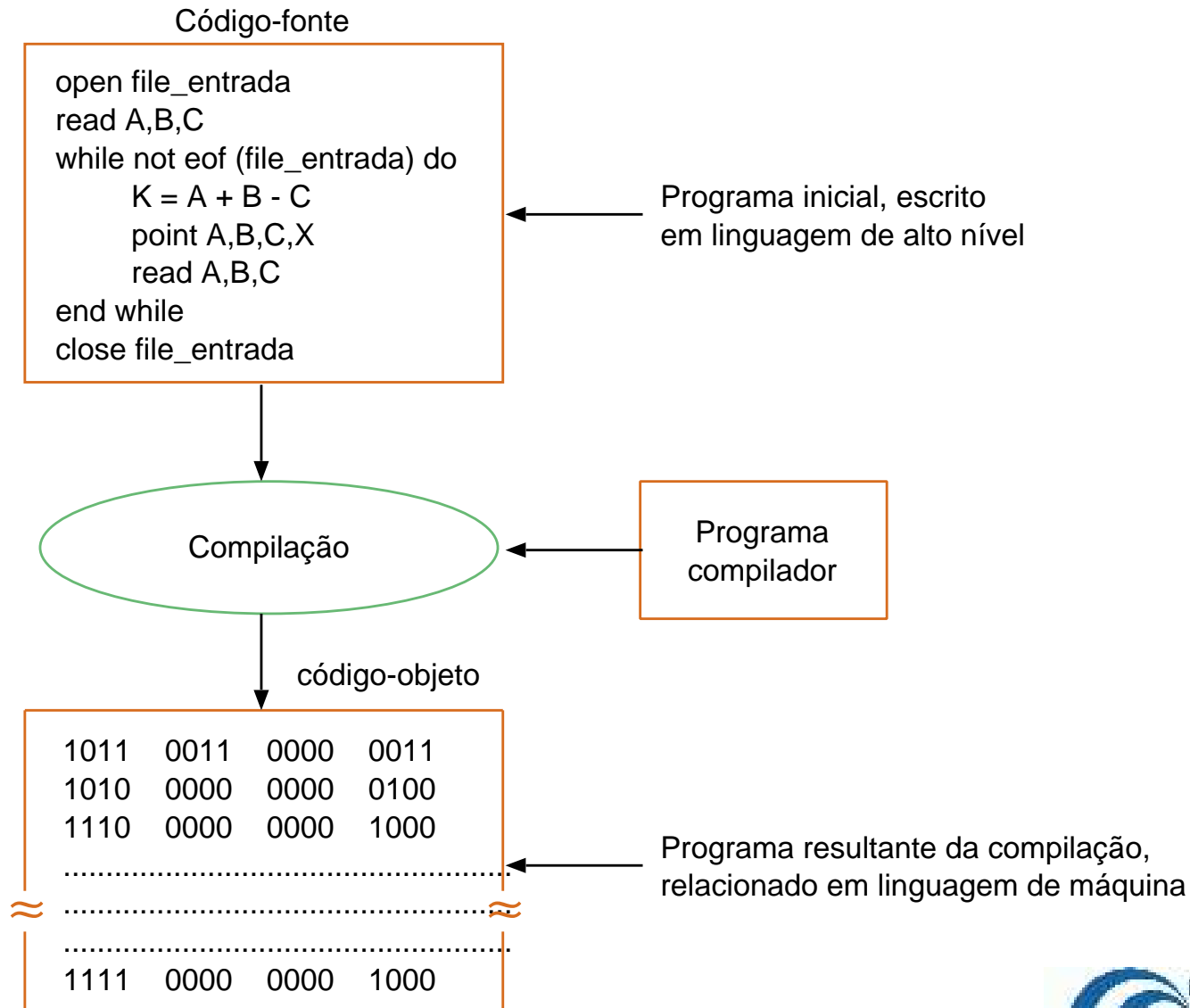
## Montador de um passo:

- Não tem a mesma clareza de execução do anterior
- Muitas instruções não podem ser montadas porque os endereços não são conhecidos e precisam ser completadas depois
- Se a tabela de endereços desconhecidos for grande (muitas referências a endereços inexistentes), a busca à suas diversas entradas será demorada, tão demorada quanto se tivesse realizado um segundo passo.

# Compilação

- Processo de análise de um programa escrito em linguagem de alto nível (programa fonte) e sua tradução em um programa em linguagem de máquina (programa objeto)
- Compilador: programa que realiza tal tarefa

# Compilação



(Fig. 9.7 do livro texto)

# Compilação

Inicialmente ocorre a análise do código fonte (front-end):

- Análise léxica
- Análise sintática
- Análise semântica

# Compilação

## Análise Léxica

- Consiste em decompor o programa fonte em seus elementos individuais (comandos, operadores, variáveis, etc), que são verificados de acordo com as regras da linguagem, gerando erros se for encontrada alguma incorreção

# Compilação

## Análise Sintática

- Consiste basicamente na criação das estruturas de cada comando, na verificação da correção dessas estruturas e na alimentação da tabela de símbolos com as informações geradas
- Monta uma árvore de análise (parse tree) para o programa fonte por inteiro de acordo com as regras gramaticais da linguagem





# Compilação

## Análise Sintática

- Com a árvore, é permitida a criação de código de máquina e verificação da correção do comando conforme as regras gramaticais especificadas na definição da linguagem
- A tabela de símbolos contém entradas para cada identificador e cada literal usado no programa fonte. **Ex:** os nomes das variáveis terão como atributo seu tipo, isto é se é inteiro, se é fracionário, etc... De modo que a operação aritmética a ser realizada com ele siga o algoritmo apropriado.

# Compilação

## Análise Semântica

- Verifica as regras semânticas estáticas, podendo produzir mensagens de erros
- Regras semânticas estáticas: regras que podem ser verificadas durante o processo de compilação, não é necessária a execução

**Exemplo:** atribuição de dado em uma expressão. O tipo de dado tem que ser coerente com o que foi declarado

# Compilação

- Módulo front-end (analísadores léxico, sintático e semântico) cria tabela de símbolos
- Módulo back-end aloca espaço de memória, define que registradores serão usados e que dados serão armazenados nele, gerando o código objeto em linguagem de máquina

# Compilação

## Exemplo:

Trecho de data division de programa em Cobol:

77 A PIC 9(6)

77 B PIC 9(6)

77 C PIC 9(6)

Nome	Tipo	Tamanho	Endereço
A	Inteiro	4	1000
B	Inteiro	4	1004
C	Inteiro	4	1008

(Fig. 9.9 do livro texto)

# Compilação

## Considerações:

- O código objeto pode ser absoluto, endereços reais de memória, ou relocável, endereços são relativos ao início do programa, transformando-se em endereços reais apenas na execução
- Na prática os compiladores ignoram muitas das instruções de máquina existentes, sendo este fato uma das desvantagens alegadas para máquinas CISC em relação às arquiteturas RISC

# Ligação ou Linkedição

- O código binário para algumas operações já existe armazenado no sistema. **Exemplos:** Comandos de entrada/saída, rotinas matemáticas especiais como seno
- Rotinas externas são organizadas em arquivos que constituem diretórios chamados de bibliotecas (libraries)
- Ligador ou linkeditor: faz a interpretação à chamada a uma rotina e respectiva conexão entre o código objeto principal e o código da rotina

# Ligação ou Linkedição

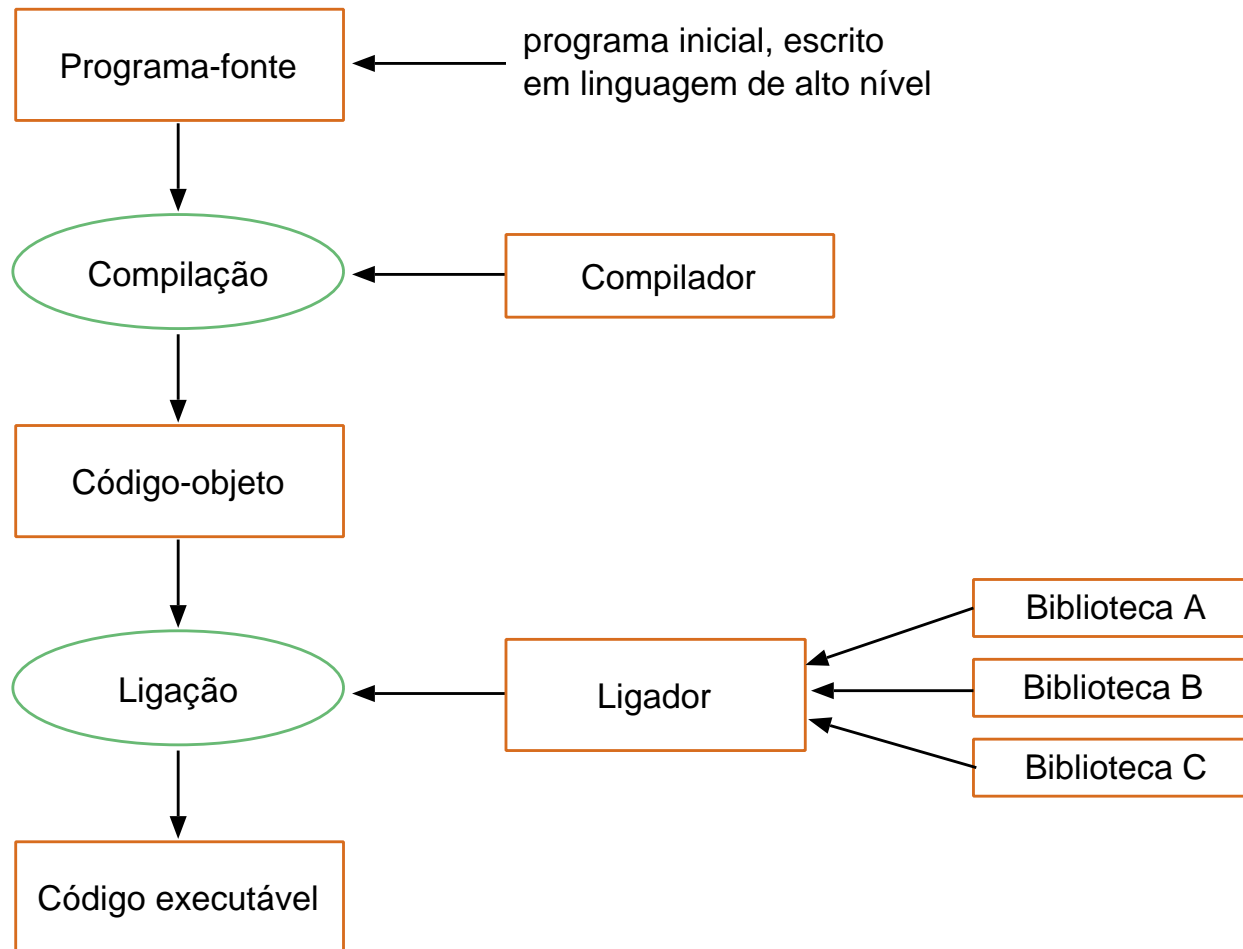
Considere o seguinte comando PASCAL:

$$X := A * B$$

- Suponha que A, B e X sejam números representados em ponto flutuante e que o compilador somente suporte aritmética de ponto-flutuante via rotina externa. O compilador irá inserir:

CALL MPY\_FP(1AB5, 1AB9, 1ABF), onde MPY\_FP é um índice para biblioteca, onde se encontra um programa já compilado e os números entre parênteses são endereços de A, B e X

# Ligação ou Linkedição



(Fig. 9.10 do livro texto)



# Ligação ou Linkedição

## Ligador:

- examina o código-objeto
- procura referências externas não resolvidas
- procura suas localizações nas bibliotecas
- substitui a linha de chamada pelo código da rotina
- emite mensagem de erro em caso de não encontrar a rotina

# Ligação ou Linkedição

## Execução de Programa:

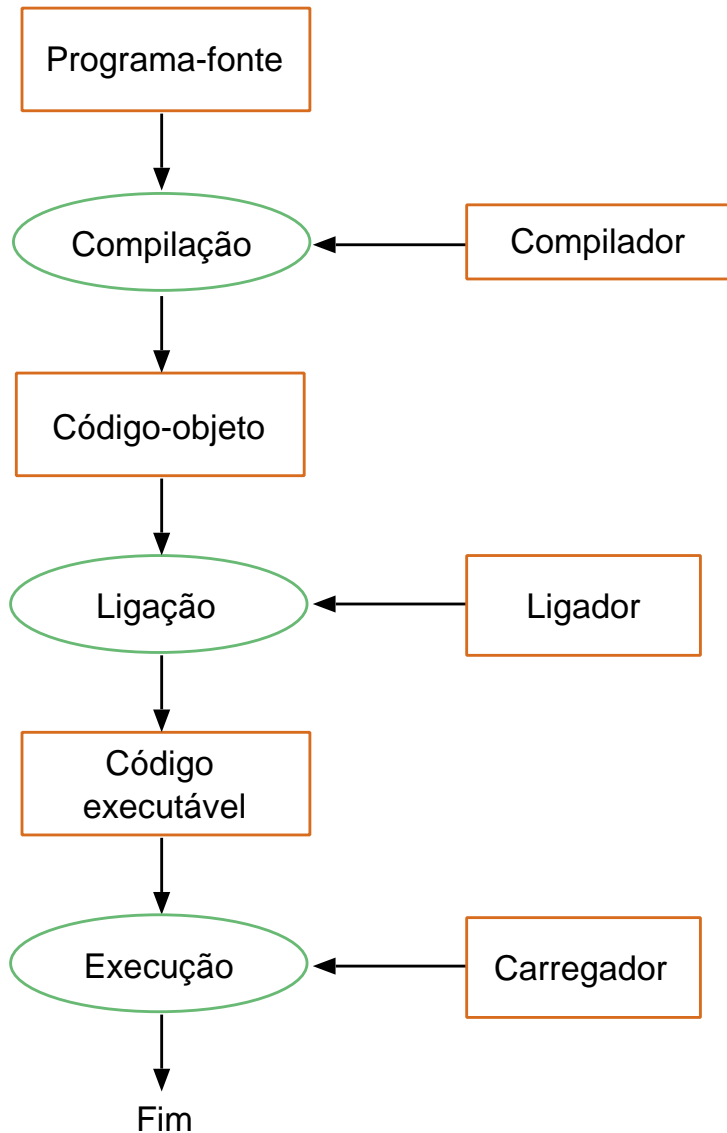
1. Armazenamento na MP do código fonte e compilador
2. O compilador gera o código objeto
3. Este arquivo pode ficar em memória secundária para ser ligado mais tarde ou ser imediatamente carregado na memória com o ligador
4. Após a ligação está formado o executável que pode ficar em memória secundária para posterior execução ou em MP para execução imediata.

# Ligação ou Linkedição

## Loader:

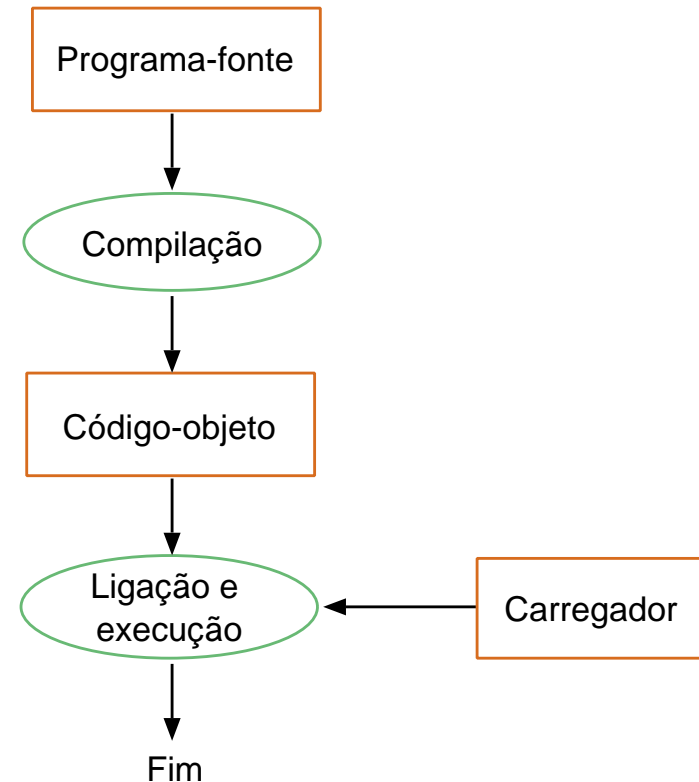
- Realiza em seqüência imediata as duas tarefas - ligação e execução do código de máquina, sem geração de código executável permanente
- Cria o executável sem armazená-lo e imediatamente inicia a execução

# Ligação ou Linkedição



**(a)** Com programas compilador, ligador e carregador distintos

(Fig. 9.11 do livro texto)



**(b)** Com programas compilador e carregador apenas

# Interpretação

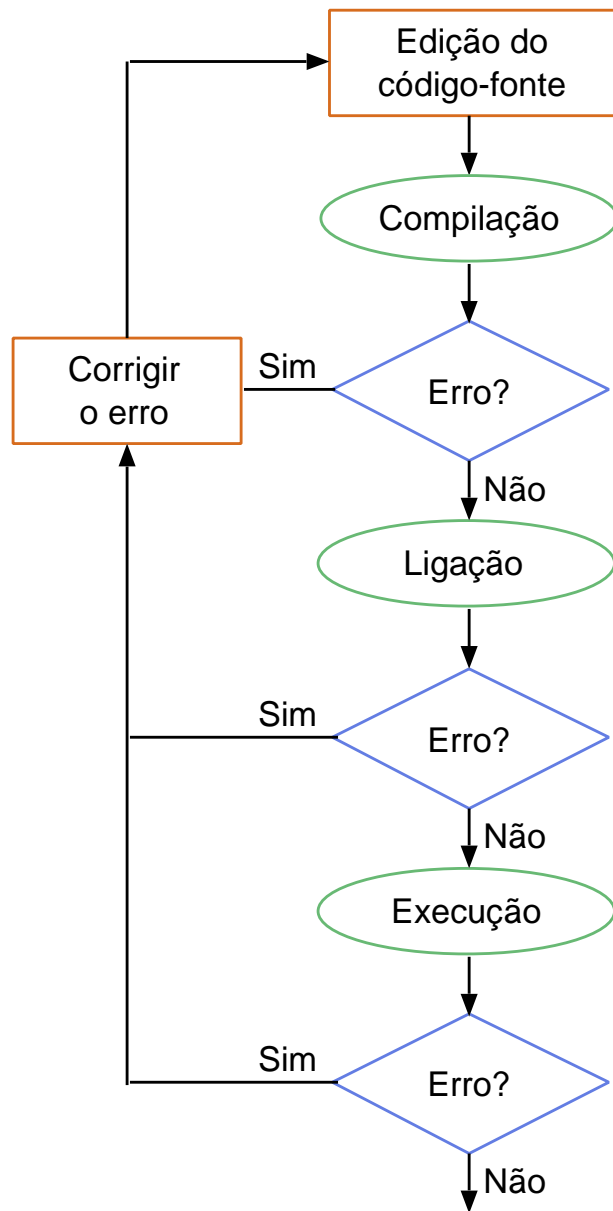
- Realiza as 3 fases, compilação, ligação e execução, comando a comando do programa-fonte
- Não há um processo explícito de compilação e ligação
- Não há produtos intermediários: código-objeto e código executável
- Um programa-fonte é diretamente interpretado pelo interpretador e produz o resultado
- Cada comando do código fonte é lido pelo interpretador, convertido em código executável e imediatamente executado antes do próximo comando

# Compilação X Interpretação

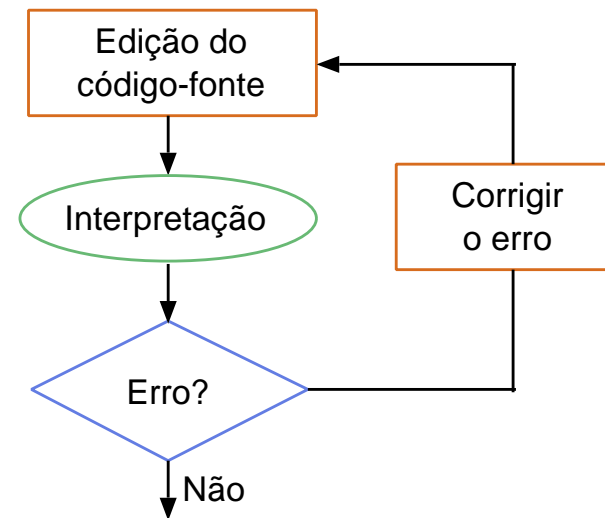
Recursos	Compilação	Interpretação
Uso da memória (durante a execução)		
- Interpretador ou compilador	Não	Sim
- Código-fonte	Não	Parcial
- Código executável	Sim	Parcial
- Rotinas de bibliotecas	Só as necessárias	Todas
Instruções de máquina (durante a execução)		
- Operações de tradução	Não	Sim
- Ligação de bibliotecas	Não	Sim
- Programa de aplicação	Sim	Sim

(Tabela 9.2 do livro texto)

# Compilação X Interpretação



(a) Compilação



(b) Interpretação

(Fig. 9.12 do livro texto)

# Compilação X Interpretação

Exemplo:

```
FOR J=1 TO 1000  
  BEGIN  
    READ A,B;  
    X := A + B;  
    PRINT X;  
  END  
END
```



# Compilação X Interpretação

## Compilação:

- os três comandos dentro do loop serão convertidos uma única vez para código executável
- Identificação do erro em tempo de execução é mais difícil

## Interpretação:

- 1000 conversões do fonte para o executável, consumindo mais CPU.
- Outro problema: programas executados freqüentemente (ex: folha de pagamento), tradução a cada execução

# Execução de Programas em Código de Máquina

```
Início do programa principal
X recebe 1
Y recebe 2
Z recebe a soma de X com Y
Se Z maior que zero
    Então: Z recebe o Quadrado (X)
    Senão: Z recebe o Quadrado (Y)
Fim do Se
Fim do programa principal

Função do Quadrado:
    Início função do Quadrado
    Retorne o produto do parâmetro da função com ele mesmo
Fim da função do Quadrado
```

Figura 9.13(a) Pseudocódigo de um algoritmo.

