Panorâmica Geral de um Sistema Computacional

Introdução

- Conceitos base
- Diferentes perspectivas de um sistema computacional
  - ✓ Utilizador
  - ✓ Programador
  - ✓ Arquitecto
  - ✓ Implementador
- Componentes de um sistema computacional
  - ✓ Processador
  - ✓ Memória
  - ✓ Periféricos
  - ✓ Interconexão (bus)

Conceitos Base - Computador

- Computador - Equipamento com uma ou mais unidades (de processamento e periféricas) que é capaz de computar controlado por programas internos, **sem intervenção humana**

- São Turing-complete, ou seja, máquinas de computação universais

Alan Turing

- Matemático Inglês, considerado por alguns o pai da Ciência da Computação
  - ✓ Contribuiu nos domínios da lógica, criptografia, inteligência artificial, **formalização da algoritmia e da computação**
  - ✓ Máquina de Turing
    - ✓ Define formalmente a "máquina universal"
  - ✓ Trabalhou no projecto Colossus

1912 - 1954

Conceitos Base

- Arquitectura de um computador – Especificação lógica do computador, desenho do interface de programação

- Organização de um computador – Como é que os componentes operam e interagem, como é que o conjunto de instruções é realizado
  - ✓ Hardware ou software?

- Desenho/Implementação de um computador – Implementação ao nível do hardware da específicação da arquitectura e da organização