# Execução de Programas em Código de Máquina

## EXAMPLE1.PAS

```
(*  
* EXAMPLE1.PAS      Exemplo em linguagem pascal  
*)  
  
Program Example1;  
Var  
x, y, z  : integer;  
function func (w: integer) : integer;  
Begin  
    Func:= w * w;  
End;  
Begin  
    x:= 1;  
    y:= 2;  
    z:= x+y;  
    if (z > 0)  
        then z:= func (x)  
        else z:= func (y);  
End
```

Figura 9.13(b) Programa em Turbo Pascal Versão 5.0, que implementa o algoritmo da Fig. 9.13(a)

# Execução de Programas em Código de Máquina

```
int func (int w);

void main (void)
{
    int x, y, z;

    x = 1;
    y = 2;
    z = x + y;
    if (z > 0)
        z = func (x);
    else z = func (y);
}

int func (int w);
{
    return (w * w);
}
```

Figura 9.13(c) Implementação do algoritmo da Fig. 9.13(a) em Turbo C++.

# Execução de Programas em Código de Máquina

[EX1\_PAS.ASM]

\_func: function func (w: integer): integer;

cs: 0000	55	PUSH	BP
cs: 0001	89E5	MOV	BP, SP
cs: 0003	B80200	MOV	AX, 0002
cs: 0006	9A4402800D	CALL	0D80:0244 # Rotina auxiliar de inicialização
cs: 000B	83EC02	SUB	SP, +02
cs: 000E	8B4604	MOV	AX, [BP + 04] # func: = w * w;
cs: 0011	F76E04	IMUL	WORD PTR [BP + 04]
cs: 0014	8946FE	MOV	[BP-02], AX
cs: 0017	8B46FE	MOV	AX, [BP-02]
cs: 001A	89EC	MOV	SP, BP
cs: 001C	5D	POP	BP
cs: 001D	C20200	RET	0002
_Example1: Program Example1;			
cs: 0020	9A0000800D	CALL	0D80:0000 # Rotina auxiliar de inicialização
cs: 0025	55	PUSH	BP
cs: 0026	89E5	MOV	BP, SP
cs: 0028	C7063E000100	MOV	WORD PTR [003E], 0001 # x: = 1
cs: 002E	C70640000200	MOV	WORD PTR [0040], 0002 # y: = 2
cs: 0034	A13E00	MOV	AX, [003E] # z: = x + y;
cs: 0037	03064000	ADD	AX, [0040]
cs: 003B	A34200	MOV	[0042], AX
cs: 003E	833E420000	CMP	WORD PTR [0042], + 00 # if (z > 0)
cs: 0043	7E0C	JLE	0051
cs: 0045	FF363E00	PUSH	[003E] # then z: = func (x)
cs: 0049	E8B4FF	CALL	_func (0000)
cs: 004C	A34200	MOV	[0042], AX
cs: 004F	EB0A	JMP	005B
cs: 0051	FF364000	PUSH	[0040] # else z: func (y);
cs: 0055	E8A8FF	CALL	_func (0000)
cs: 0058	A34200	MOV	[0042], AX
cs: 005B	89EC	MOV	SP, BP
cs: 005D	5D	POP	BP
cs: 005E	31C0	XOR	AX, AX
cs: 0060	9AD800800D	CALL	0D80:00DB # Rotina auxiliar de finalização

Figura 9.13(ab) Programa gerado pelo compilador Pascal relativo à compilação do programa mostrado na Fig. 9.13(b).

# Execução de Programas em Código de Máquina

```

EX1_C.ASM

_main: void main (void)
cs: 02C2          55          PUSH      BP
cs: 02C3          88RC       MOV       BP, SP
cs: 02C5          83EC02     SUB       SP, + 02
cs: 02C8          56         PUSH      SI
cs: 02C9          57         PUSH      DI
# EXAMPLE1#11:  x = 1;
cs: 02CA          BF0100     MOV       DI, 0001
#EXAMPLE1#12:  y = 2;
cs: 02CD          C746FE0200 MOV       WORD PTR [BP - 02], 0002
#EXAMPLE1#13:  z = x + y;
cs: 02D2          8BC7       MOV       AX, DI
cs: 02D4          0346FE     ADD       AX, [BP - 02]
cs: 02D7          8BF0       MOV       SI, AX
#EXAMPLE1#14:  if (z > 0)
cs: 02D9          0BF6       OR        Si, SI
cs: 02DB          7E03       JLE       #EXAMPLE#16 (02E0)
#EXAMPLE1#15:  z = func (x);
cs: 02DD          57         PUSH      DI
cs: 02DE          EB03       JMP      02E3
#EXAMPLE#16:  z = func (y);
cs: 02E0          FF76FE     PUSH      [BP - 02]
cs: 02E3          E80900     CALL     _func (02EF)
cs: 02E6          59         POP       CX
cs: 02E7          8BF0       MOV       SI, X
#EXAMPLE#17: }
cs: 02E9          5F         POP      DI
cs: 02EA          5E         POP      SI
cs: 02EB          8BE5       MOV      SP, BP
cs: 02ED          5D         POP      BP
cs: 02EE          C3         RET
_func: int func (int w)
cs: 02EF          55         PUSH      BP
cs: 02F0          8BEC       MOV      BP, SP
cs: 02F2          8B5E04     MOV      BX, [BP + 04]
#EXAMPLE#21  return (w * w);
cs: 02F5          8BC3       MOV      AX, BX
cs: 02F7          F7EB       IMUL     BX
cs: 02F9          EB00       JMP      02FB
#EXAMPLE1#22: }
cs: 02FB          5D         POP      BP
cs: 02FC          C3         RET

```

Figura 9.13(ac) Programa gerado pelo compilador C relativo à compilação do programa mostrado na Fig. 9.13(c).

## EX1\_ASM.ASM

cs: 0100	50	PUSH	AX	#Salva os registradores a serem
cs: 0101	53	PUSH	BX	#usados na rotina
cs: 0102	51	PUSH	CX	
cs: 0103	52	PUSH	DX	
cs: 0104	B90100	MOV	CX, 0001	# x = 1
cs: 0107	BA0200	MOV	DX, 0002	# y = 2
cs: 010A	89C8	MOV	AX, CX	# z = x + y
cs: 010C	01D0	ADD	AX, DX	
cs: 010E	89C3	MOV	BX, AX	
cs: 0110	09DB	OR	BX, BX	#if (z > 0)
cs: 0112	7E03	JLE	0117	
cs: 0114	51	PUSH	CX	#Then x como parâmetro
cs: 0115	EB01	JMP	0118	
cs: 0117	52	PUSH	DX	#Else y como parâmetro
cs: 0118	E80600	CALL	0121	#Chama rotina int func (parâmetro)
cs: 011B	5B	POP	BX	#z = func (parâmetro)
cs: 011C	5A	POP	DX	#Restaura os registradores usados
cs: 011D	59	POP	CX	
cs: 011E	5B	POP	BX	
cs: 011F	58	POP	AX	
cs: 0120	C3	RET		#Fim da rotina
cs: 0121	55	PUSH	BP	#Início da rotina int func (parâmetro).
cs: 0122	89E5	MOV	BP, SP	
cs: 0124	8B4604	MOV	AX, [BP + 04]	#Recebe parâmetro
cs: 0127	F7E8	IMUL	AX	#Calcula o produto
cs: 0129	894604	MOV	[BP + 04], AX	#Prepara o valor do retorno
cs: 012C	5D	POP	BP	
cs: 012D	C3	RET		#Fim da rotina int func (parâmetro).

Figura 9.13(ad) Programa gerado pelo montador do DOS 5.0, relativo à montagem do programa mostrado na Fig. 9.13(c).

# Execução de Programas em Código de Máquina

Consideremos a expressão:

$$X = Y + Z - T$$

O programa imprime X se este for diferente de zero

```
REAL      X, Y, Z, T
X:        X + Z - T
IF        X <> 0
THEN      PRINT X
ELSE
END
```

(Fig. 9.14 do livro texto)



# Execução de Programas em Código de Máquina

	ORG	
	LDA	Y
	ADD	Z
	SUB	T
	STR	X
	JZ	FIM
	PRT	X
FIM	HLT	

Figura 9.15 Programa em linguagem de montagem para solucionar a expressão  $X = Y + Z - T$ .

# Execução de Programas em Código de Máquina

Para converter o referido programa em linguagem de máquina assumiremos:

- O processador/MP utilizado possui as mesmas características definidas no capítulo 6 e as mesmas instruções
- As variáveis usadas no programa são:

# Execução de Programas em Código de Máquina

variável	endereço	valor
Y	1F	051
Z	20	03E
T	21	003
X	22	01A

# Execução de Programas em Código de Máquina

- O programa está armazenado na MP a partir do endereço 18, e no instante inicial vamos considerar que:
  1.  $CI = 18$
  2. Os valores armazenados no RI e ACC são da instrução anterior, não importante para o início de nosso programa

# Execução de Programas em Código de Máquina

	ORG	
	LDA	Y
	ADD	Z
	SUB	T
	STR	X
	JZ	FIM
	PRT	X
FIM	HLT	

(Fig. 9.15 do livro texto)

ENDEREÇOS	CONTEÚDOS
18	11F
19	320
1A	421
1B	222
1C	51E
1D	B22
1E	000
1F	051
20	03E
21	003
22	01A

(Fig. 9.16 do livro texto)

## Execução de Programas em Código de Máquina

	ORG	
	<del>LDA</del>	Y
	ADD	Z
	SUB	T
	STR	X
	JZ	FIM
	PRT	X
FIM	HLT	

(Fig. 9.15 do livro texto)

ENDEREÇOS	CONTEÚDOS
18	11F
19	320
1A	421
1B	222
1C	51E
1D	B22
1E	000
1F	051
20	03E
21	003
22	01A

(Fig. 9.16 do livro texto)

### Instrução 1

CI	RI	ACC
19	11F	051

Próxima  
Instrução

Voltar

# Exercícios

*Fazer todos os exercícios do capítulo 9 do livro texto*

## Aula 9

### Professores:

Lúcia M. A. Drummond  
Simone de Lima Martins

### Conteúdo:

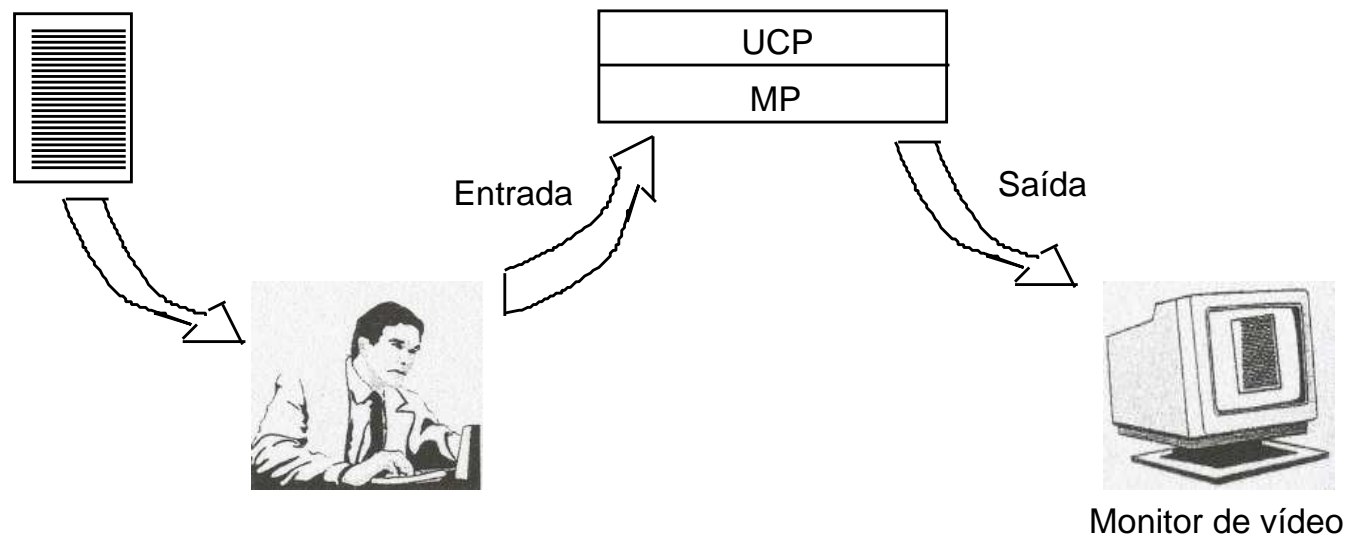
#### Entrada e Saída (E/S)

- Dispositivos externos
- Interface de E/S
- Operações de E/S



# Dispositivos externos

- Dispositivos externos são denominados dispositivos periféricos ou periféricos e permitem a comunicação da máquina com o ambiente externo
  - Dispositivos voltados para a comunicação com o usuário

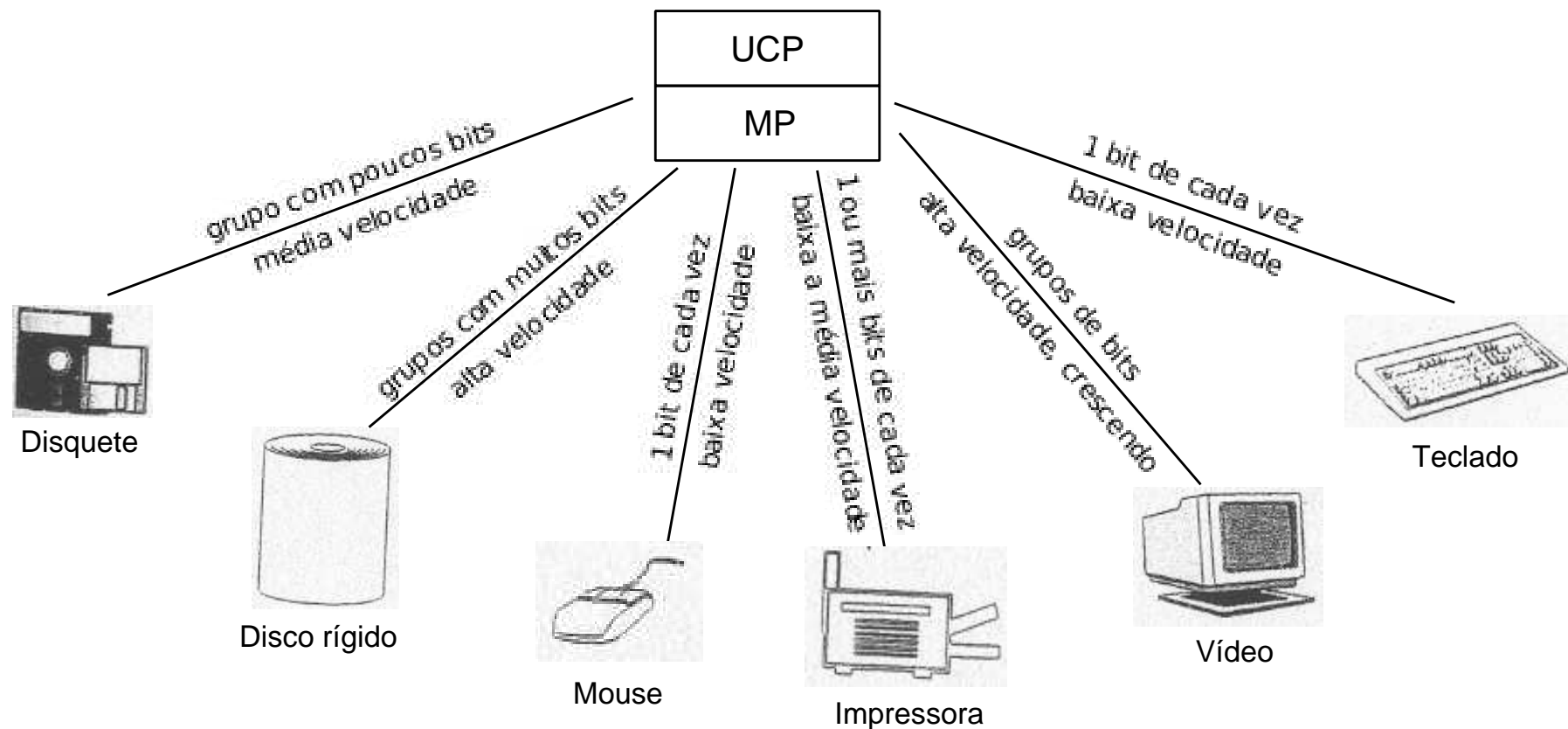


(Fig. 10.2 do livro texto)

- Dispositivos voltados para a comunicação com a máquina
  - Discos magnéticos e sensores
- Dispositivos voltados para a comunicação com dispositivos remotos
  - Modem e interface de rede

# Dispositivos externos

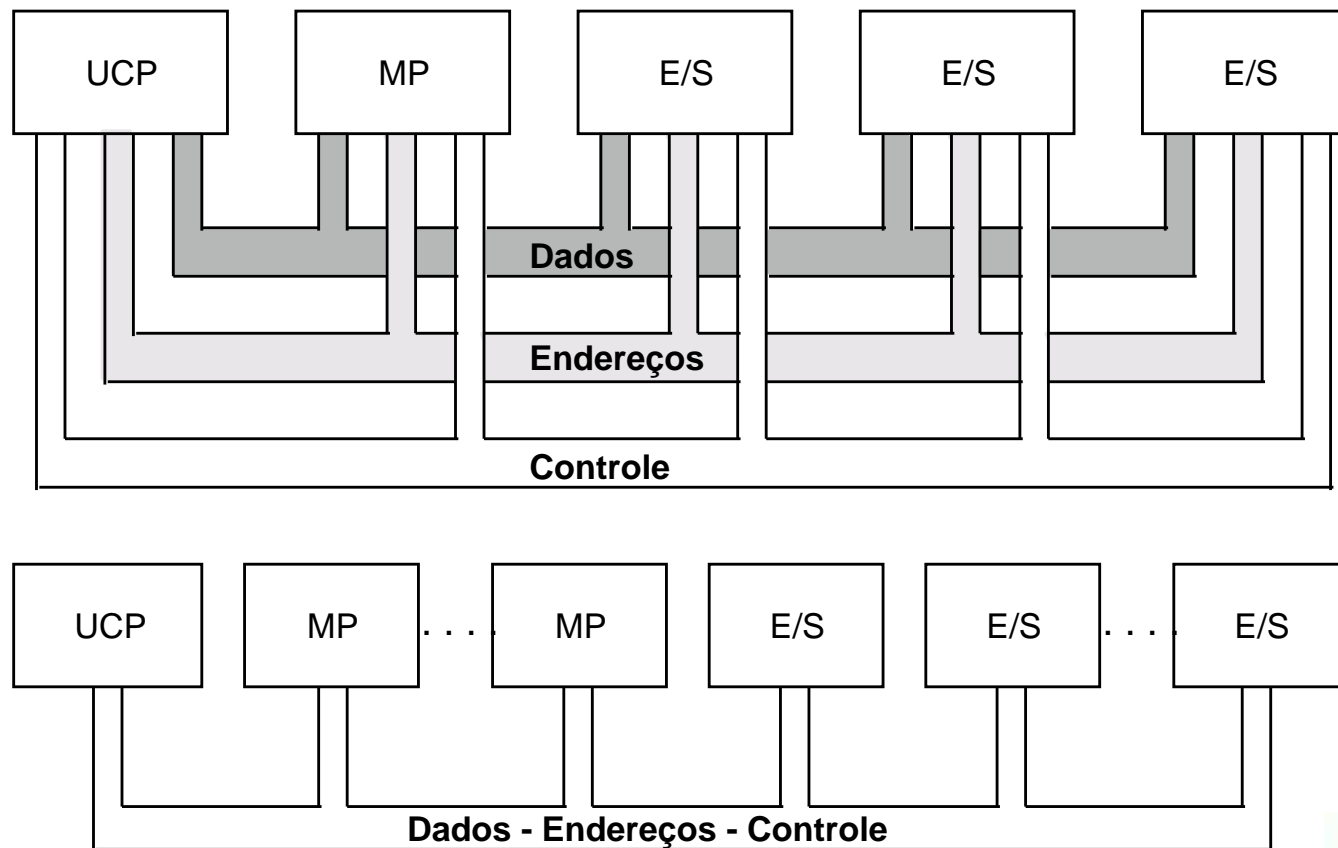
- Existe uma grande variedade de periféricos com diferentes formas de funcionamento



(Fig. 10.6 do livro texto)

# Dispositivos externos

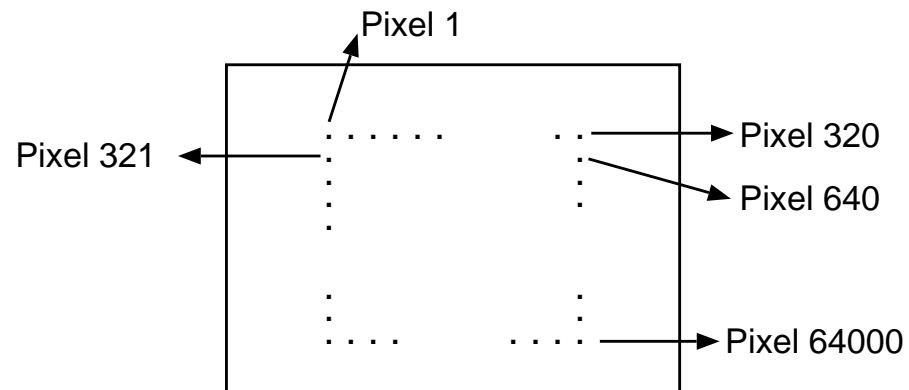
- Dispositivos de E/S são conectados a módulos de E/S que se comunicam com os outros componentes da máquina através do barramento



(Fig. 10.4 do livro texto)

# Dispositivos externos

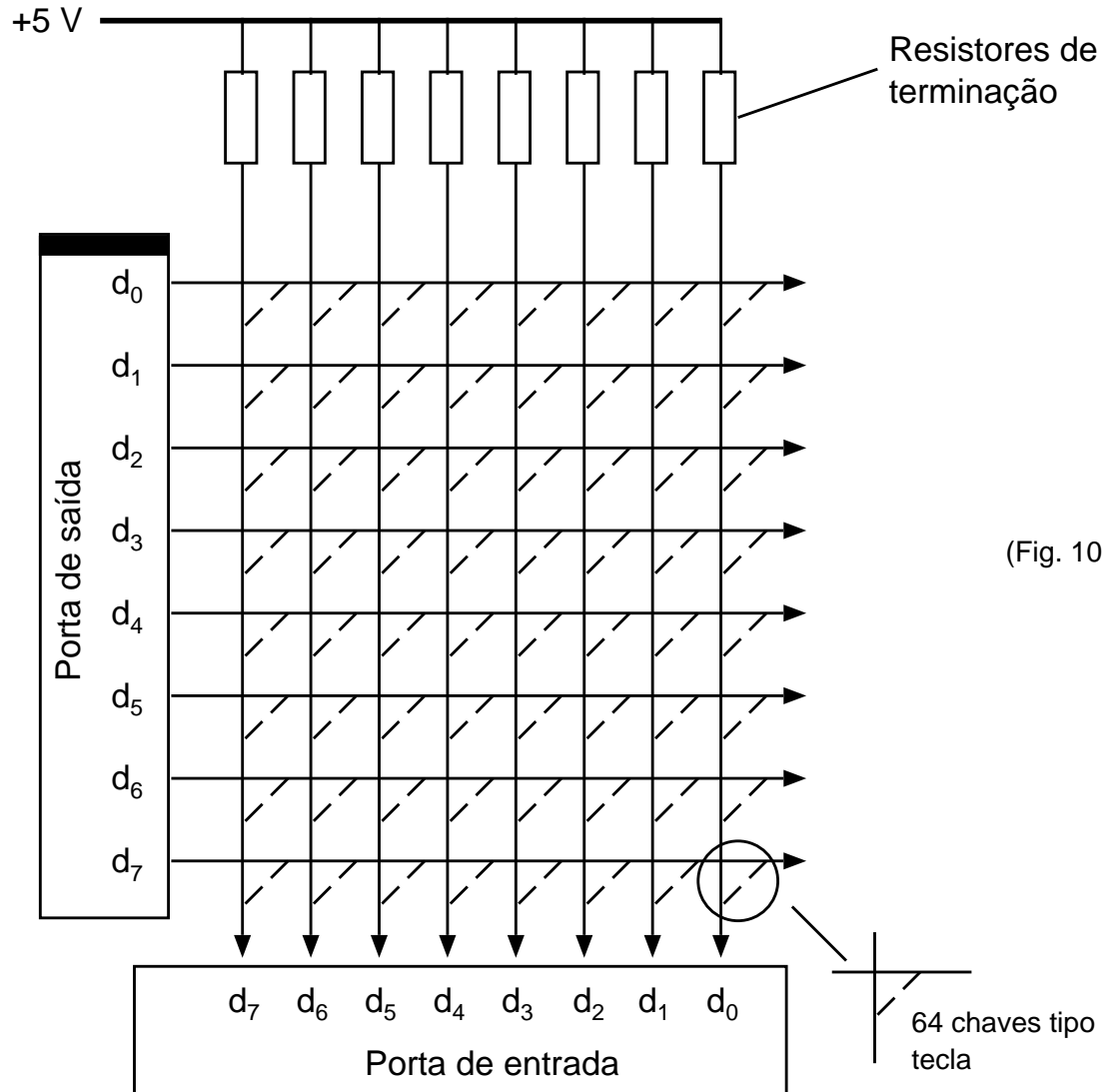
- Monitor de vídeo em modalidade gráfica



(Fig. 10.24 do livro texto)

# Dispositivos externos

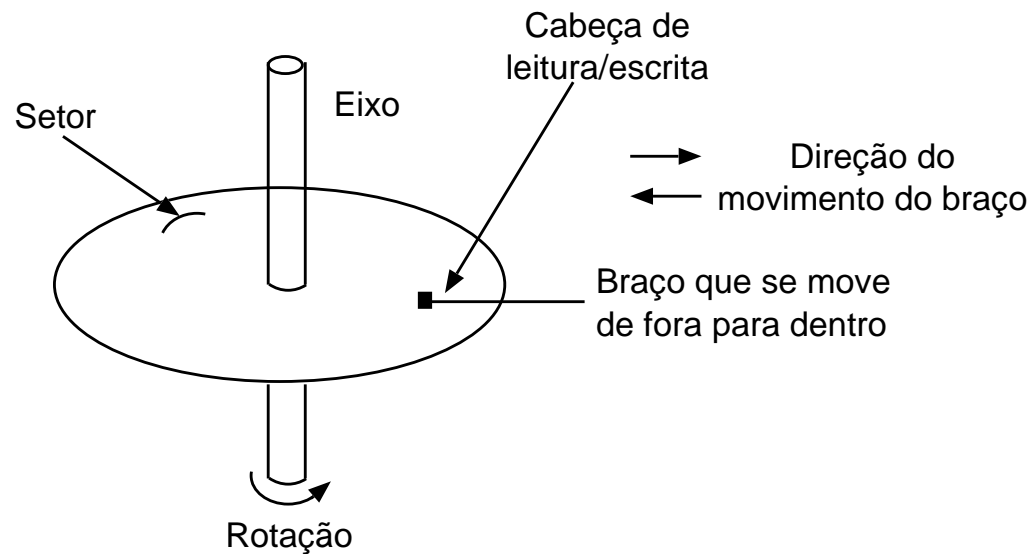
- Teclado



(Fig. 10.43 do livro texto)

# Dispositivos externos

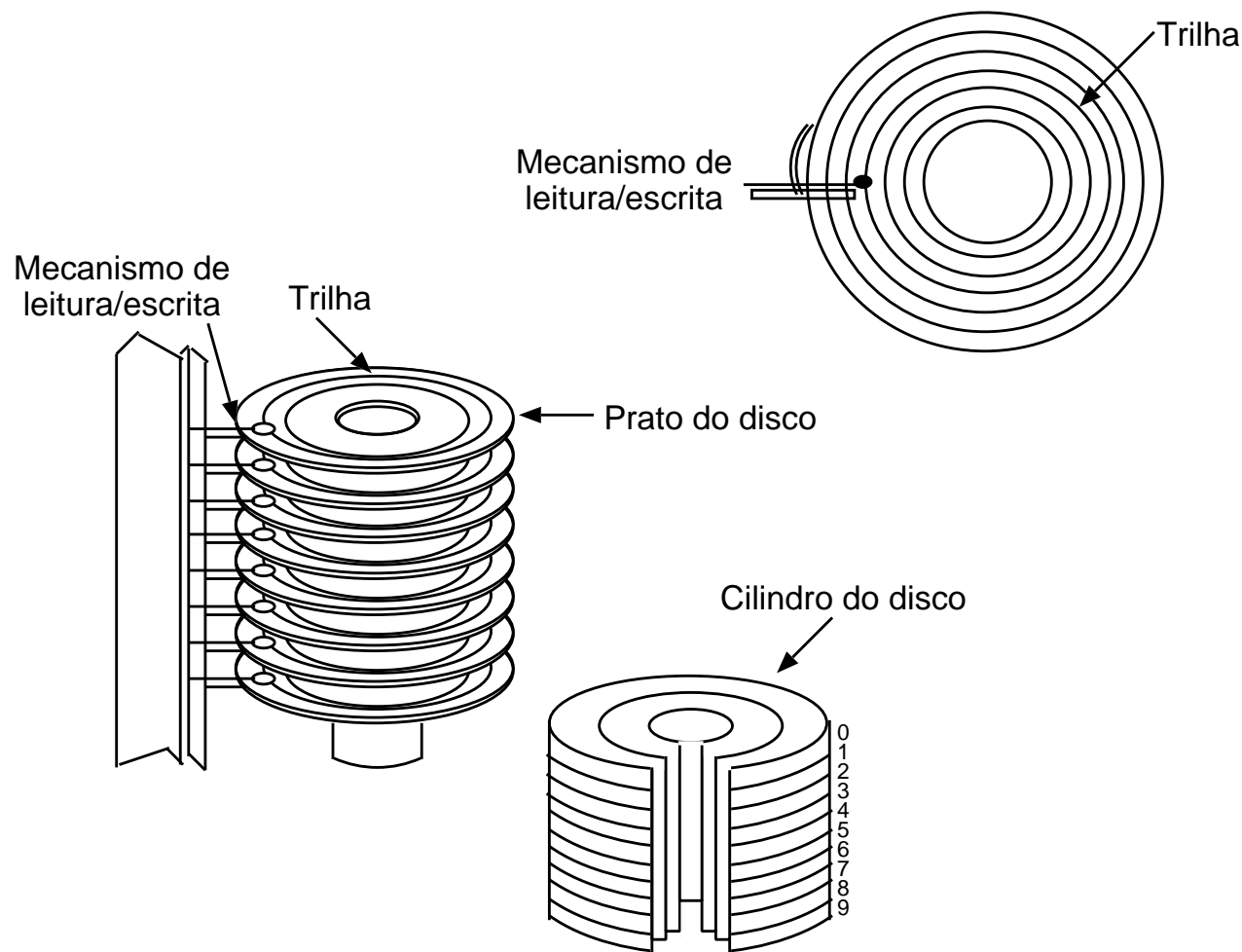
- Disco magnético



(Fig. 10.32 do livro texto)

# Dispositivos externos

- Disco magnético



(Fig. 10.34 do livro texto)

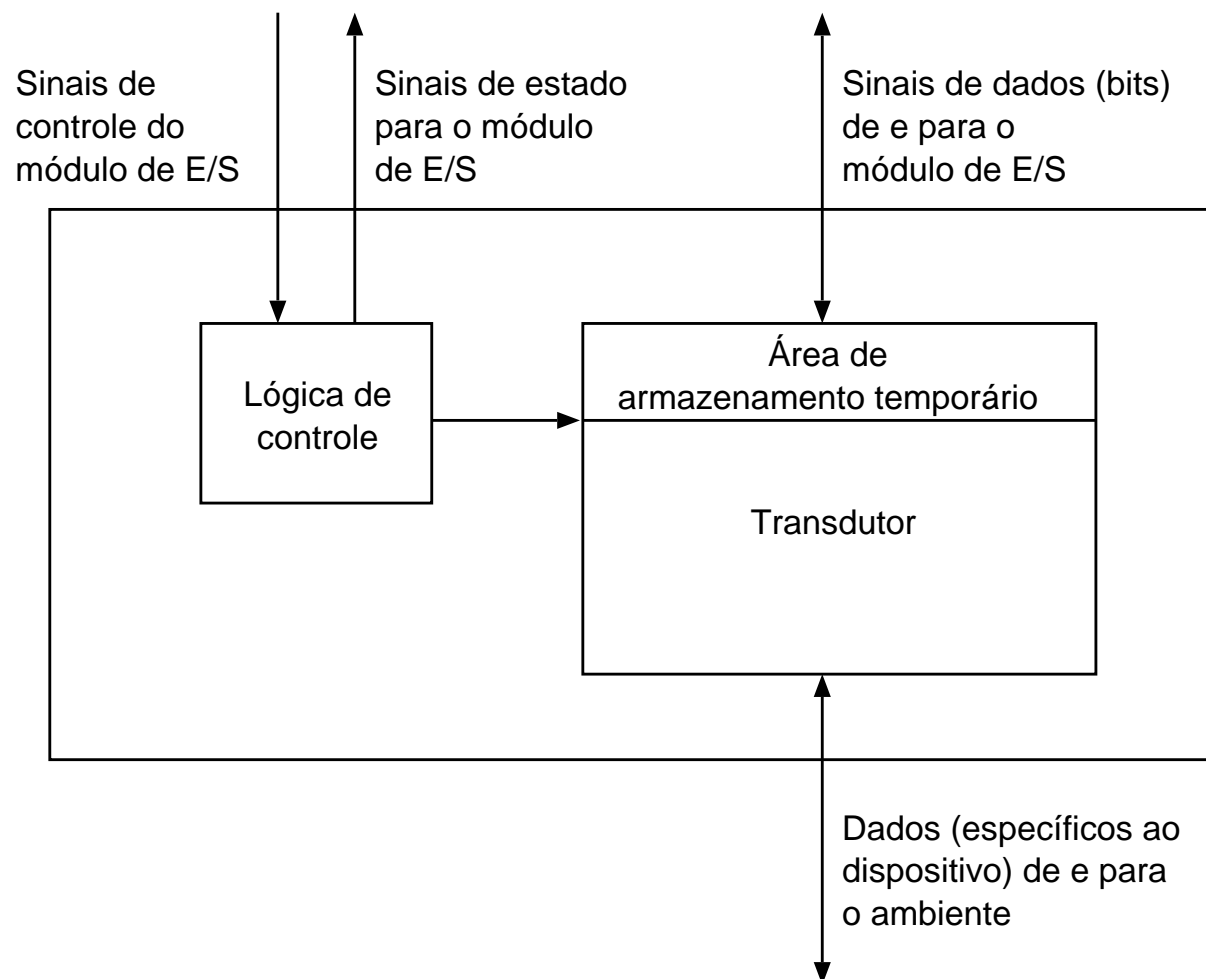
# Interface de E/S

- Existe uma grande variedade de periféricos
  - Entregam e recebem quantidades diferentes de dados
  - Trabalham em velocidades diferentes
  - Utilizam diferentes formatos
- Todos são mais lentos que a UCP e memória principal
- Necessita-se de módulos, controladores ou interfaces de E/S



# Interface de E/S

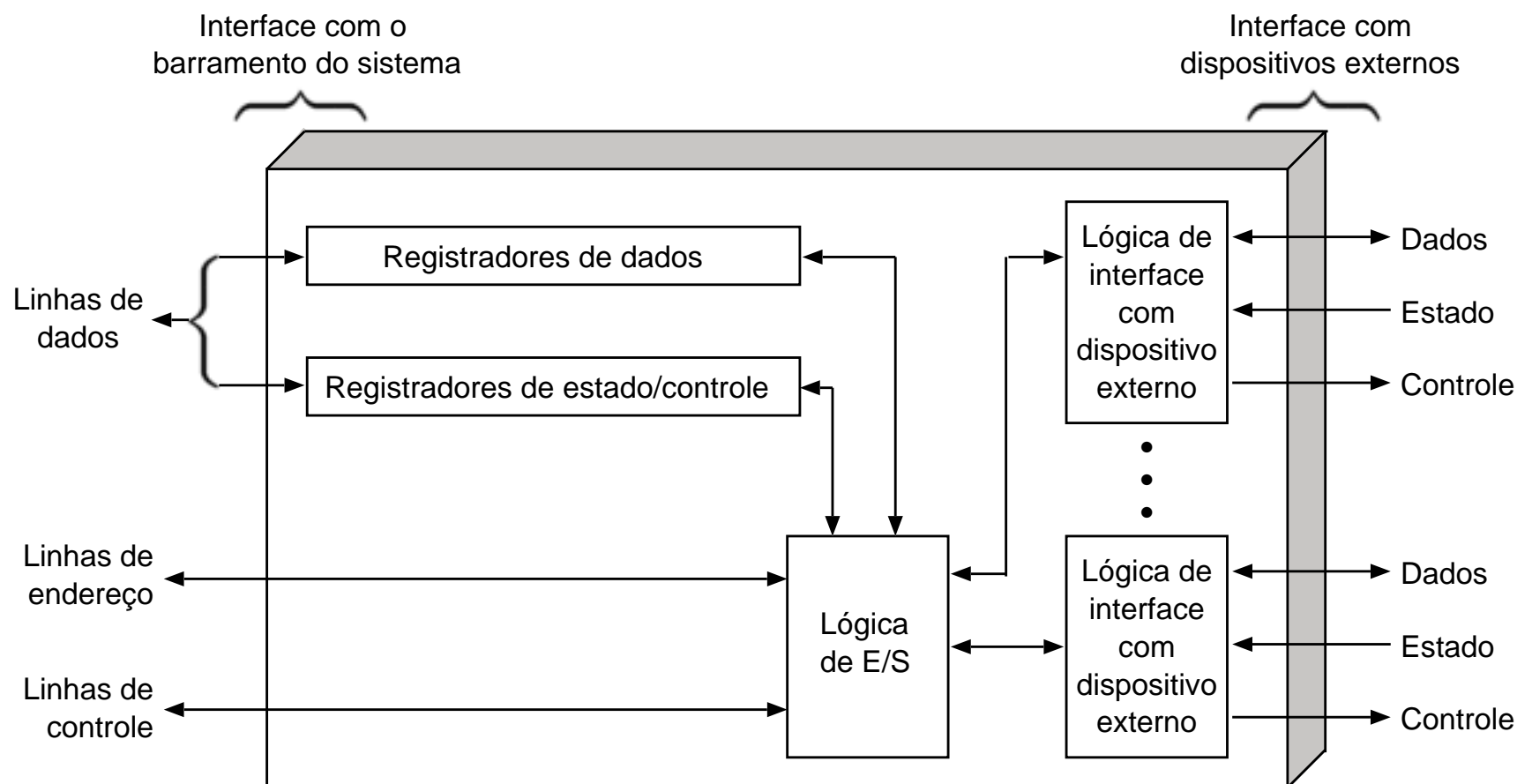
- Modelo geral de um dispositivo externo



(Fig. 6.2 do livro de Arquitetura e Organização de Computadores" , William Stallings)

# Interface de E/S

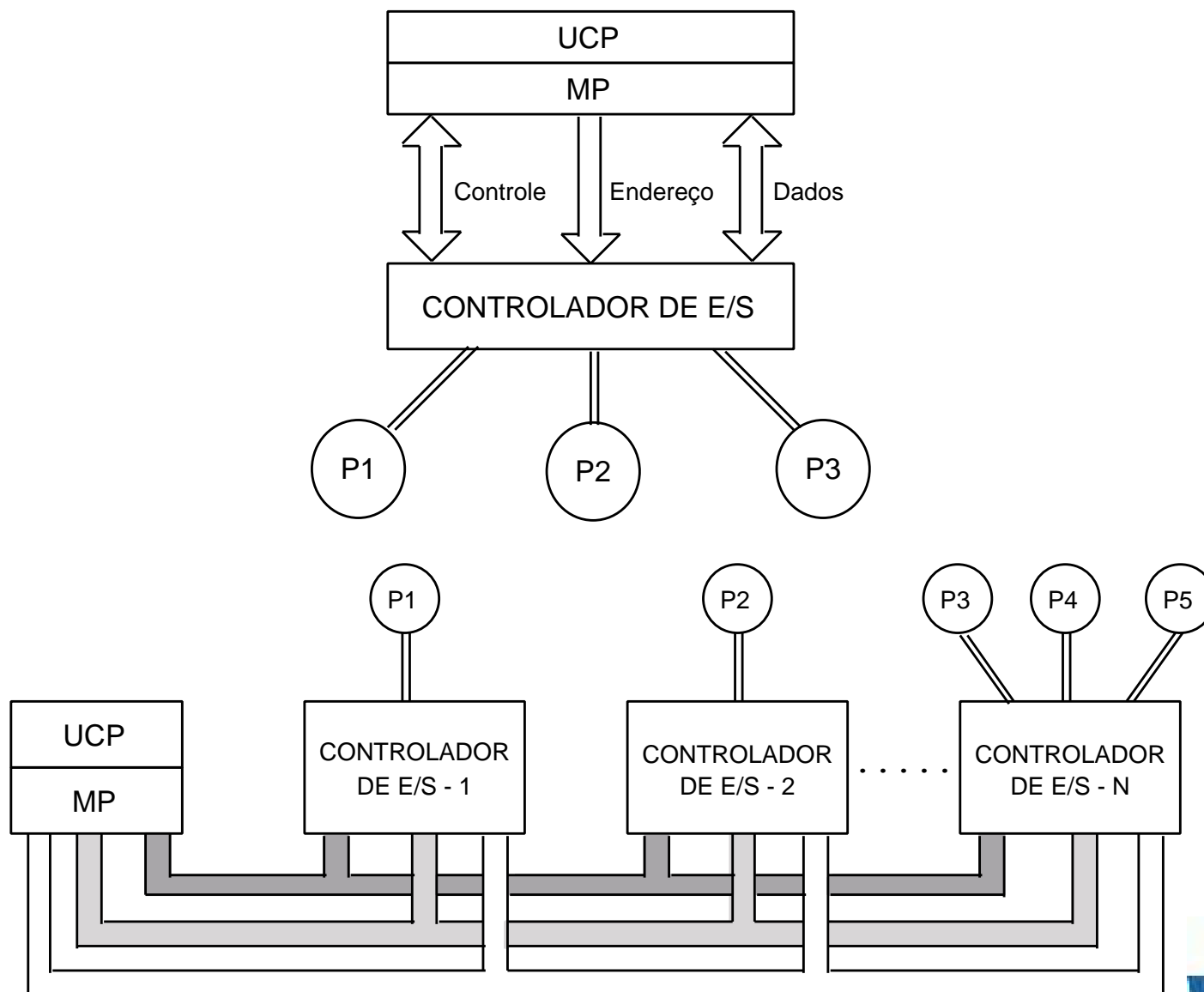
- Diagrama de blocos



(Fig. 6.4 do livro de Arquitetura e Organização de Computadores" , William Stallings)