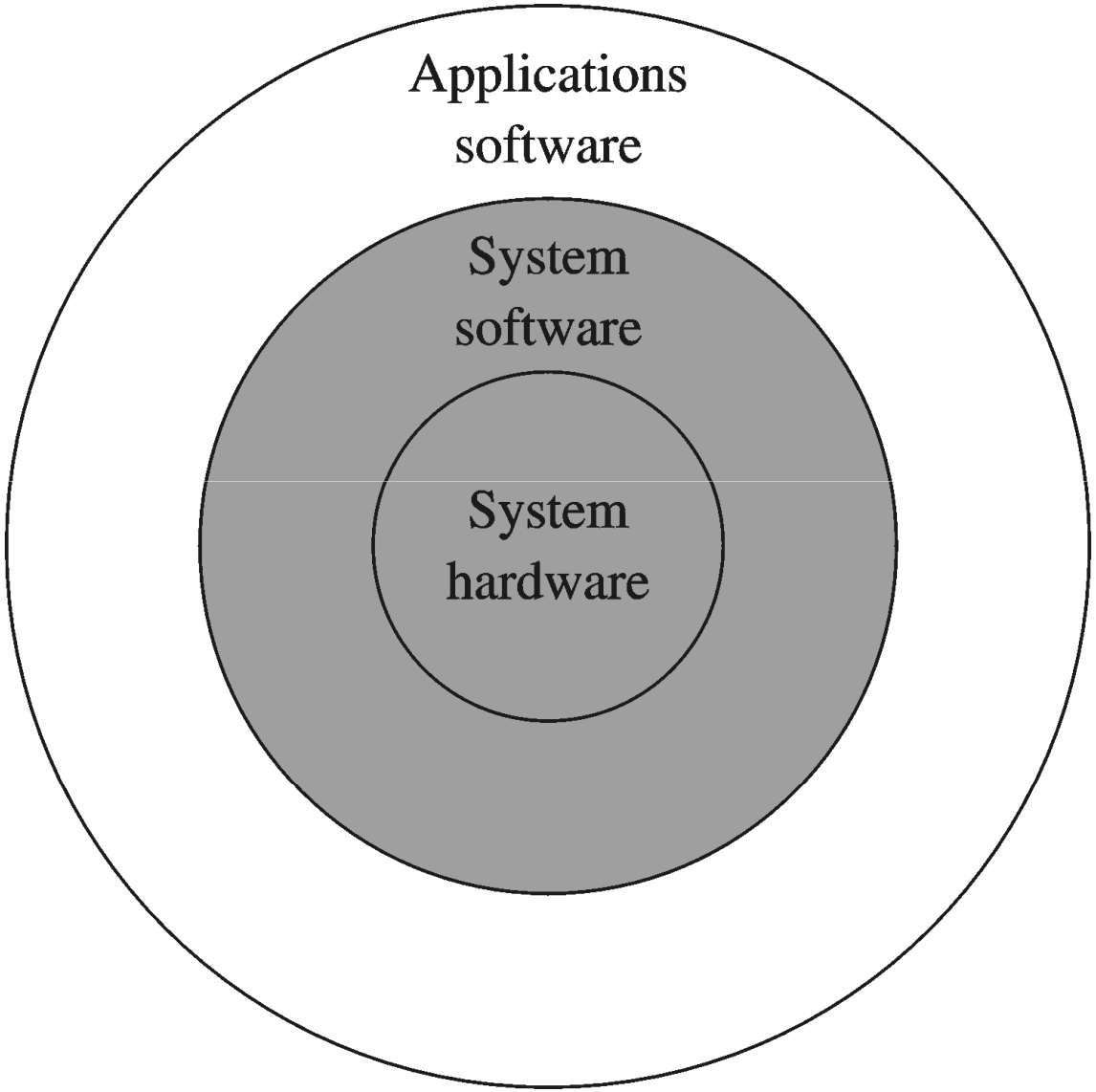
Conceitos Base - Programa

- Conjunto de **instruções** escritas numa linguagem que alguma "máquina" é capaz de **reconhecer e executar**
- Instruções:
  - ✓ Mais próximas do domínio de aplicação
    - ✓ mais complexas e específicas
  - ✓ Mais próximas da arquitectura do computador
    - ✓ mais simples e genéricas
- Múltiplos níveis de linguagens
- Noção de interface
- Noção de transparência (ou opacidade)

Conceitos Base

Termo	Décimal	Binário
K (kilo)	10^3	2^{10}
M (mega)	10^6	2^{20}
G (giga)	10^9	2^{30}
T (tera)	10^{12}	2^{40}
P (peta)	10^{15}	2^{50}

Perspectiva do Utilizador



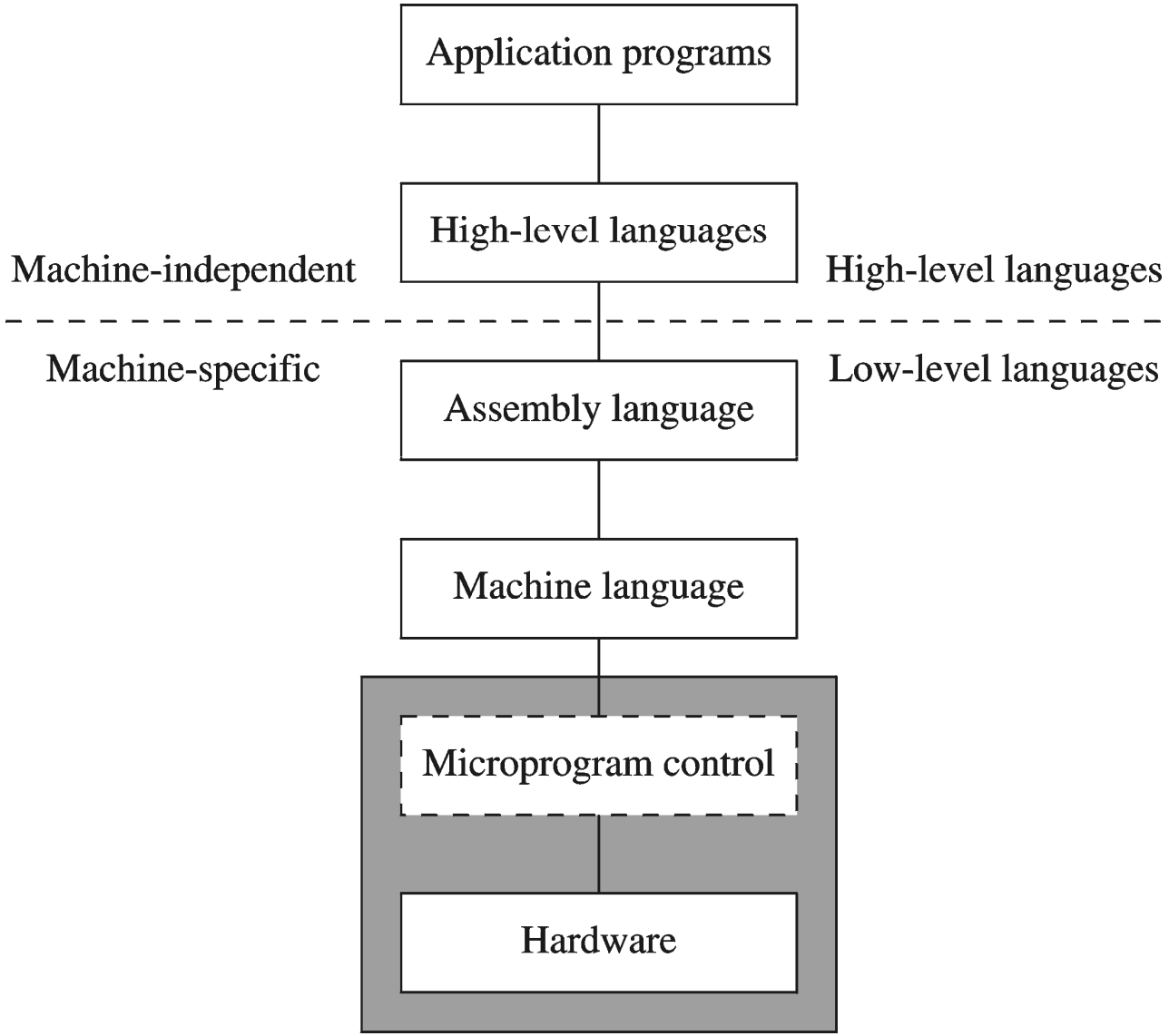
Perspectiva do Programador

- Depende do tipo e nivel da linguagem utilizada
- Hierarquia de linguagens
  - Linguagem máquina
  - Assembly
  - Linguagens de alto-nível
  - Aplicações

Aumento do nível de abstracção

- Independentes da máquina
  - ✓ Linguagens de alto-nível e aplicações
- Específicas da máquina
  - ✓ Linguagem máquina e assembly

Perspectiva do Programador



Linguagem máquina

- Linguagem nativa ao processador
- Consiste num alfabeto de 0s e 1s
  - ✓ Exemplo: 1111 1111 0000 0110 0000 1010 0000 0000B

- Tipos de dados
  - ✓ Inteiros (1, 2, 4 ou 8 bytes)
    - ✓ Servem também para representar endereços
  - ✓ Números em vírgula flutuante (4, 8 ou 10 bytes)
  - ✓ Não tem dados estruturados, como vectores
    - ✓ Apenas dados reservados (alocados) continuamente em memória

Linguagem máquina

- Operações
  - ✓ Aritmética sobre dados em registos ou memória
  - ✓ Transferir dados entre registos e memória
    - ✓ Load – carregar dados de memória para um registo
    - ✓ Store – guardar o conteúdo de um registo em memória
  - ✓ Controlo do fluxo de execução
    - ✓ Saltos condicionais e incondicionais
  - ✓ Chamadas a subrotinas

Assembly

- Conjunto de menmónicas que tornam o código máquina legível para humanos
  - ✓ Exemplo: inc eax
- Preserva o modelo da linguagem máquina, incrementando um pouco o nível de abstracção
  - ✓ Exemplo: podemos usar a mesma mnemónica para fazer a mesma operação sobre tipos de dados diferentes
- Tem uma correspondencia quase directa para com a linguagem máquina
- Serve como linguagem intermédia na compilação de linguagens de mais alto-nível

Tradução Assembly – Ling. Máquina

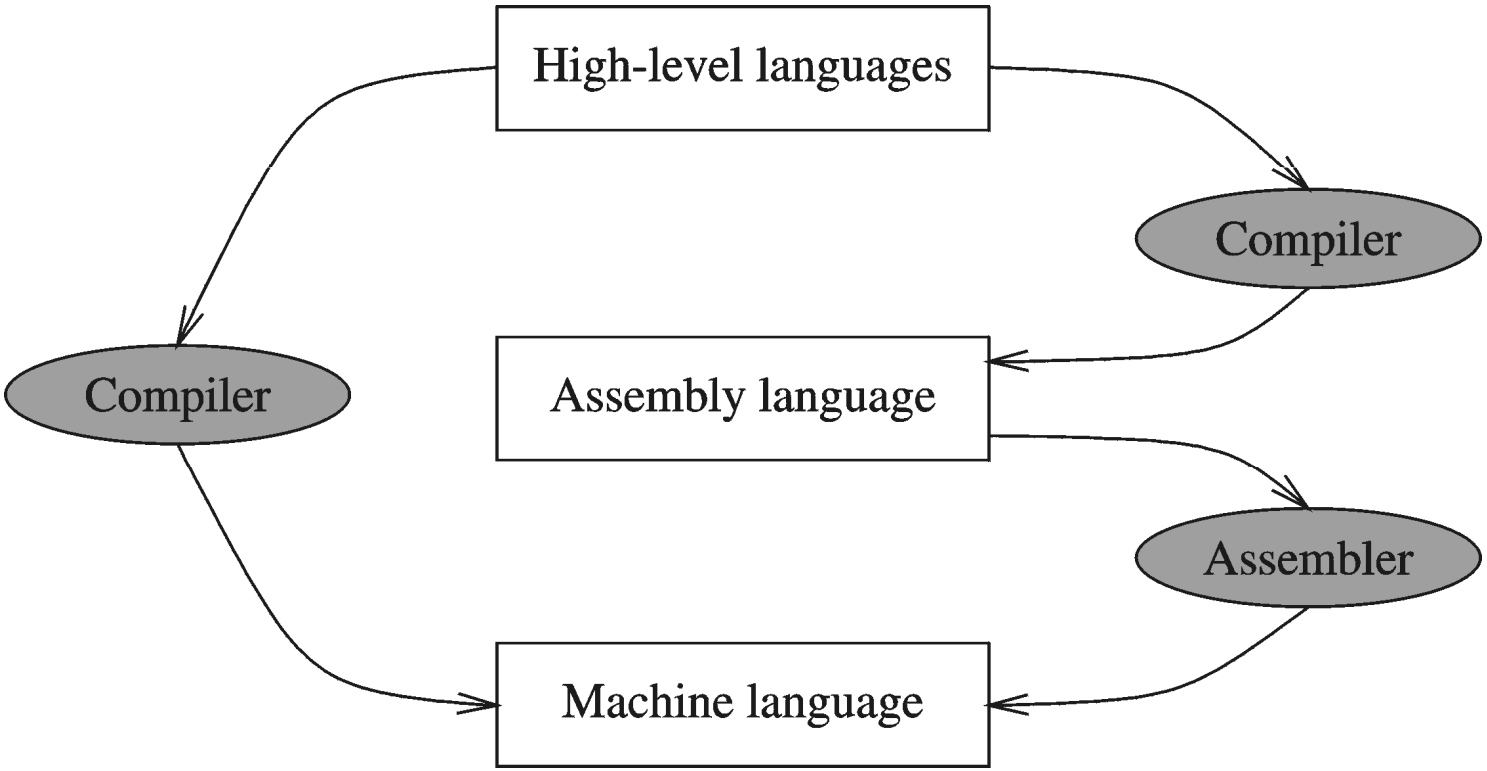
Linguagem Assembly	Linguagem Máquina (em Hexadecimal)
inc [result]	FF060A00
mov [class_size],45	C7060C002D00
and [mask],128	80260E0080
add [marks],10	83060F000A