# Interface de E/S

- Conexão UCP/MP a controladores de E/S



(Fig. 10.8 do livro texto)

# Interface de E/S

- Controle e sincronismo do fluxo de dados entre a UCP/MP e periférico
  - Exemplo:
    1. UCP interroga interface de E/S para verificar estado do dispositivo a ela conectado
    2. A interface de E/S retorna o estado do dispositivo
    3. Se o dispositivo estiver pronto para operar, a UCP requisita a transferência de dados, enviando um comando para a interface de E/S
    4. A interface de E/S recebe ou transmite uma unidade de dados de/para o dispositivo externo
    5. Os dados são transferidos de/para a interface de E/S para/de a UCP
  - Interações entre UCP e interface de E/S são realizadas através de compartilhamento de barramentos

# Interface de E/S

- Comunicação com a UCP
  - Decodificação de comando
  - Os dados são transmitidos através do barramento de dados
  - Informação de estado
  - Reconhecimento de endereço
- Comunicação com os dispositivos
  - Comandos, informação de estado e dados

# Interface de E/S

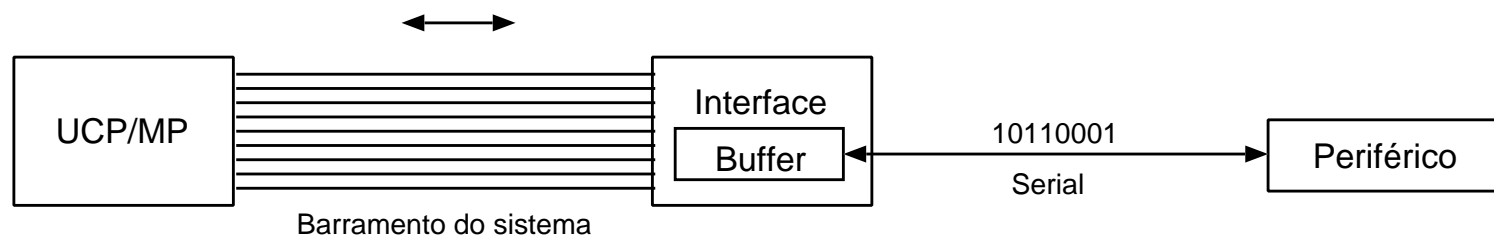
- Armazenamento temporário de dados
  - As taxas de transferência de dados dos dispositivos periféricos apresentam ordens de grandeza menores que as taxas de transferência entre memória principal e UCP
  - Compreendem uma ampla faixa de valores
- Detecção de erros

# Interface de E/S

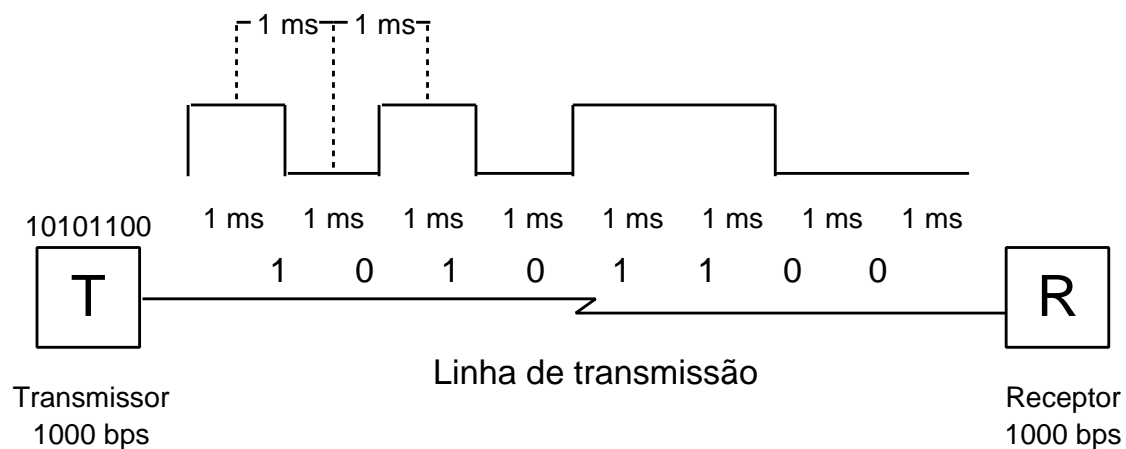
- Transmissão de dados entre periférico e interface de E/S
  - Serial
    - Bit a bit
  - Paralela
    - Grupos de bits de cada vez

# Interface de E/S

- Transmissão serial



(Fig. 10.9 do livro texto)



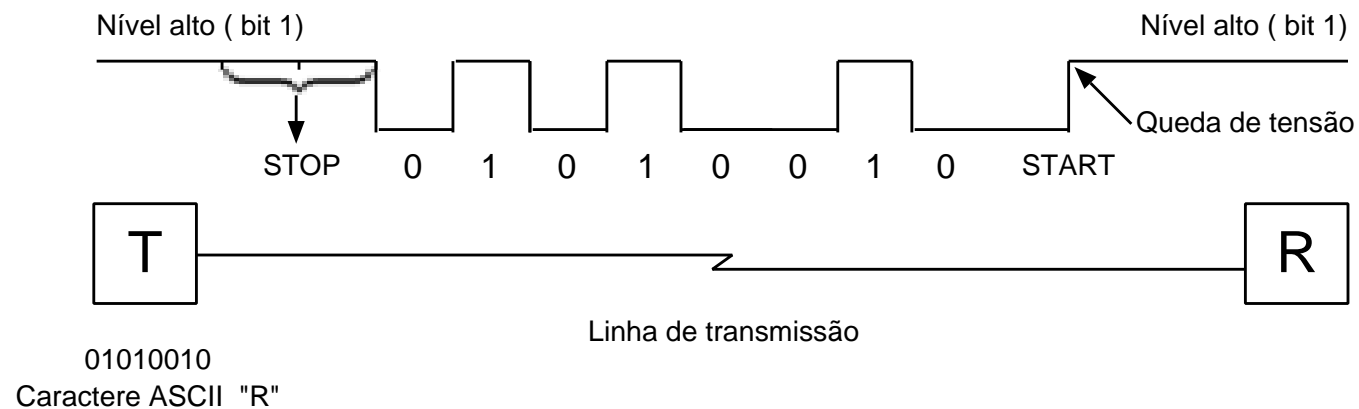
Transmissor: 1 bit =  $1/1000 \text{ s} = 1 \text{ ms}$

(Fig. 10.10 do livro texto)

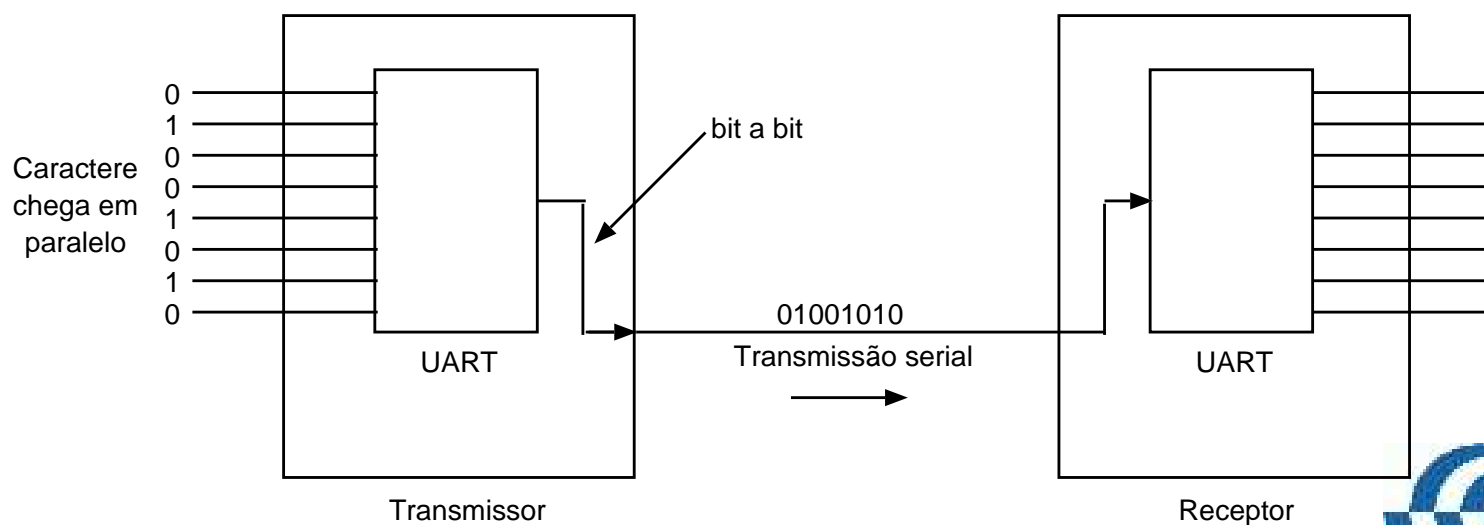


# Interface de E/S

- Transmissão serial assíncrona



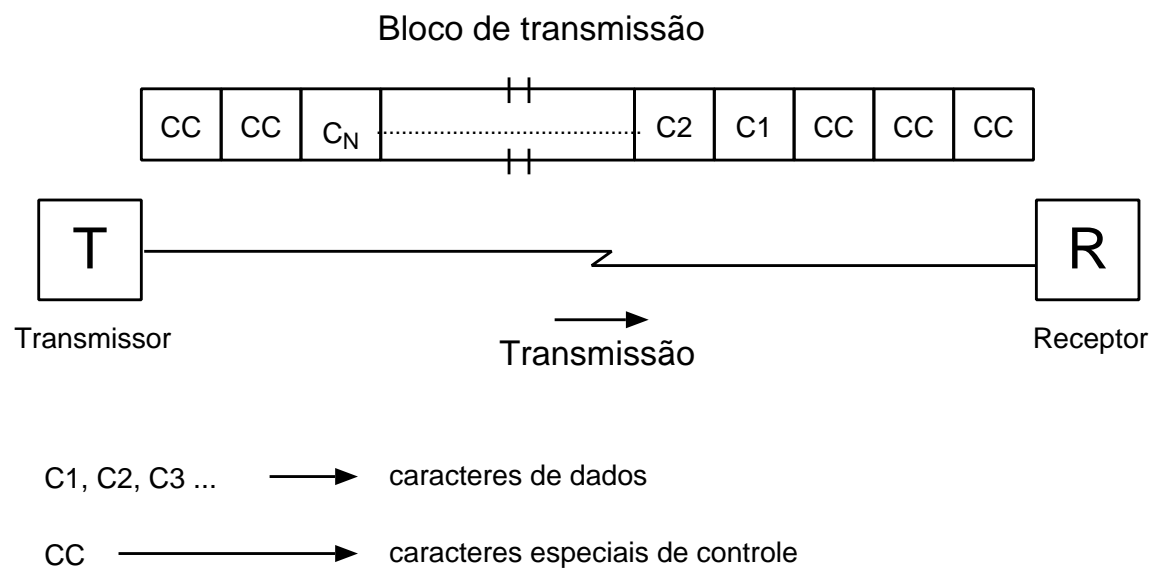
(Fig. 10.11 do livro texto)



(Fig. 10.12 do livro texto)

# Interface de E/S

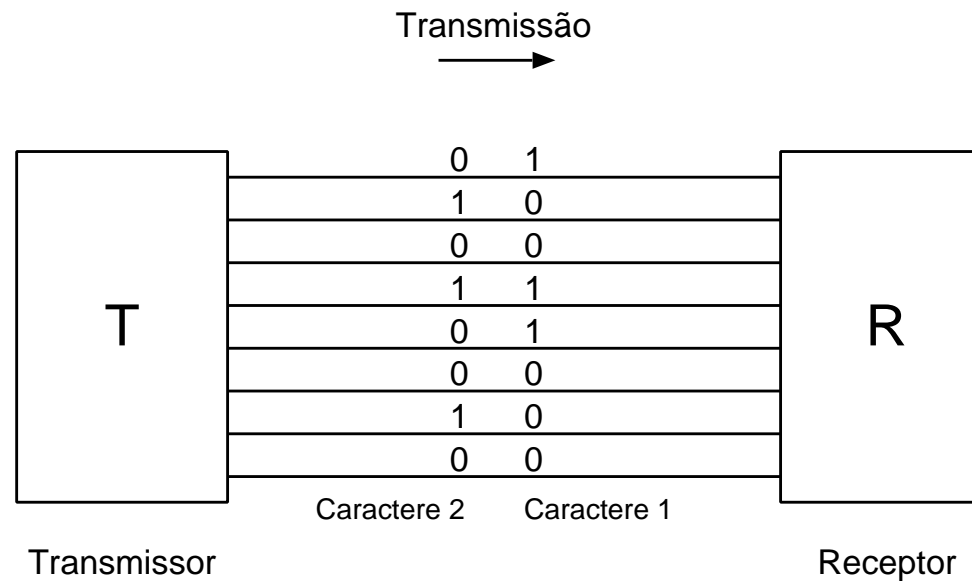
- Transmissão serial síncrona



(Fig. 10.14 do livro texto)

# Interface de E/S

- Transmissão paralela



(Fig. 10.15 do livro texto)

# Operações de E/S

- Endereçamento de dispositivos
  - E/S mapeada na memória
    - Existe um único espaço de endereçamento para posições de memória e dispositivos de E/S
    - Registradores de dados e de estado das interfaces de E/S são vistos pela UCP como endereços de memória
    - São utilizadas as mesmas instruções para acessar memória e dispositivos
  - E/S independente
    - Espaços de endereçamento diferentes para posições de memória e dispositivos de E/S
    - Necessita linhas diferentes para indicar acesso à memória ou dispositivos
    - Comandos especiais de E/S

# Operações de E/S

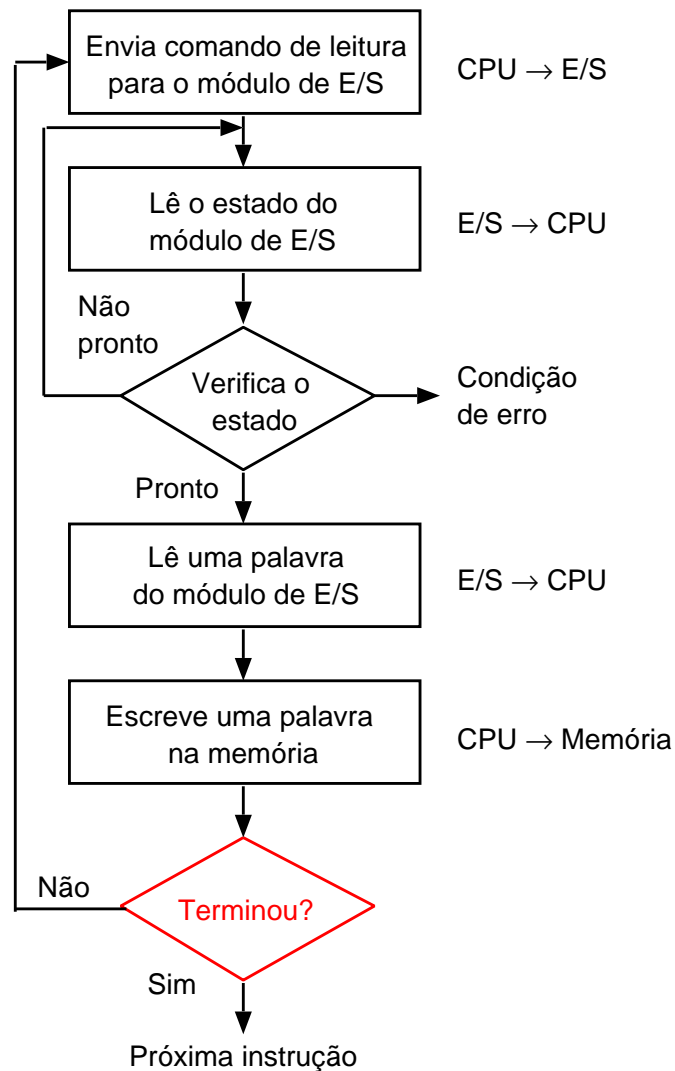
- Técnicas de realização das operações de Entrada e Saída
  - Por programa
  - Interrupção
  - Acesso direto à memória

# Operações de E/S

- Entrada e Saída por programa
  - UCP tem controle direto da operação de E/S
    - Detecção do estado do dispositivo
    - Envio de comandos de leitura ou escrita
    - Transferência de dados
  - UCP tem que monitorar toda a realização da operação
  - Desperdício de tempo da UCP

# Operações de E/S

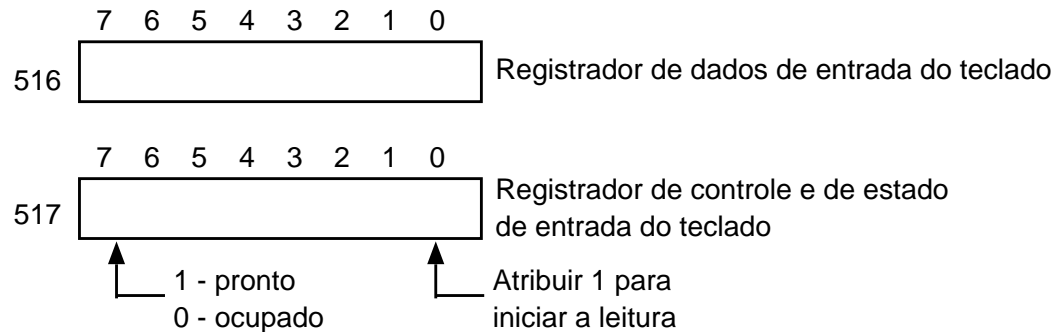
- Entrada e Saída por programa



(Fig. 6.5(a) do livro de Arquitetura e Organização de Computadores", William Stallings)

# Operações de E/S

- Entrada e Saída por programa
  - E/S mapeada na memória



ENDEREÇO	INSTRUÇÃO	OPERANDO	COMENTÁRIO
200	Carregar acumulador	" 1"	
	Armazenar acumulador	517	Iniciar leitura do teclado
202	Carregar acumulador	517	Obter byte de estado
	Desviar se sinal = 0	202	Repetir até que esteja pronto
	Carregar acumulador	516	Carregar byte de dados

(Fig. 6.6(a) do livro de Arquitetura e Organização de Computadores" , William Stallings)

## E/S independente

ENDEREÇO	INSTRUÇÃO	OPERANDO	COMENTÁRIO
200	Iniciar E/S	5	Iniciar leitura do teclado
201	Testar E/S	5	Testar se a operação foi completada
	Desviar se não pronto	201	Repetir até que seja completada
	Leitura	4	Carregar byte de dados

(Fig. 6.6(b) do livro de Arquitetura e Organização de Computadores" , William Stallings)

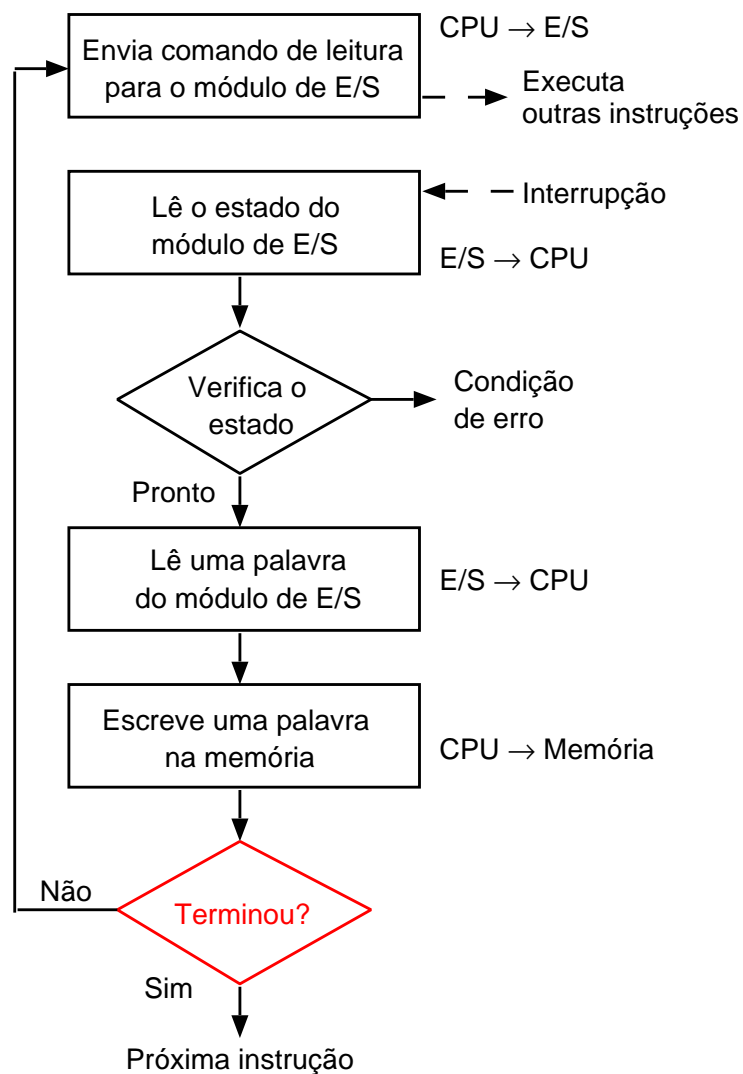


# Operações de E/S

- Interrupção
  - UCP não precisa monitorar dispositivo
  - A interface de E/S interrompe a UCP quando o dispositivo está pronto para realizar a transferência de dados com a UCP