Assembler

- O assembler tem o papel de traduzir as mnemónicas (linguagem assembly) no respectivo código máquina do processador
- Podemos escrever (em assembly) instruções que não existem no ISA
  - ✓ O assembler verifica isso, e dará o respectivo erro
- Existem outras instruções que são destinadas ao próprio programa assembler (directivas)
- Para um dado código máquina podem existir vários "assemblies"
  - ✓ Podemos definir um nós próprios, desde que implementemos o seu assembler

Compilador

- Traduz o código fonte da linguagem de alto-nível para código máquina
  - ✓ Directamente
  - ✓ Indirectamente, via assembler



Tradução Linguagem C - Assembly

Linguagem C	Linguagem Assembly
result = count1 + count2 + count3 + count 4	**mov eax**, [count1] **add eax**, [count2] **add eax**, [count3] **add eax**, [count4] **mov** [result], **eax**

Tradução Linguagem C - Assembly

- Algumas instruções de alto nível podem ser mapeadas directamente em instruções assembly

Linguagem C	Linguagem Assembly
result++;	**inc** [result]
size = 45;	**mov** [size], 45
marks += 10;	**add** [marks], 10

Tradução Linguagem C - Assembly

- Algumas instruções de alto nível podem ser mapeadas directamente em instruções assembly

Linguagem C	Linguagem Assembly
size = value;	**mov eax**, [value] **mov** [size], **eax**
sum += x + y + z	**mov eax**, [sum] **add eax**, [x] **add eax**, [y] **add eax**, [z] **mov** [sum], **eax**

Instruction Set Architecture (ISA)

- A especificação do conjunto de instruções fornecido por um processador e da sua semântica constitui uma *Instruction Level Architecture* (ISA)
- O ISA define um processador lógico que é independente da sua implementação
- Exemplos de ISAs são: IA-32, AMD64/Intel64, SPARC e JVM
- A implementação do IA-32 numa máquina Intel difere duma máquina AMD
- A implementação da JVM no Windows difere, por exemplo, da do Linux

Vantagens das Linguagens de Alto-nível

- Desenvolvimento mais rápido
  - ✓ Instruções de mais alto-nível → menos instruções por programa
- Manutenção mais simples
- Programas são portáveis
  - ✓ Pouca dependência da máquina alvo
  - ✓ Compilador traduz para o código máquina do ISA correspondente
  - ✓ Programas em assembly não são portáveis

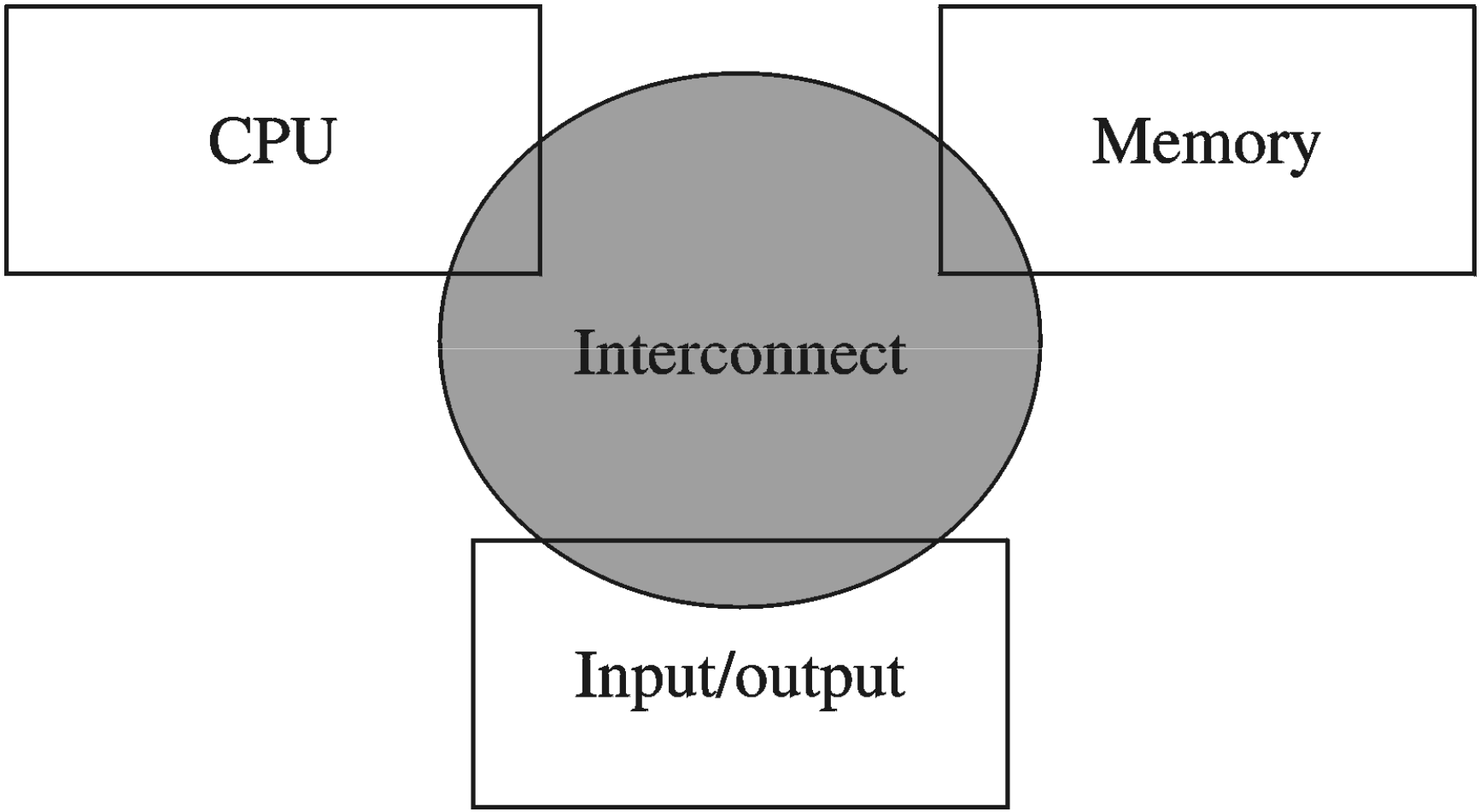
Porquê Programar em Assembly?

- Eficiência
  - ✓ Espacial – programas em assembly resultam, normalmente, em programas em código máquina mais compactos
  - ✓ Temporal – ao programar mais perto da máquina pode-se afinar o código para uma determinado ISA e obter ganhos de performance