# Operações de E/S

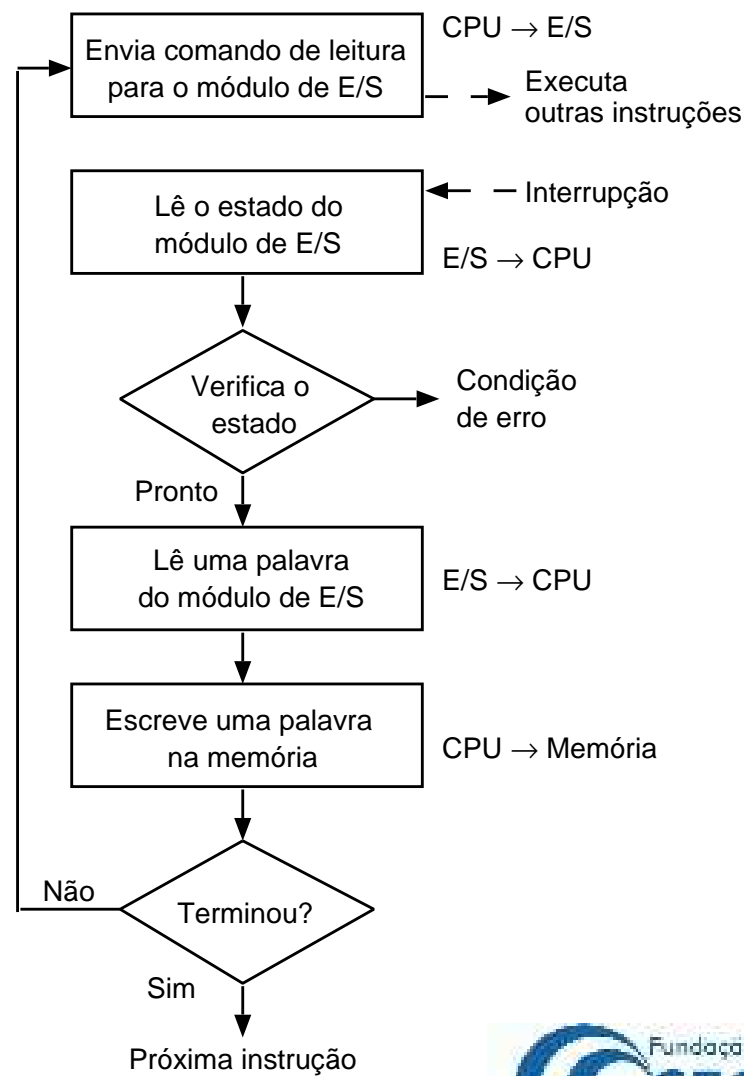
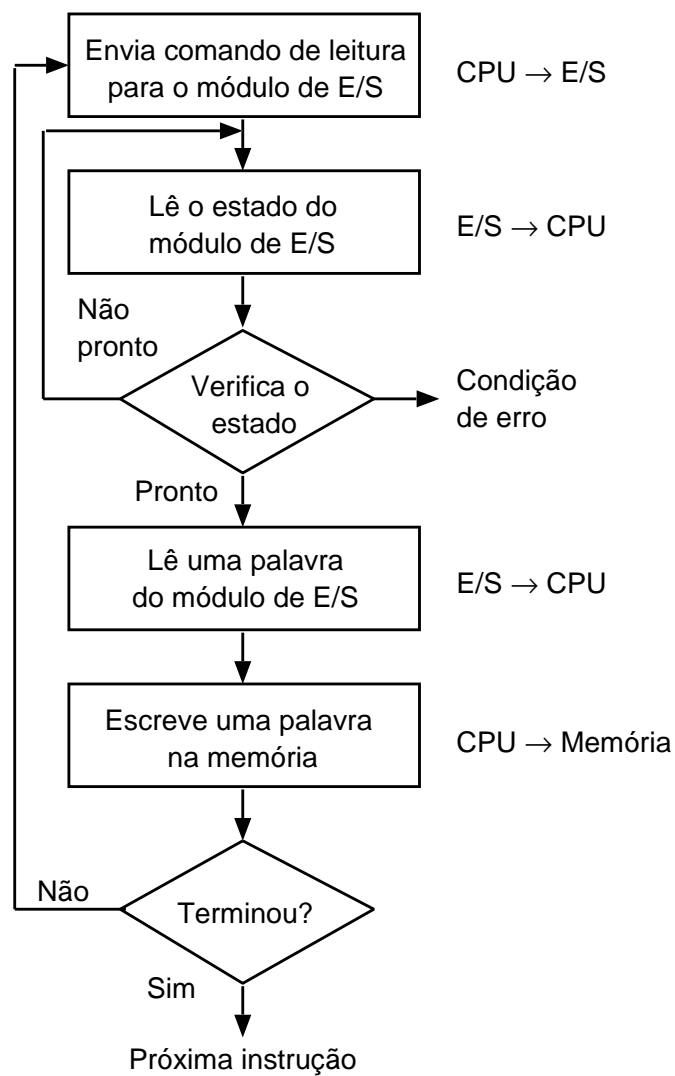
- Interrupção



(Fig. 6.5(b) do livro de Arquitetura e Organização de Computadores" , William Stallings)

# Operações de E/S

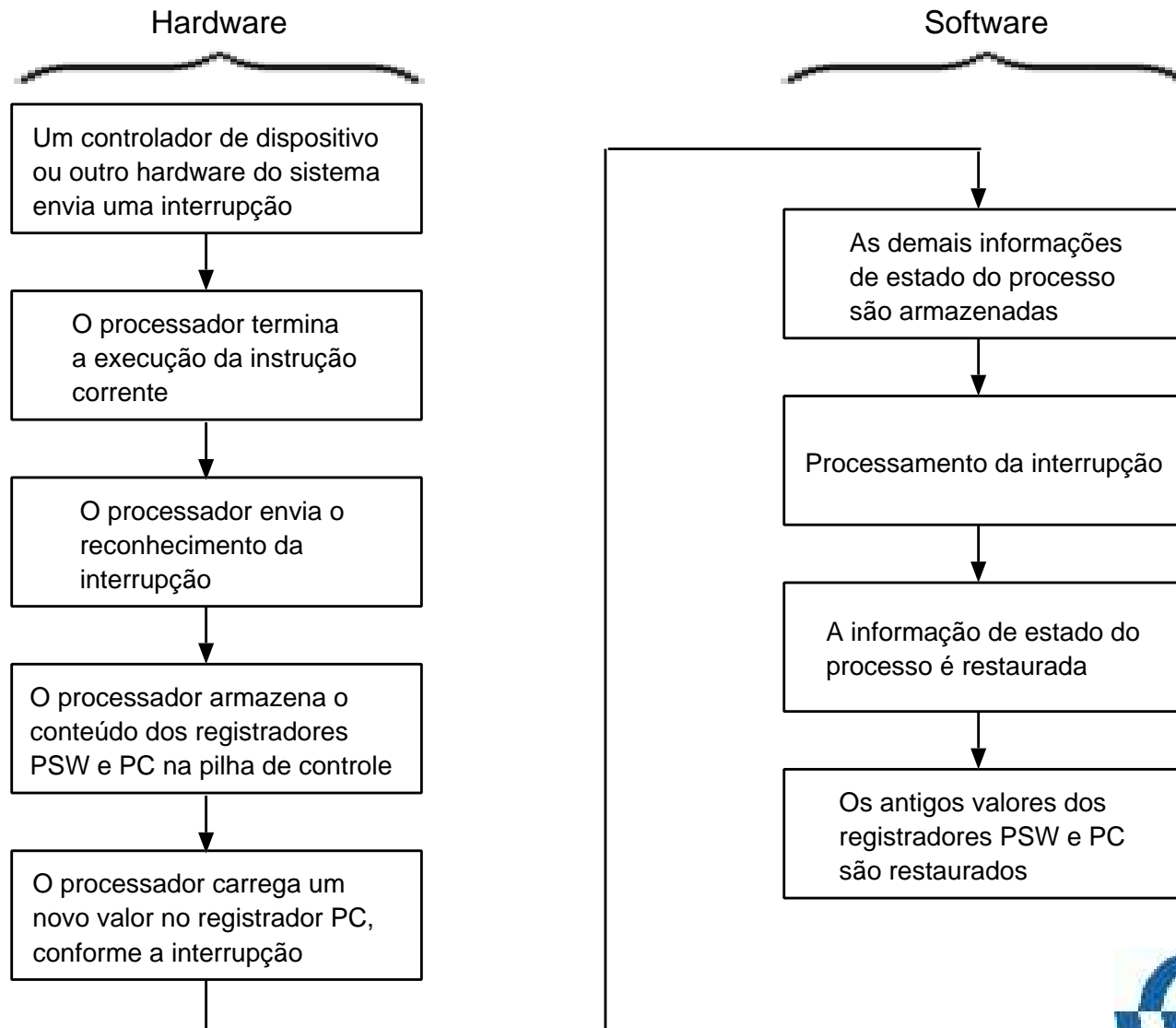
- E/S por programa e Interrupção



(Fig. 6.5(a) e (b) do livro de Arquitetura e Organização de Computadores", William Stallings)

# Operações de E/S

- Interrupção



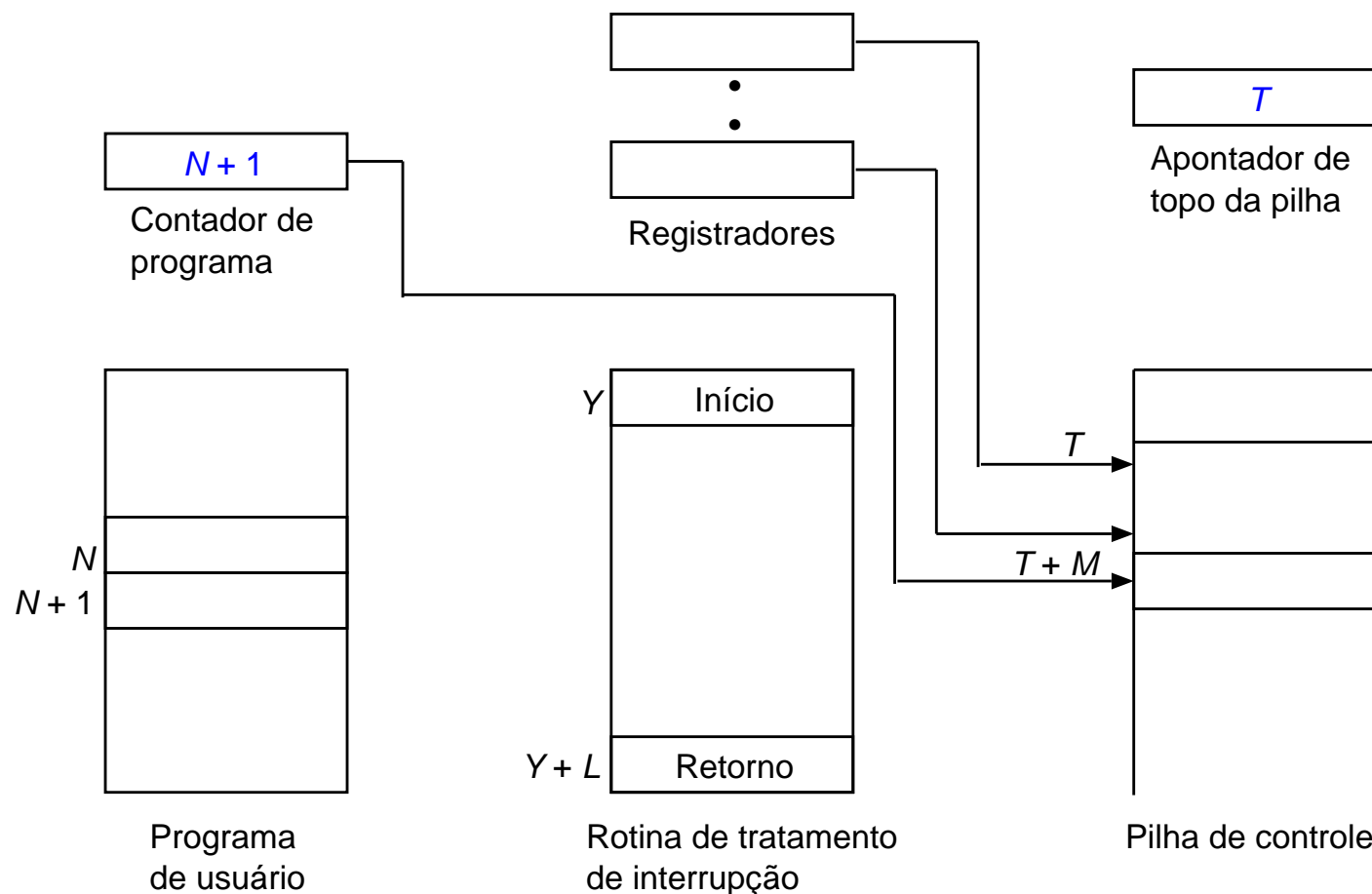
(Fig. 6.7 do livro de Arquitetura e Organização de Computadores", William Stallings)

# Operações de E/S

- Interrupção
  - UCP envia um comando de leitura
  - Executa outro procedimento
  - Verifica se existe uma interrupção ao final de cada ciclo de instrução
  - Se existe algum pedido de interrupção:
    - Salva contexto
    - Processa atendimento à interrupção
      - Obtém e armazena dados

# Operações de E/S

- Interrupção



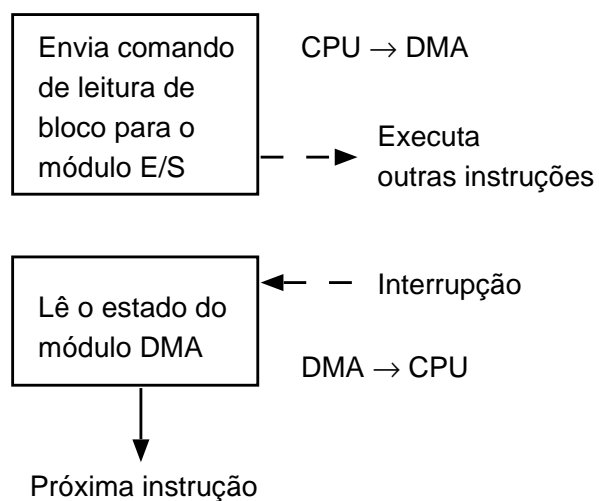
(Fig. 6.8(a) do livro de Arquitetura e Organização de Computadores" , William Stallings)

# Operações de E/S

- Desvantagens de E/S por programa e por interrupção
  - A taxa de transferência de E/S é limitada pela velocidade com que a UCP pode testar e servir um dispositivo
  - A UCP se ocupa de gerenciar a transferência de dados de E/S, tendo de executar várias instruções a cada transferência
- Técnica de acesso direto à memória mais eficiente
  - DMA (Direct Memory Access)

# Operações de E/S

- Acesso direto à memória

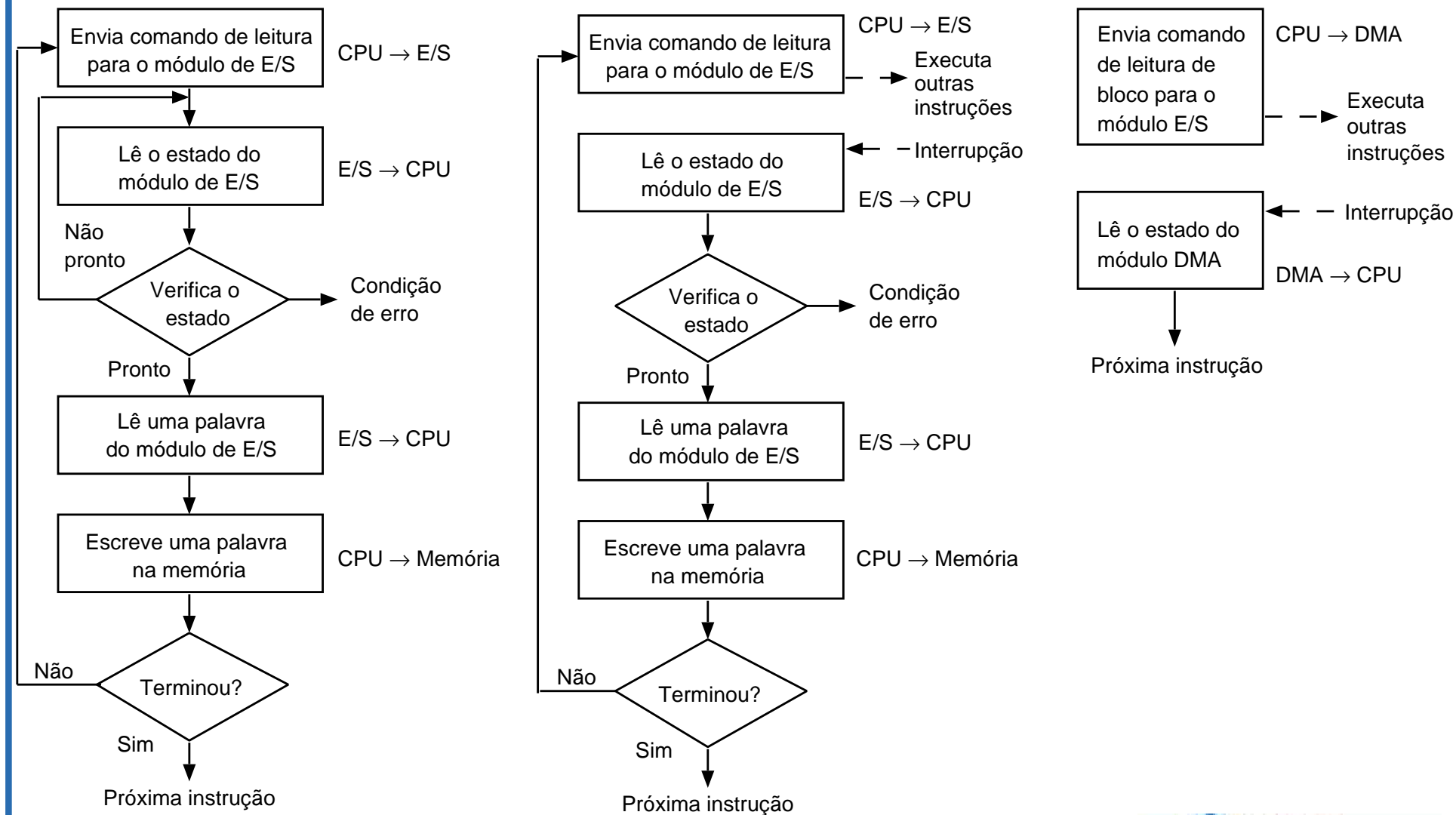


(Fig. 6.5(c) do livro de Arquitetura e Organização de Computadores" , William Stallings)



# Operações de E/S

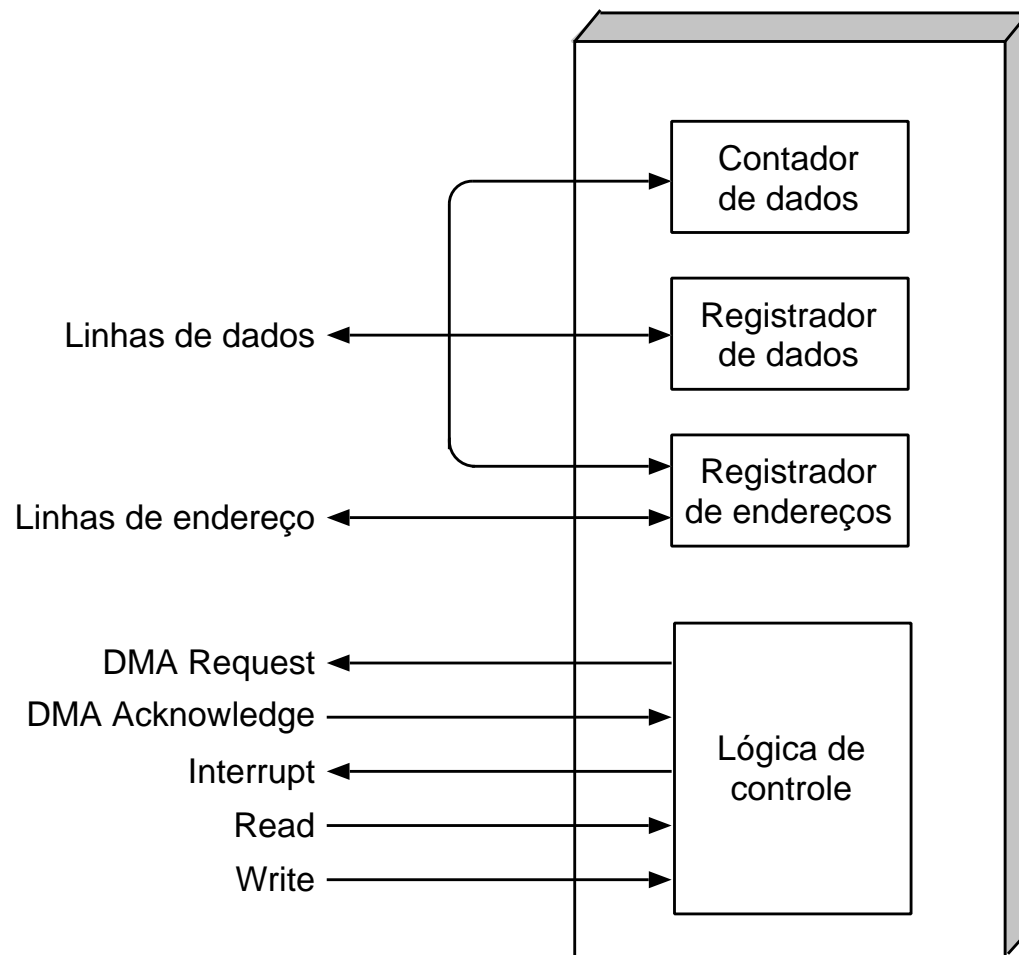
- E/S por programa, Interrupção e Acesso direto à memória