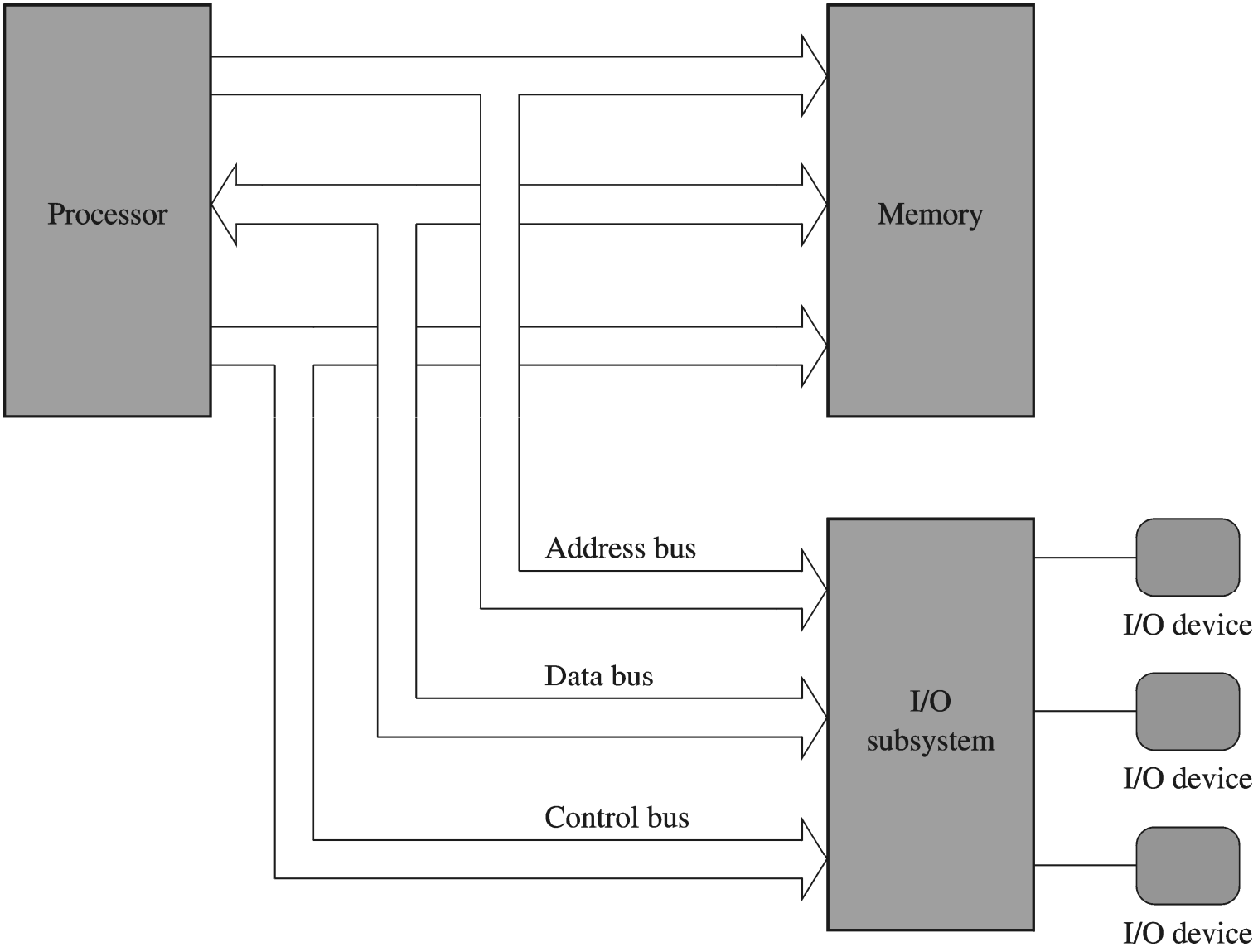
Perspectiva do Arq. de Computadores

- Perspectiva de alto-nível
  - ✓ Não se concentra nos detalhes de baixo-nível
    - ✓ Exemplo: Flip-flops
  - ✓ Constroi o computador a partir de funcionalidades - componentes de mais alto-nível
    - ✓ Exemplo: ALU
  - ✓ Consiste em três componentes princípais
    - ✓ Processador
    - ✓ Memória
    - ✓ Dispositivos de entrada/saída
  - ✓ Ligados através de uma rede de interconexão composta por vários bus