(Fig. 6.5 do livro de Arquitetura e Organização de Computadores", William Stallings)

# Operações de E/S

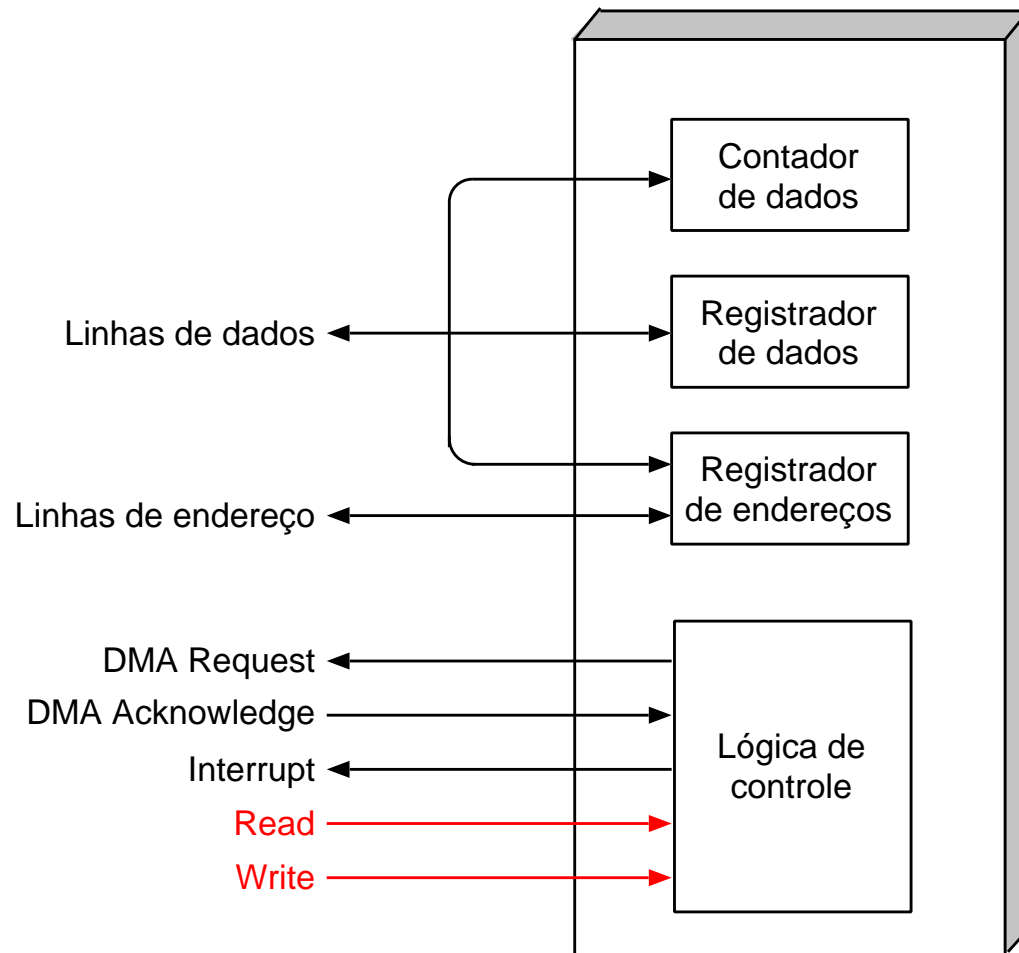
- Diagrama de blocos de um controlador de DMA



(Fig. 6.12 do livro de Arquitetura e Organização de Computadores", William Stallings)

# Operações de E/S

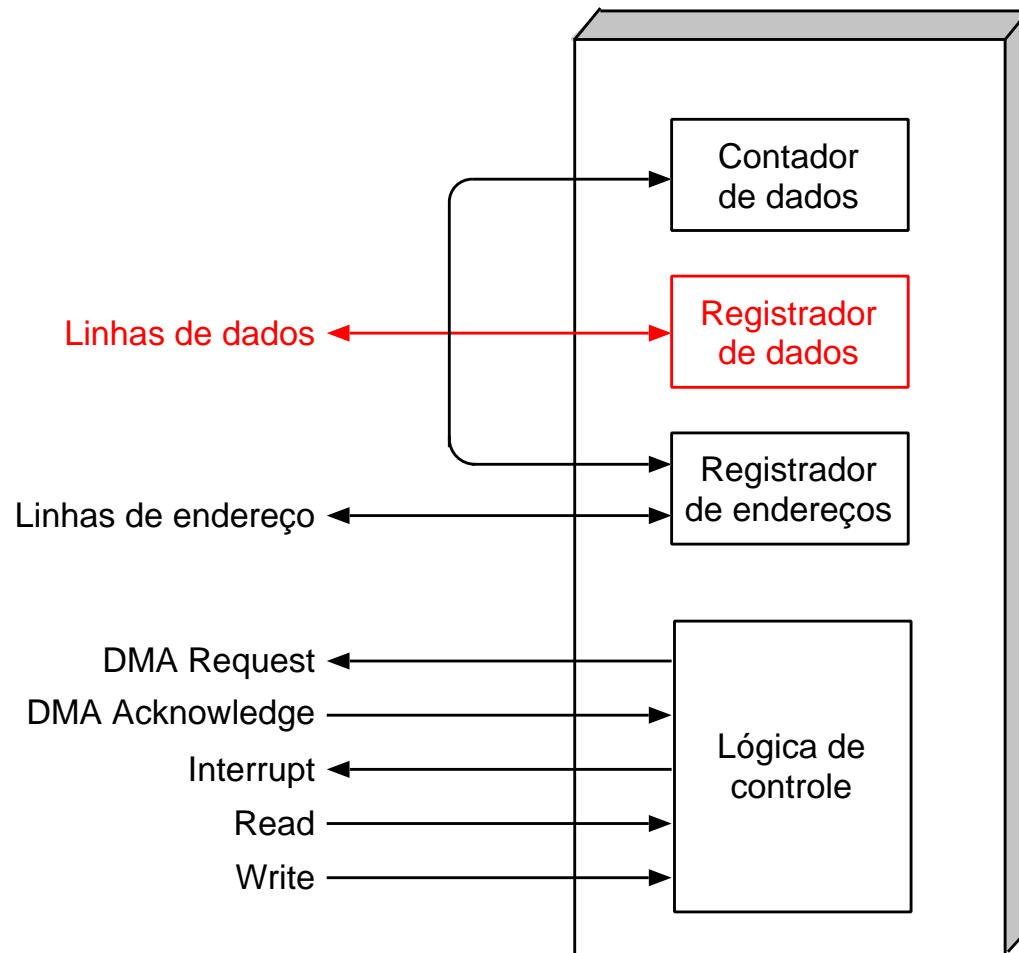
- Informação enviada pela UCP para o controlador de DMA
  - Indicação de operação de leitura ou escrita



(Fig. 6.12 do livro de Arquitetura e Organização de Computadores", William Stallings)

# Operações de E/S

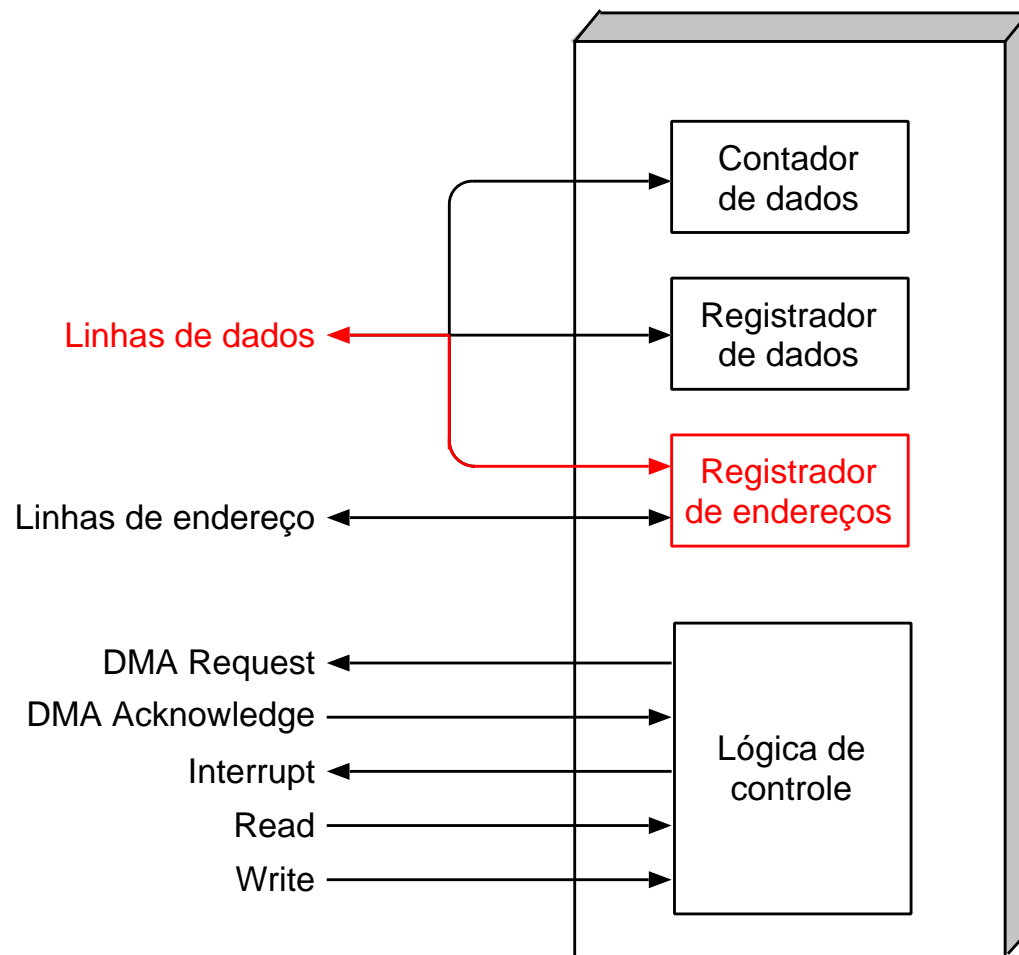
- Informação enviada pela UCP para o controlador de DMA
  - Endereço do dispositivo de E/S enviado pelas linhas de dados



(Fig. 6.12 do livro de Arquitetura e Organização de Computadores", William Stallings)

# Operações de E/S

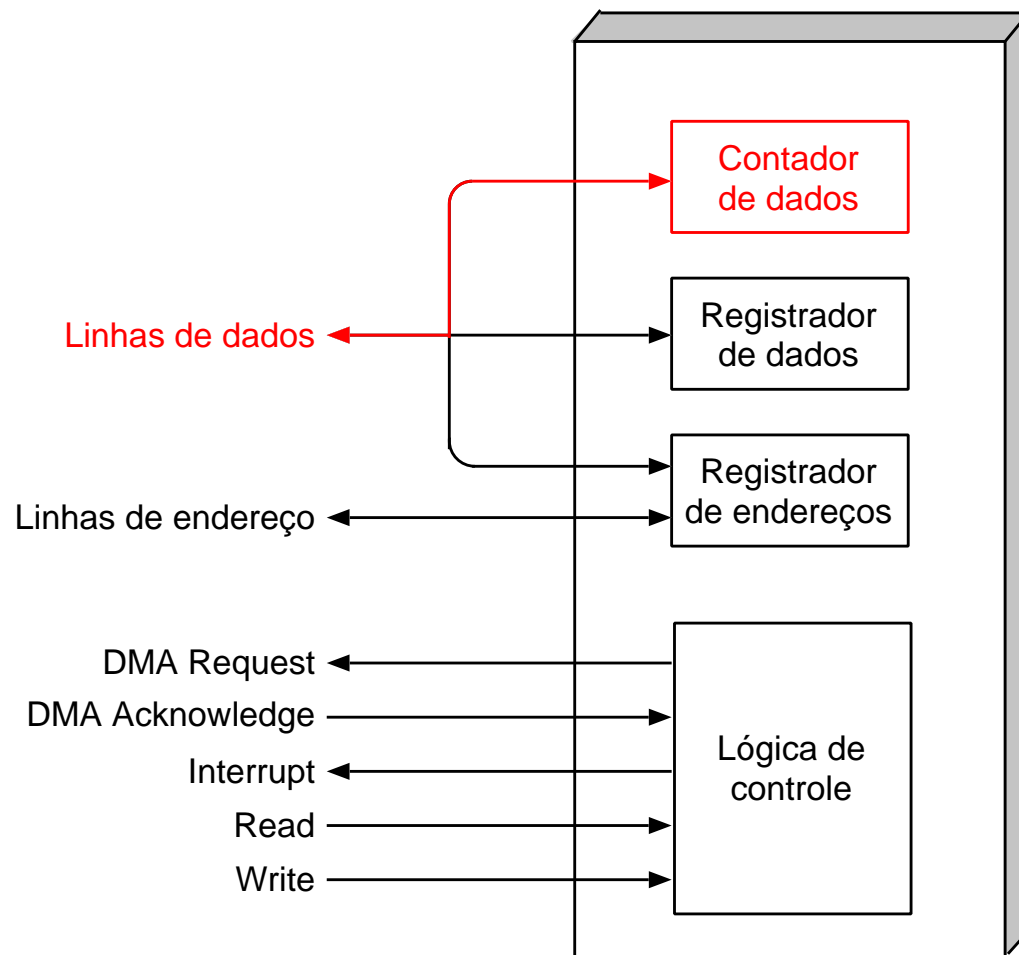
- Informação enviada pela UCP para o controlador de DMA
  - Endereço de memória para início de leitura ou escrita de dados enviado pelas linhas de dados



(Fig. 6.12 do livro de Arquitetura e Organização de Computadores", William Stallings)

# Operações de E/S

- Informação enviada pela UCP para o controlador de DMA
  - Número de palavras a serem lidas ou escritas enviado pelas linhas de dados



(Fig. 6.12 do livro de Arquitetura e Organização de Computadores", William Stallings)

# Operações de E/S

- Operação do controlador de DMA
  - UCP executa outras instruções
  - O controlador de DMA transfere diretamente todo o bloco de dados de ou para a memória
  - O controlador de DMA envia um sinal de interrupção quando a transferência é concluída

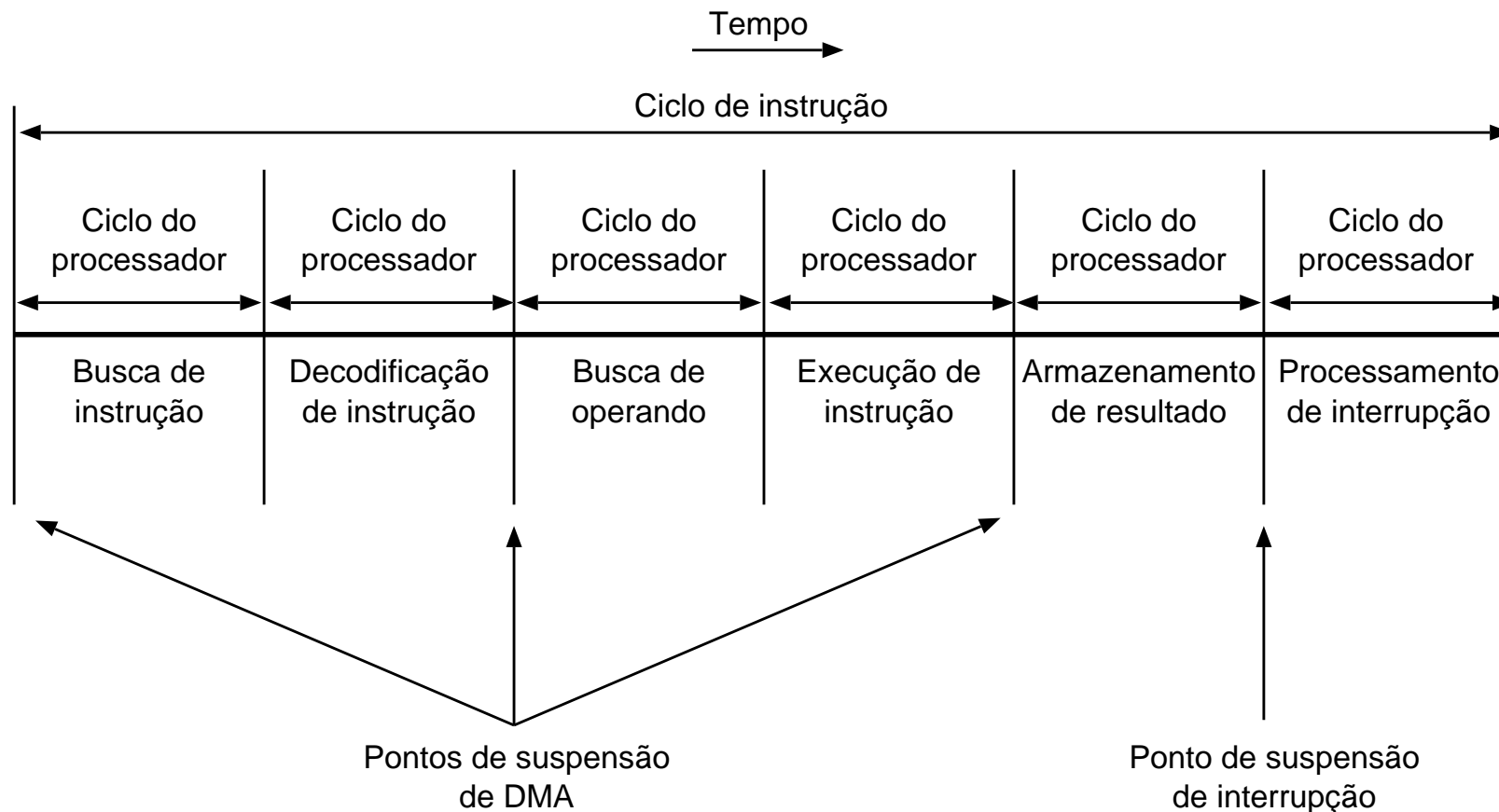
# Operações de E/S

- O controlador de DMA tem que acessar o barramento
- O controlador de DMA efetua a transferência de uma palavra por acesso ao barramento
- A UCP não pode acessar o barramento mas não necessita executar instruções relacionadas à transferência
  - Não realiza troca de contexto
- A UCP é suspensa antes que precise acessar o barramento
  - Antes de ler ou escrever da/na memória
- Torna a execução das instruções um pouco mais lenta mas não tanto quanto nos casos em que a UCP realiza a transferência



# Operações de E/S

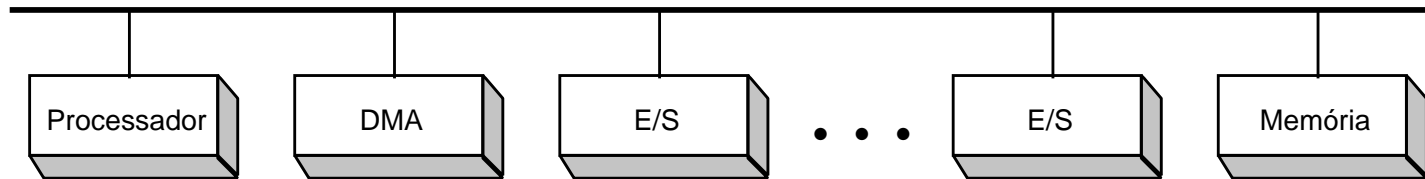
- Pontos de suspensão de DMA e de interrupção



(Fig. 6.13 do livro de Arquitetura e Organização de Computadores", William Stallings)

# Operações de E/S

- Configuração de DMA (1)

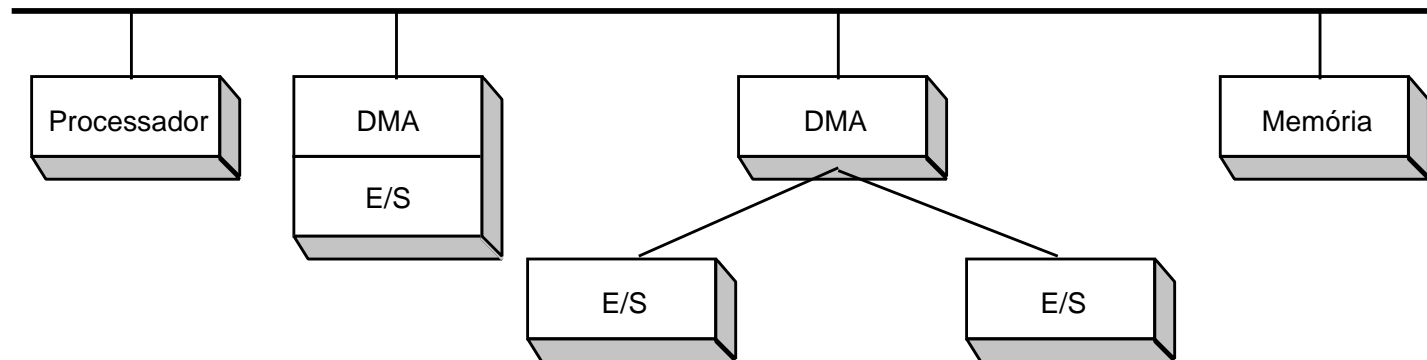


(Fig. 6.14(a) do livro de Arquitetura e Organização de Computadores" , William Stallings)

- Cada transferência utiliza duas vezes o barramento
- A UCP é suspensa duas vezes

# Operações de E/S

- Configuração de DMA (2)