Perspectiva do Arq. de Computadores



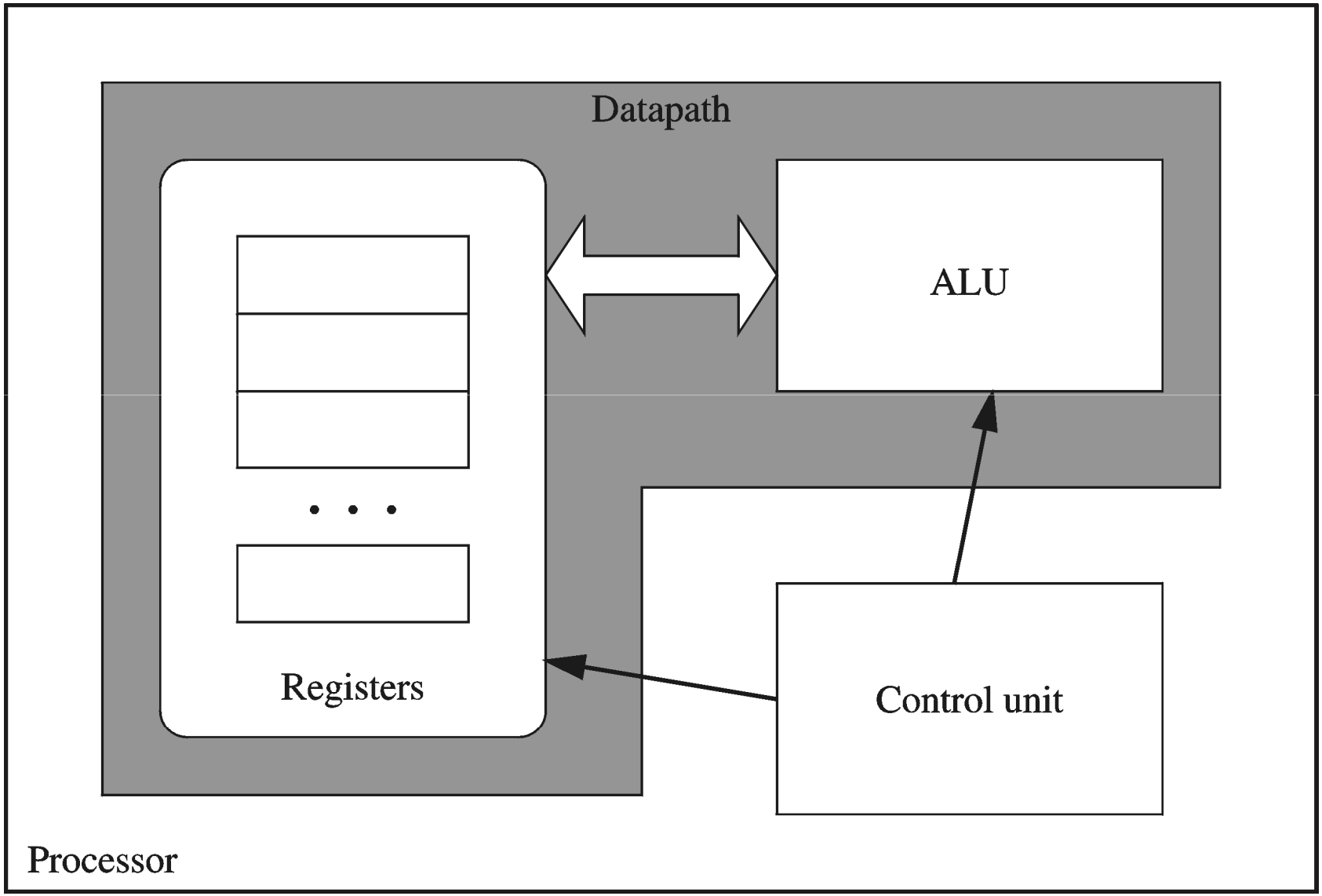
Perspectiva do Arq. de Computadores



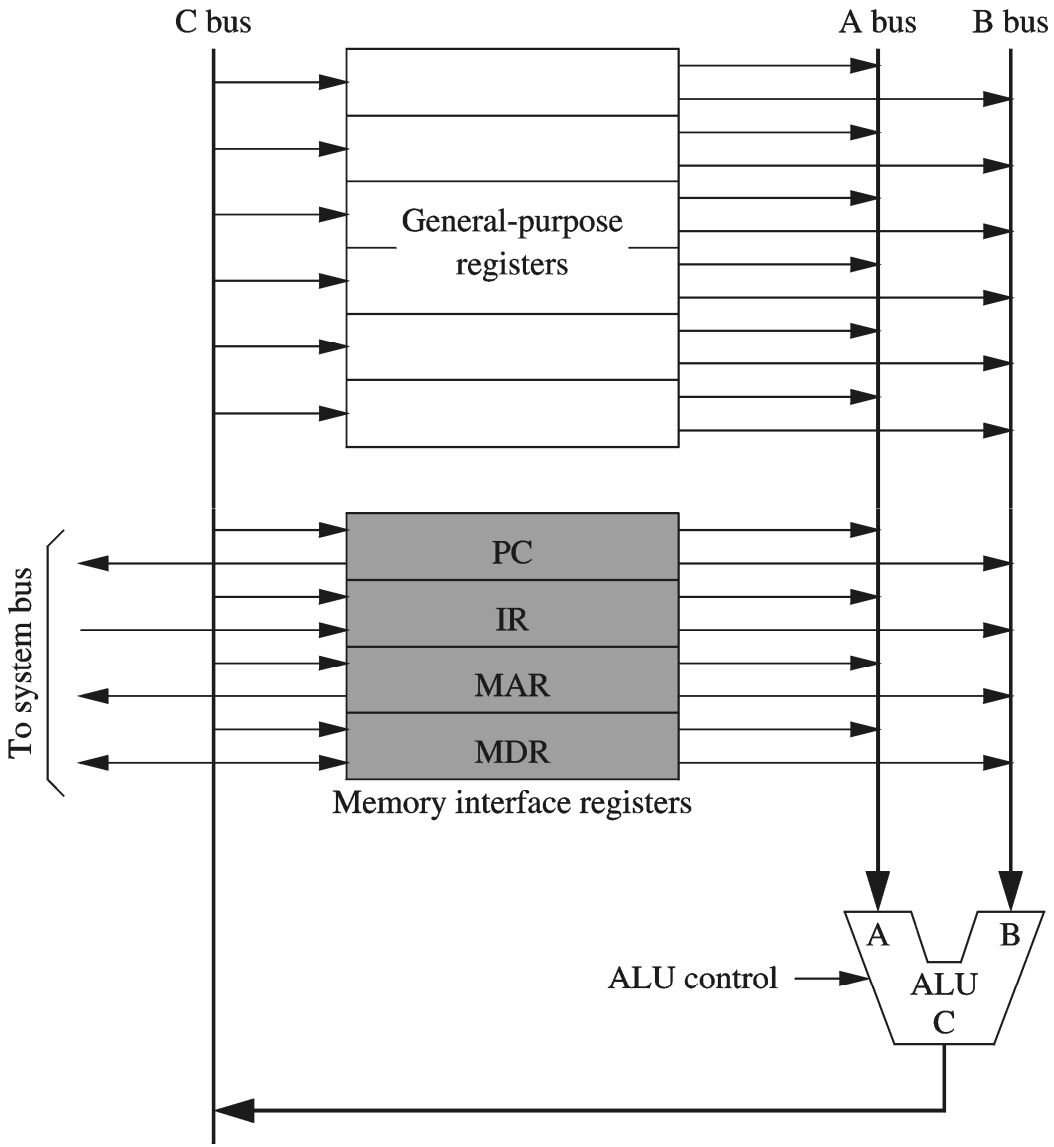
Perspectiva do Implementador

- Implementam/realizam a especificação da arquitectura
- Utilizam componentes de baixo-nível