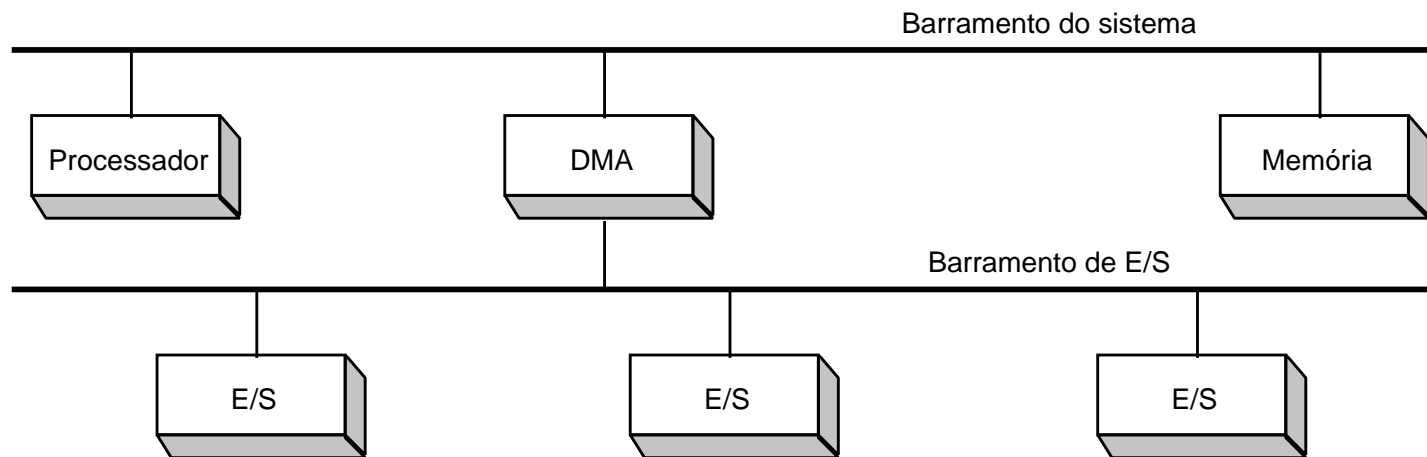


(Fig. 6.14(b) do livro de Arquitetura e Organização de Computadores" , William Stallings)

- Um controlador pode atender mais de um dispositivo
- Cada transferência utiliza uma vez o barramento
- A UCP é suspensa uma vez

# Operações de E/S

- Configuração de DMA (3)



(Fig. 6.14(c) do livro de Arquitetura e Organização de Computadores" , William Stallings)

- Todos os dispositivos são conectados ao controlador através de um único barramento
- Cada transferência utiliza uma vez o barramento
- A UCP é suspensa uma vez

# Exercícios

- Capítulo 10 do livro texto
  - 14, 15, 21, 22, 25, 26, 30, 31 ,32

## Aula 10

### Professores:

Lúcia M. A. Drummond  
Simone de Lima Martins

### Conteúdo:

#### Arquiteturas Avançadas

- Arquiteturas RISC
- Processamento Paralelo

# Arquiteturas RISC

- Reduced Instruction Set Computer se contrapõe à arquiteturas até então predominantes - CISC - Complex Instruction Set Computer

# Arquiteturas RISC

- Desenvolvimento por três caminhos:
  - Projeto da IBM, desenvolvido em meados da década de 1970, sem sucesso comercial. A IBM ganhou mercado por volta de 1990 com os processadores RS/6000 e posteriormente com a família POWER PC, desenvolvida em conjunto com a Motorola e com a Apple.
  - Estudos em Stanford, por John Hennessy, que redundaram nos processadores Mips.
  - Estudos em Berkley, por David Patterson, que redundaram em processadores desenvolvidos pela Sun



# Arquiteturas RISC

- Alguns estudos verificaram:
  - Linguagem de alto nível com comandos poderosos para facilitar a vida dos programadores
  - Gap semântico: separação acentuada entre operações de linguagem de alto nível e em linguagem de máquina
  - Compiladores complexos
- Soluções:
  - Aumentar a quantidade de instruções
  - Incluir mais modos de endereçamento
  - Utilizar mais microprogramação

CISC

# Arquiteturas RISC

- Vários pesquisadores realizaram estudos sobre comportamento do programa. Em 1982, Patterson fez uma pesquisa analisando programas científicos, de emprego geral e de editoração.

Comando	Ocorrência		Peso nas inst. máq.		Peso em ref. à MP	
	Pascal	C	Pascal	C	Pascal	C
Assign	45%	38%	23%	13%	14%	15%
Loop	5%	3%	42%	32%	33%	26%
Call	15%	12%	31%	33%	44%	45%
IF	29%	43%	11%	21%	7%	13%
Goto	—	3%	—	—	—	—
Outros	6%	1%	3%	1%	2%	1%

(Fig. 11.2 do livro texto)

# Arquiteturas RISC

- Outros estudos chegaram a mesma conclusão:
  - Necessidade de aperfeiçoar o Hw para atender à demanda de recursos
  - Não eram necessárias tantas instruções de máquina, se apenas algumas delas eram utilizadas na maioria dos programas
  - Muitas instruções significa muitos bits em cada código de operação - instrução com maior comprimento, mais tempo de interpretação
  - Desenvolvimento de arquiteturas **RISC**

# Arquiteturas RISC

Sistemas	Tipo	Ano	Qtde. inst.	Qtde. reg.	Tamanho inst.
IBM /370-168	CISC	1973	208	16	16-48 bits
Intel 80486	CISC	1989	147	8	1-17 bits
Intel Pentium	CISC	1993	150	8	1-17 bits
Power PC 601	RISC	1993	184	32-I 32-PF	32 bits
Sparc 10	RISC	1987	52	até 528	32 bits
Apha 21064	RISC	1992	125	32-I 32-PF	32 bits

(Fig. 11.1 do livro texto)

# Arquiteturas RISC - Características

- Menor quantidade de instruções e tamanho fixo
- Execução otimizada, o sistema deve produzir resultados com melhor desempenho, mesmo considerando que uma menor quantidade de instruções vá conduzir a programas mais longos
- Facilidade na busca e incremento do CI

# Arquiteturas RISC - Características

- Execução otimizada de chamada de funções
- Utilização de registradores para passagem de parâmetros e recuperação dos dados
- Enquanto em arquiteturas CISC eram necessárias leitura e escrita na memória principal
- RISC: Mais registradores já que ocorre a redução de circuitos para decodificação e execução de instruções

# Arquiteturas RISC - Características

- Menor quantidade de modos de endereçamento
  - Apenas dois tipos de instrução para acesso à memória: LOAD/STORE (utilizando modo direto)
  - Redução de ciclos de relógio para execução das demais instruções

# Arquiteturas RISC - Características

- Modo de execução com *Pipelining*
  - Uso altamente produtivo do pipeline
  - Instruções de formatos simples e únicos tiram maior proveito do pipeline - estágios consomem o mesmo tempo



# Arquiteturas RISC - Características

Característica	Considerações
Menor quantidade de instruções que as máquinas CISC	<ul style="list-style-type: none"> <li>• Simplifica o processamento de cada instrução e torna este item mais eficaz.</li> <li>• Embora o processador RS/600 possua 184 instruções, ainda assim é bem menos que as 303 instruções dos sistemas VAX-11. Além disso, a maioria das instruções é realizada em 1 ciclo de relógio, o que é considerado o objetivo maior dessa arquitetura.</li> </ul>
Execução otimizada de chamada de funções	<ul style="list-style-type: none"> <li>• As máquinas RISC utilizam os registradores da UCP (em maior quantidade que os processadores CISC) para armazenar parâmetros e variáveis em chamadas de rotinas e funções. Os processadores CISC usam mais a memória para a tarefa.</li> </ul>
Menor quantidade de modos de endereçamento	<ul style="list-style-type: none"> <li>• As instruções de processadores RISC são basicamente do tipo Load/Store, desvio e de operações aritméticas e lógicas, reduzindo com isso seu tamanho.</li> <li>• A grande quantidade de modos de endereçamento das instruções de processadores CISC aumenta o tempo de execução das mesmas.</li> </ul>
Utilização em larga escala de <i>pipelining</i>	<ul style="list-style-type: none"> <li>• Um dos fatores principais que permite aos processadores RISC atingir seu objetivo de completar a execução de uma instrução pelo menos a cada ciclo de relógio é o emprego de <i>pipelining</i> em larga escala.</li> </ul>

(Fig. 11.3 do livro texto)

# Arquiteturas RISC - Medidas de Desempenho

- MIPS - milhões de instruções por segundo. Não é uma boa medida para comparar RISC com CISC. RISC mais instruções simples.
- MFLOPS - milhões de operações de ponto-flutuante por segundo. Unidade mais apropriada para medir a velocidade com cálculos matemáticos: programas científicos, cálculos meteorológicos. Os programas usados para teste são escritos em FORTRAN.
- SPECmark - (system performance evaluation committee) - 10 programas - 6 com cálculos matemáticos e 4 com operações de inteiros e caracteres (escritos em C e Fortran). Medidas em SPECmark, composição dos resultados.

# Arquiteturas RISC - Observações RISC x CISC

- CISC:
  - Menos instruções produz código-objeto menor?  
Não, necessariamente, as instruções podem ser maiores
  - Com menos instruções o programa executa mais rapidamente?  
Não, necessariamente, o tempo de execução de cada instrução pode ser grande

# Arquiteturas RISC - Observações RISC x CISC

- RISC:
  - Instruções possuem código de operação com menor quantidade de bits - tempo de decodificação menor
  - Instruções executadas diretamente pelo hardware e não por um microprograma - execução mais rápida

# Processamento Paralelo

Computador: máquina seqüencial

- Algoritmo: seqüência de instruções
- Processadores: executam instruções seqüencialmente
- Instruções: executada por seqüência de operações (busca, execução, armazenamento)

**Visão não totalmente verdadeira!**

# Processamento Paralelo

- Nível de microoperações: vários sinais de controle gerados ao mesmo tempo
- Técnica de pipeline: sobreposição de etapas de execução
- Máquinas superescalares: diversas unidades de execução em um mesmo processador, que podem executar várias instruções de um mesmo programa em paralelo

# Processamento Paralelo

Outras oportunidades de exploração de paralelismo para melhorar o desempenho:

- Multiprocessadores simétricos: múltiplos processadores compartilhando memória
- Clusters: diversos computadores independentes, conectados entre si, organizados de forma cooperativa
- Máquinas de acesso não uniforme à memória (NUMA): multiprocessador no qual o tempo gasto para acesso à memória varia de acordo com a posição da palavra na memória
- Máquinas vetoriais: otimizam a ULA para processamento de vetores de ponto-flutuante

# Processamento Paralelo

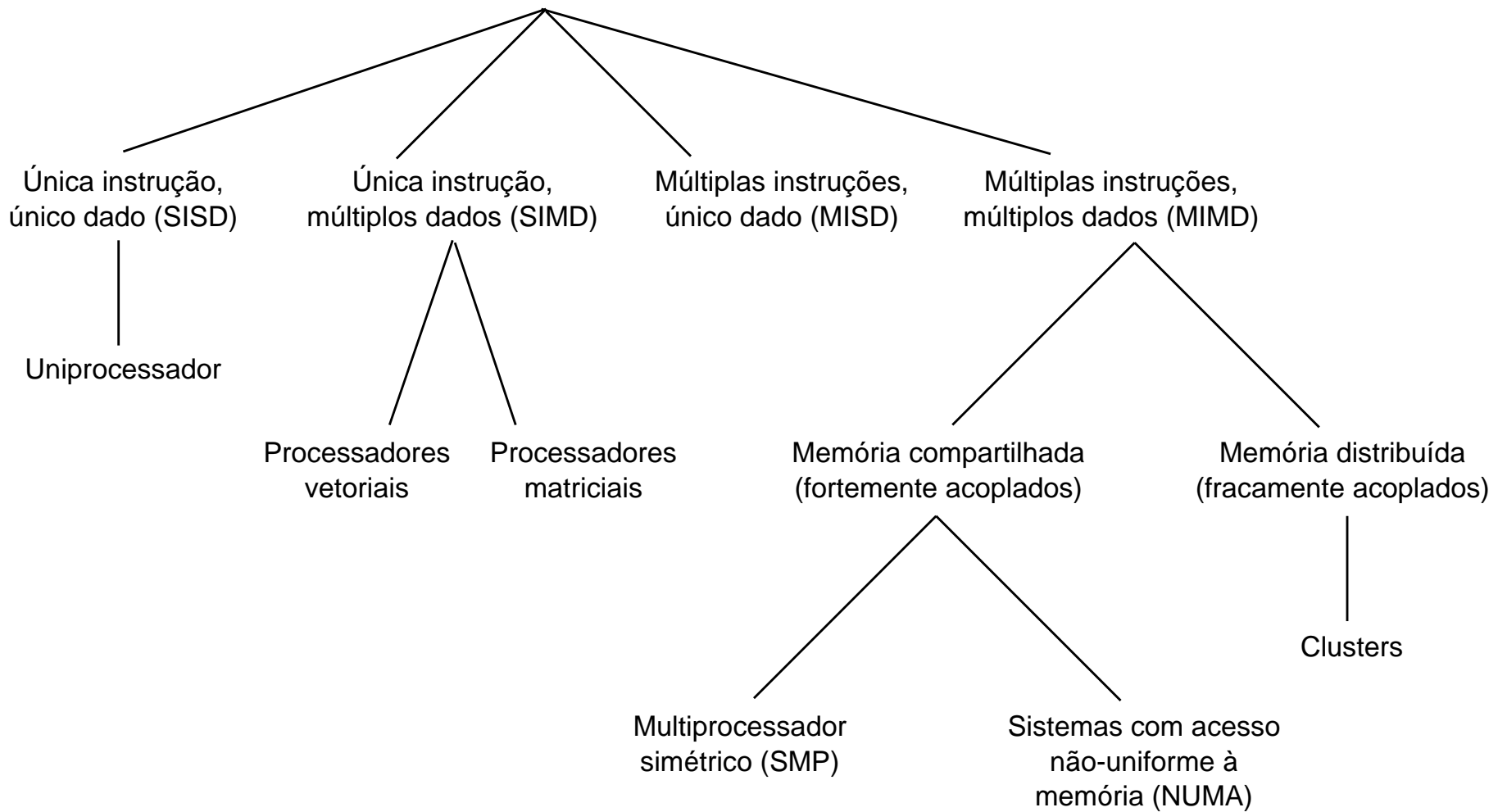
## Taxonomia de Flynn (1972)

- SISD - único processador executa uma única seqüência de instrução sobre dados armazenados em uma única memória.
- SIMD - vários elementos de processamento. Cada um tem uma memória de dados. Cada instrução é executada sobre um conjunto de dados diferente. Processadores vetoriais e matriciais.
- MISD - seqüência de dados é transmitida para um conjunto de processadores, cada um dos quais executa uma seqüência de instruções diferente. Nunca foi implementado.
- MIMD - conjunto de processadores executa simultaneamente seqüências diferentes de instruções. SMPs, clusters, sistemas NUMA.



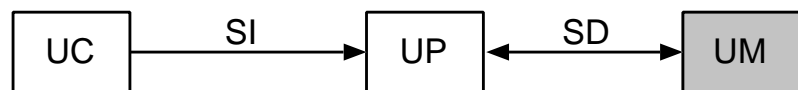
# Processamento Paralelo

## Organizações de processadores

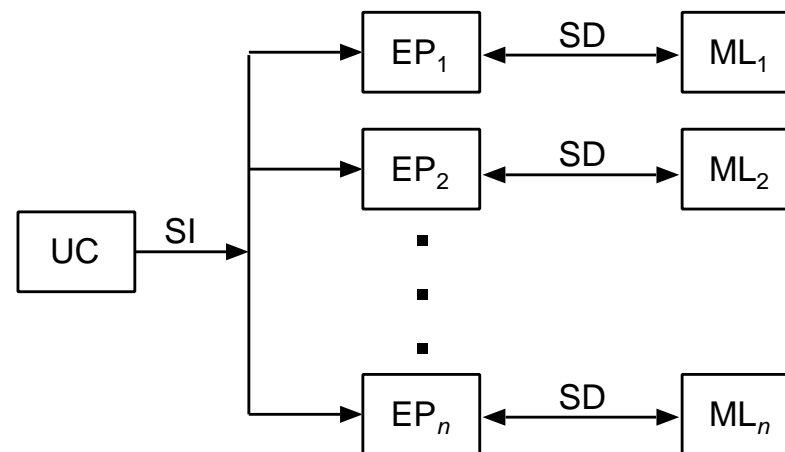


(Fig. 16.1 do livro de Arquitetura e Organização de Computadores", William Stalling)

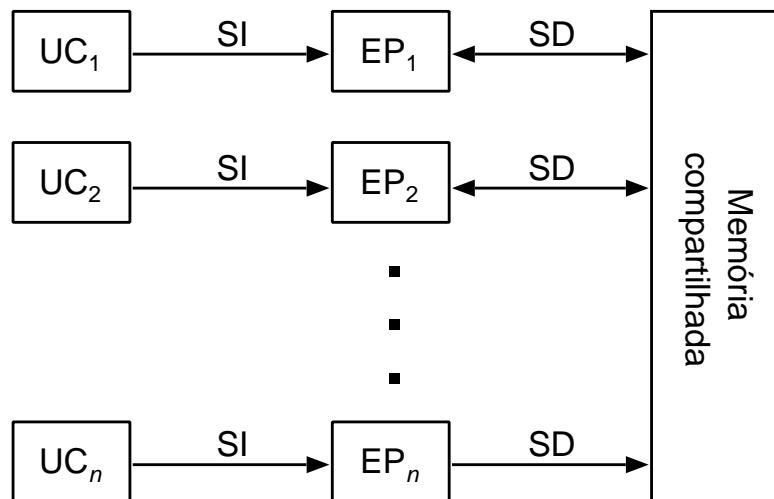
# Processamento Paralelo



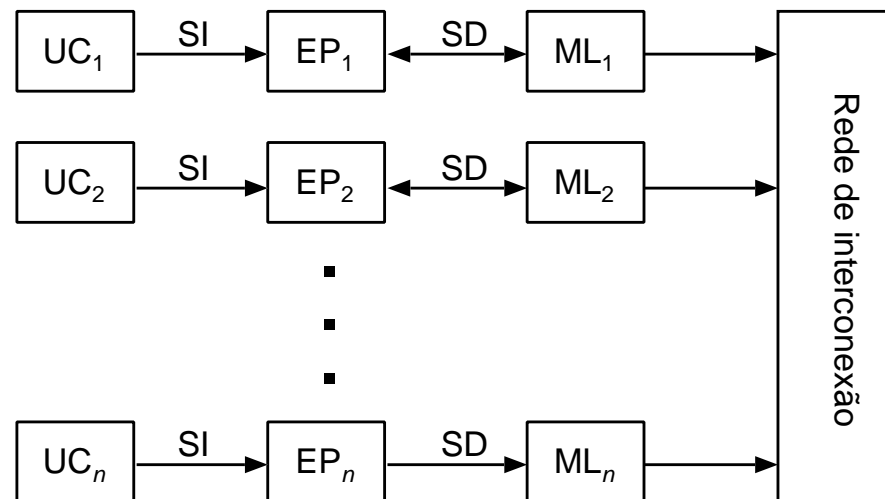
(a) SISD



(b) SIMD (com memória distribuída)



(c) MIMD (com memória compartilhada)



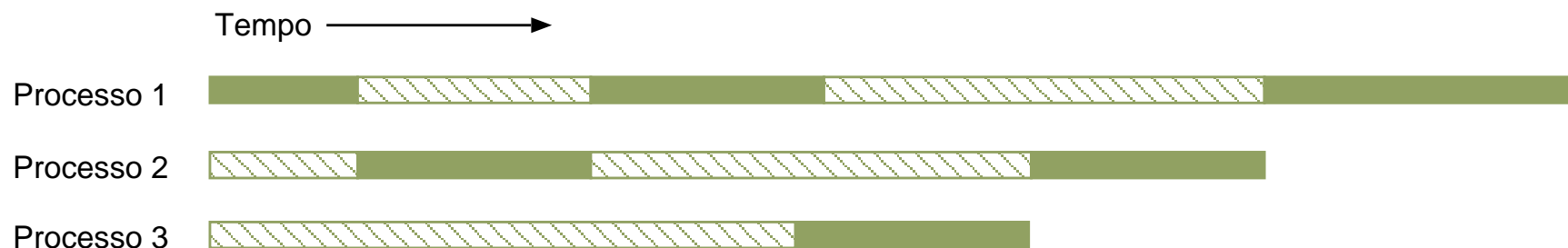
(d) MIMD (com memória distribuída)

# Processamento Paralelo - Multiprocessadores Simétricos

- Características:
  - Existem 2 ou mais processadores similares
  - Processadores compartilham memória
  - Compartilham acesso aos dispositivos de E/S
  - Todos os processadores podem desempenhar a mesma função
  - Sistema controlado por SO integrado
  - Exemplo: IBM S/390, até 10 processadores.

# Processamento Paralelo - Multiprocessadores Simétricos


- Vantagens em relação a uniprocessadores:
  - Desempenho: se o trabalho efetuado pelo computador pode ser organizado de forma que algumas porções possam ser feitas em paralelo, então um sistema com múltiplos processadores resulta em maior desempenho.



(a) Tempo compartilhado (multiprogramação)



(b) Tempo compartilhado e sobreposição (multiprocessamento)

 Bloqueado       Em execução

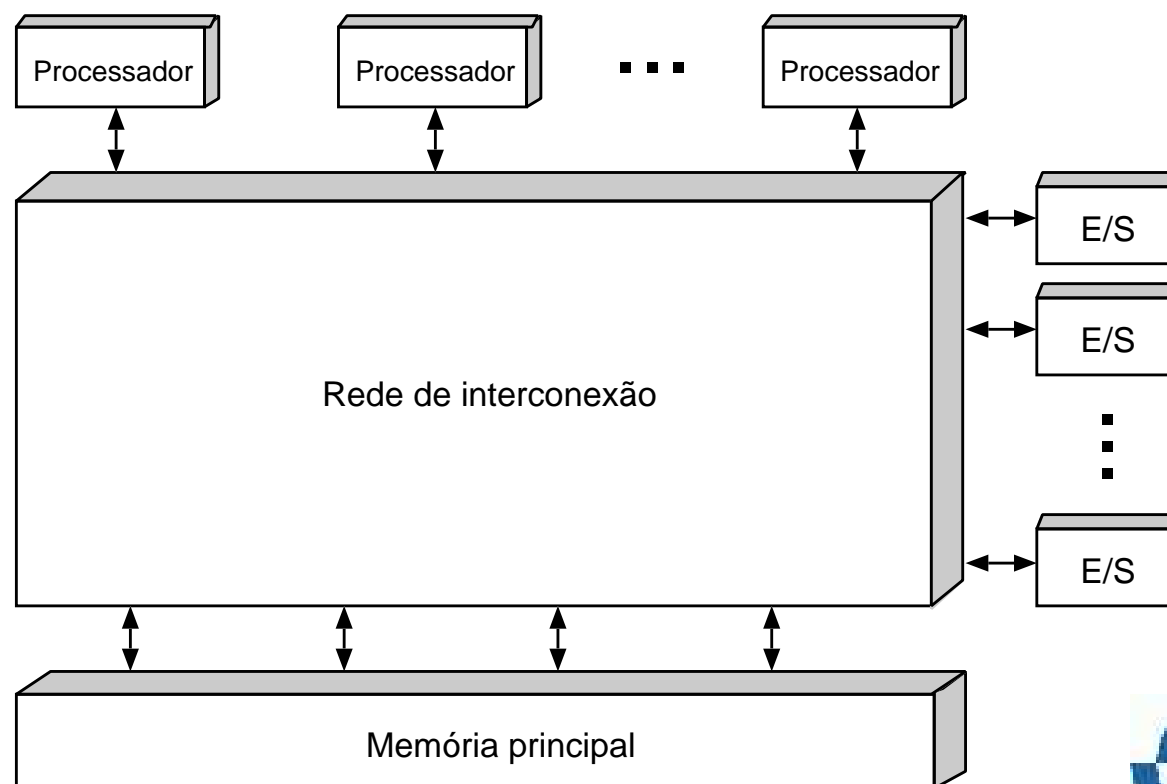
(Fig. 16.3 do livro de Arquitetura e Organização de Computadores" , William Stalling)

# Processamento Paralelo - Multiprocessadores Simétricos

- Vantagens em relação a uniprocessadores:
  - Disponibilidade: a falha de um processador não causa a parada do sistema. O sistema pode continuar a funcionar com desempenho reduzido.
  - Crescimento incremental: para aumentar o desempenho pode se adicionar processador
  - Escalabilidade: fabricantes podem oferecer uma larga faixa de produtos com características de desempenho e custo diferentes, com base no número de processadores

# Processamento Paralelo - Multiprocessadores Simétricos

- Organização:
  - Dois ou mais processadores (ULA, UC, registradores, cache)
  - Memória e dispositivos de E/S compartilhados
  - Processadores podem se comunicar por meio da memória (mensagens e informações armazenadas em áreas comuns)



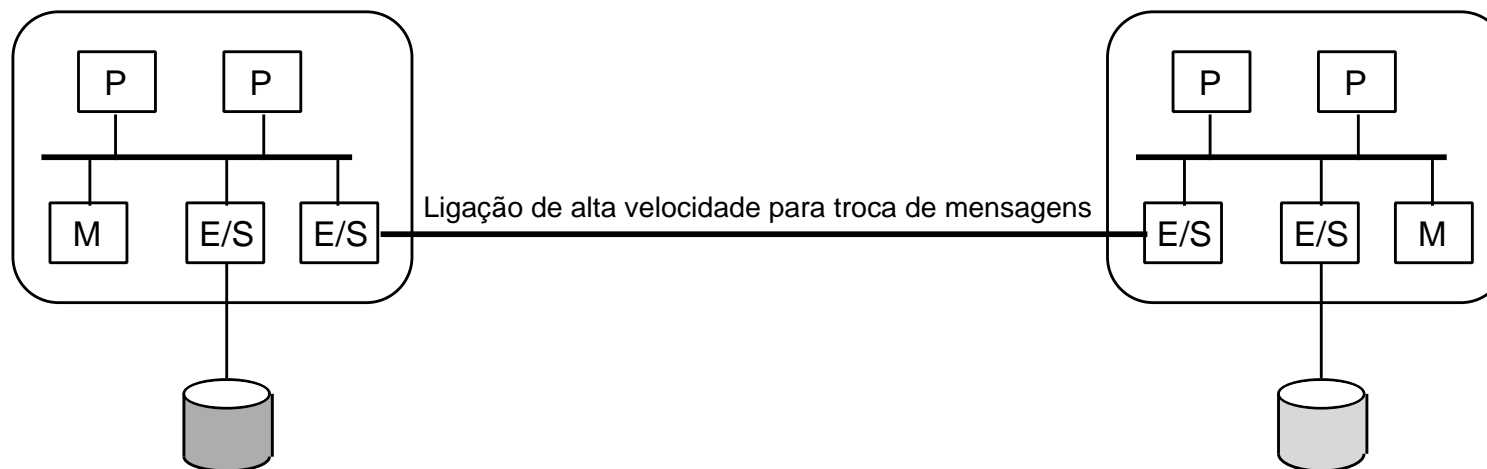
(Fig. 16.4 do livro de Arquitetura e Organização de Computadores", William Stalling)

# Processamento Paralelo - Clusters

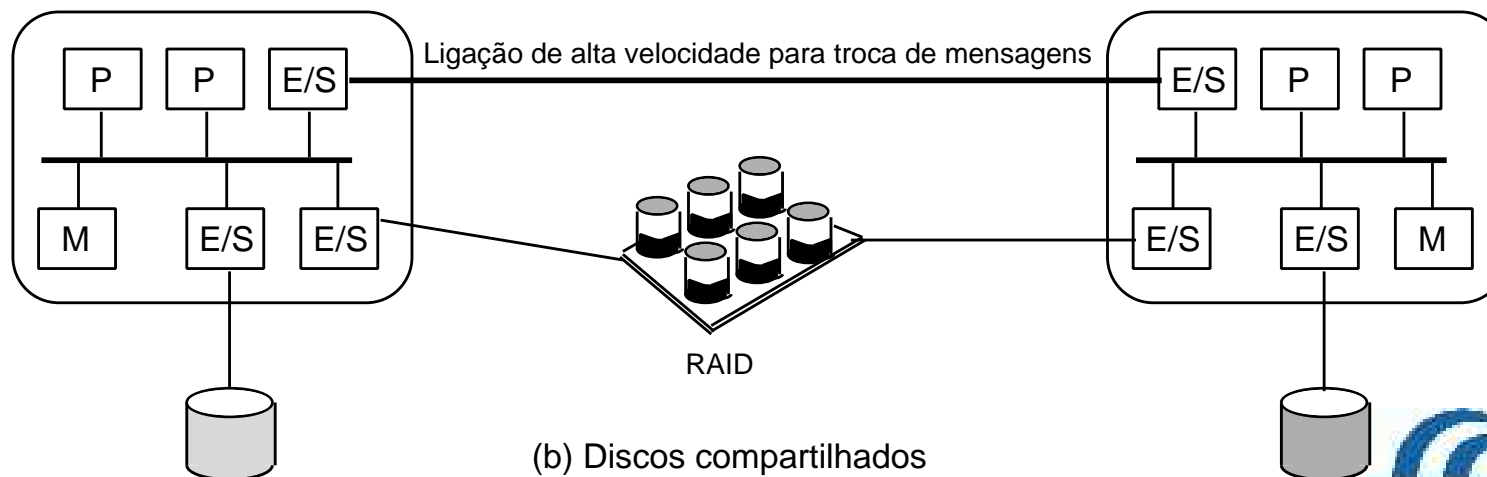
- Grupo de computadores completos interconectados, trabalhando juntos, como um recurso computacional unificado
- Benefícios:
  - Escalabilidade absoluta: é possível criar clusters muito grandes, dezenas de máquinas (cada máquina pode ser um multiprocessador)
  - Escalabilidade incremental: é possível expandi-lo de forma incremental
  - Alta disponibilidade: falha de um nó do cluster não significa perda total do serviço. Tolerância a falhas.
  - Melhor relação custo/ desempenho: facilidade de construir o sistema a partir de nós básicos comercialmente disponíveis

# Processamento Paralelo - Clusters

- Configurações: classificação baseada na forma como os computadores do cluster compartilham acesso aos discos.



(a) Servidor independente, sem compartilhamento de discos



(b) Discos compartilhados

(Fig. 16.9 do livro de Arquitetura e Organização de Computadores" , William Stalling)



# Processamento Paralelo - Clusters x SMP

SMP:

- Mais próximo do modelo de um único processador para o qual a maior parte das aplicações foi escrita
- Requer menos espaço físico e suprimento de energia que um cluster comparável

Clusters:

- Muito superiores em relação à escalabilidade absoluta e incremental
- Superiores em relação à disponibilidade, todos os componentes do sistema são altamente redundantes

# Processamento Paralelo - NUMA

## Definições:

- Acesso uniforme à memória (UMA): todos os processadores têm acesso a todas as partes da memória principal. O tempo de acesso é o mesmo - SMP.
- Acesso não uniforme à memória (NUMA): Todos os processadores têm acesso a todas as partes da memória principal. O tempo de acesso difere em relação à posição e processador.
- NUMA com coerência de cache (CC-NUMA): um sistema NUMA no qual é mantida coerência de cache entre as memórias cache dos vários processadores. Ex: Origin da Silicon Graphics, até 1024 processadores

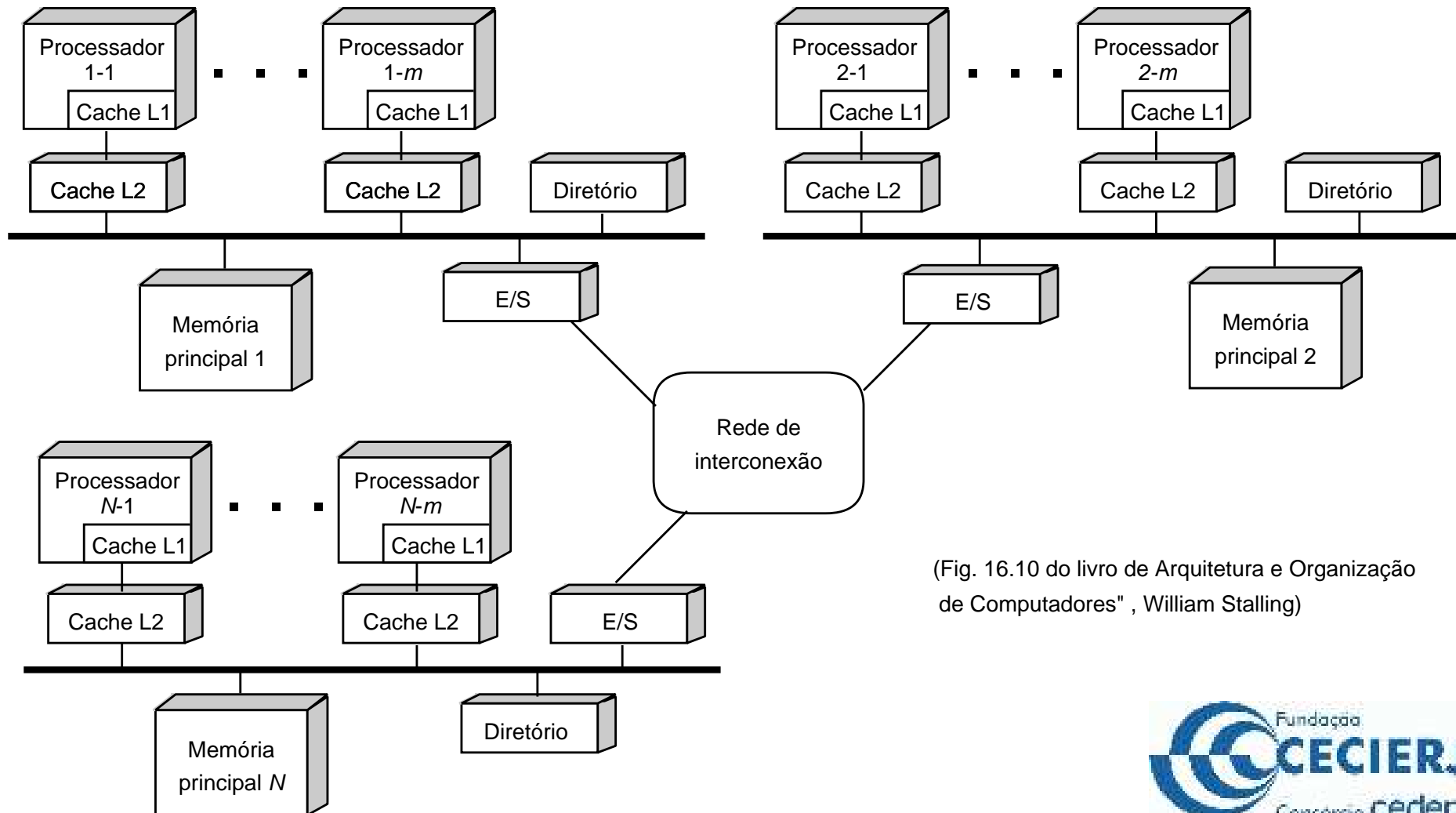
# Processamento Paralelo - NUMA

## Motivação:

- Em sistemas SMP existe um limite prático para o número de processadores que podem ser usados
- Um esquema de cache efetivo reduz o tráfego no barramento entre qualquer processador e a memória principal. A medida que aumenta o tráfego no barramento, este passa a constituir um gargalo de desempenho do sistema.
- A degradação limita o número de processadores
- Cluster: resolve o problema, mas a aplicação deve ser adaptada ao ambiente
- CC-NUMA grande área de memória no sistema, permitindo vários nós multiprocessadores, cada qual com o seu próprio barramento

## Organização:

- Cada nó do sistema inclui uma memória principal, mas do ponto de vista dos processadores existe uma única memória endereçável, com cada posição de memória tendo um endereço único em todo o sistema.



(Fig. 16.10 do livro de Arquitetura e Organização de Computadores", William Stalling)

# Processamento Paralelo - CC-NUMA

Consideração:

- Disponibiliza desempenho efetivo em níveis de paralelismo mais altos que o fornecido por sistemas SMP, sem requerer mudanças substanciais no software.

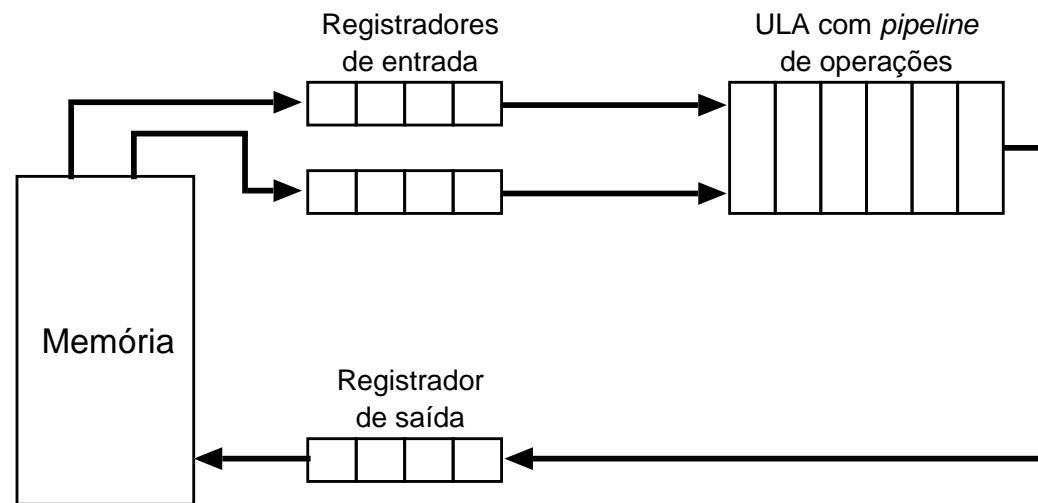
# Processamento Paralelo - Computação Vetorial

- Computadores capazes de resolver problemas matemáticos relativos a processos reais em: aerodinâmica, sismologia, meteorologia e física atômica, nuclear e de plasma.
- Problemas efetuam repetidas operações aritméticas de ponto-flutuante em grandes vetores de números
- Supercomputadores muito custosos, otimizados para computação vetorial, projetados para efetuar centenas de milhões de operações de ponto flutuante por segundo

# Processamento Paralelo - Computação Vetorial

Abordagens:

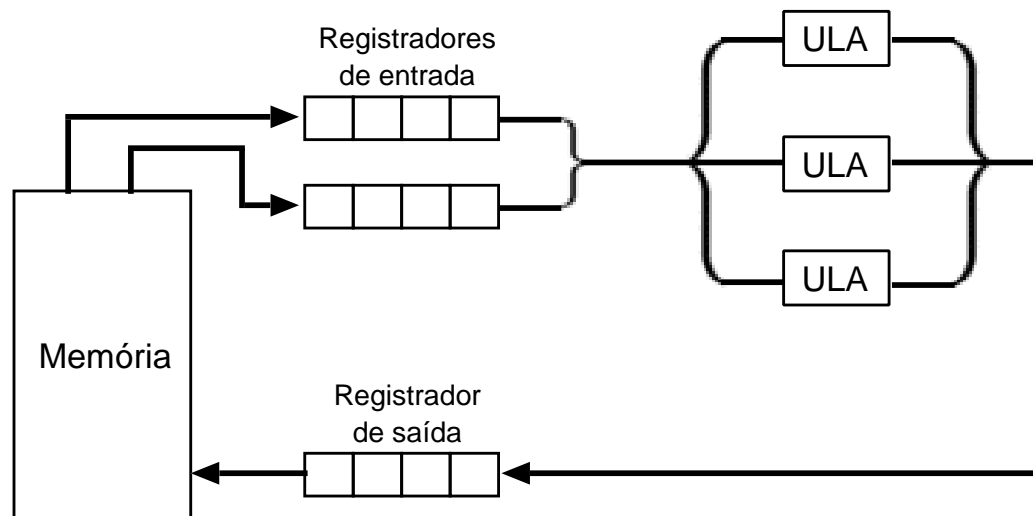
- ULA com pipeline de operações (vector processor)



(Fig. 16.13(a) do livro de Arquitetura e Organização de Computadores", William Stalling)

# Processamento Paralelo - Computação Vetorial

- ULAs paralelas (array processor)

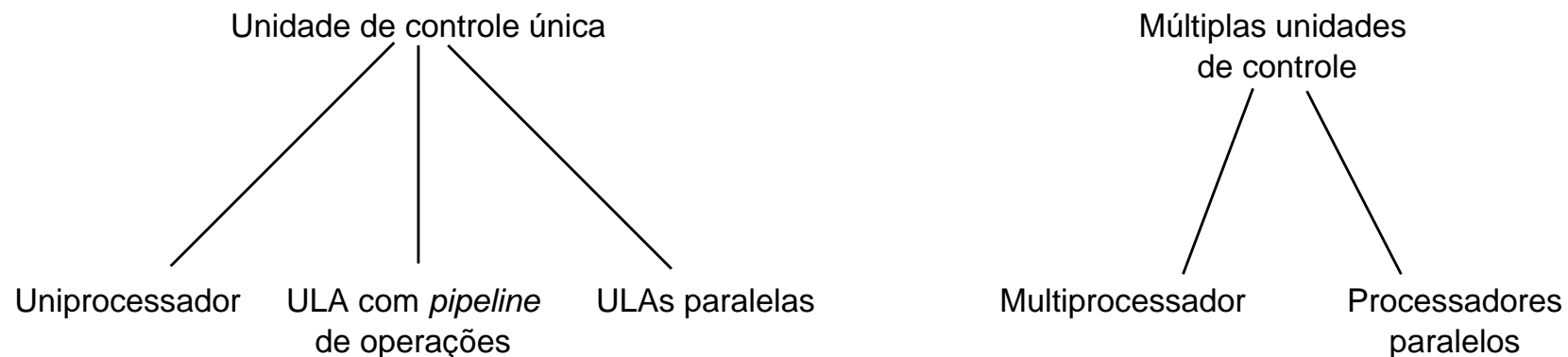


(Fig. 16.13(b) do livro de Arquitetura e Organização de Computadores", William Stalling)



# Processamento Paralelo - Computação Vetorial

- Processadores Paralelos: múltiplos processadores trabalhando de forma cooperativa sobre uma dada tarefa



(Fig. 16.15 do livro de Arquitetura e Organização de Computadores" , William Stalling)

# Bibliografia

*Arquiteturas RISC: Introdução à Organização de Computadores -  
Mário A. Monteiro - LTC- Capítulo 11*

*Processamento Paralelo: Arquitetura e Organização de  
Computadores - William Stalling - Prentice-Hall - Capítulo 16*

# Exercícios

*Arquiteturas RISC: Introdução à Organização de Computadores -  
Mário A. Monteiro - LTC- Capítulo 11*

*Exercícios: 1, 2 ,3, 4 e 6*

*Processamento Paralelo: Arquitetura e Organização de  
Computadores - William Stalling - Prentice-Hall - Capítulo 16*

*Exercício: 16.8*