- O processador, por exemplo, é composto por:
  - ✓ Unidade de control
  - ✓ Fluxo de dados
    - ✓ ALU
    - ✓ Registos

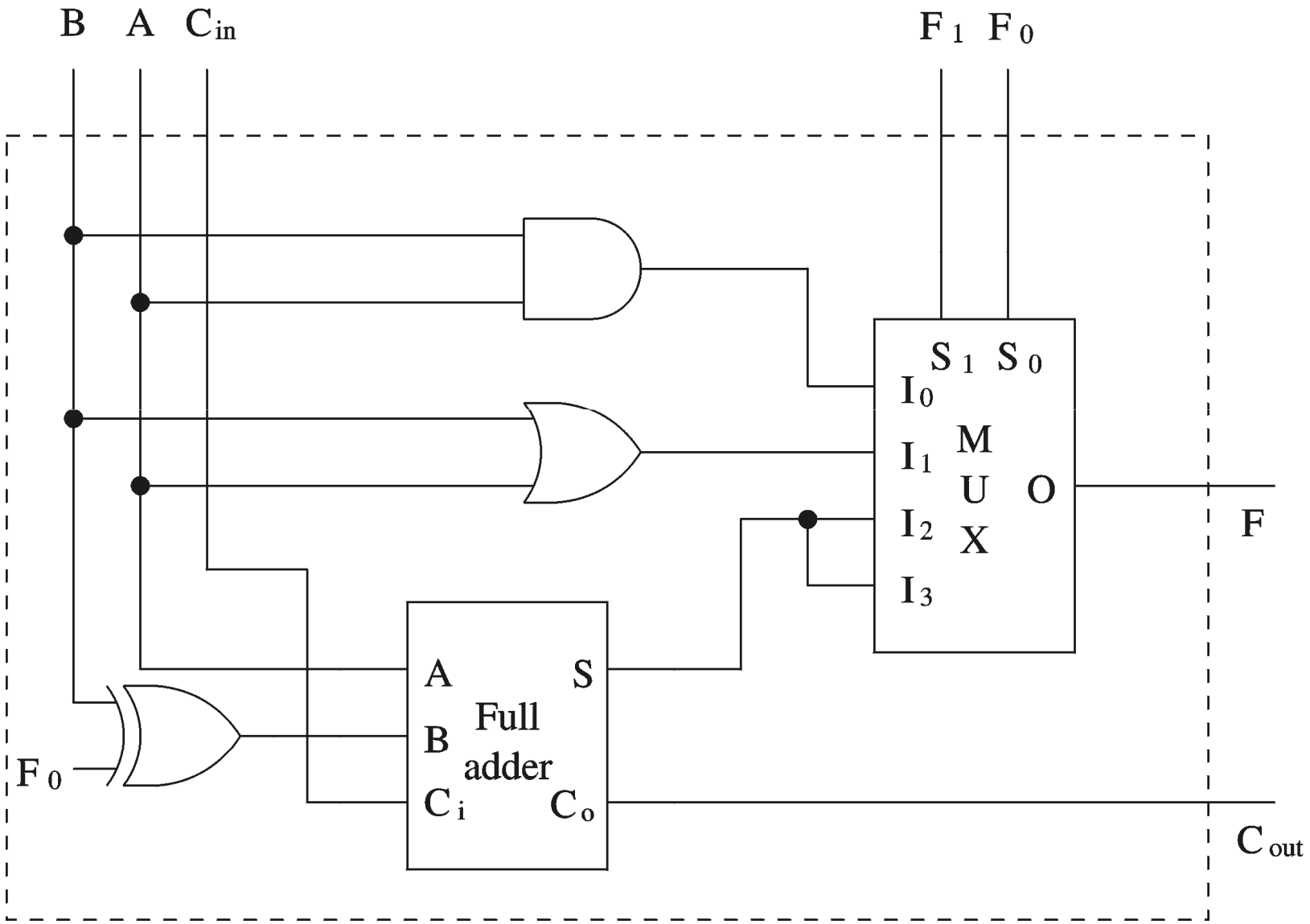
Perspectiva do Implementador



Perspectiva do Implementador



Perspectiva do Implementador



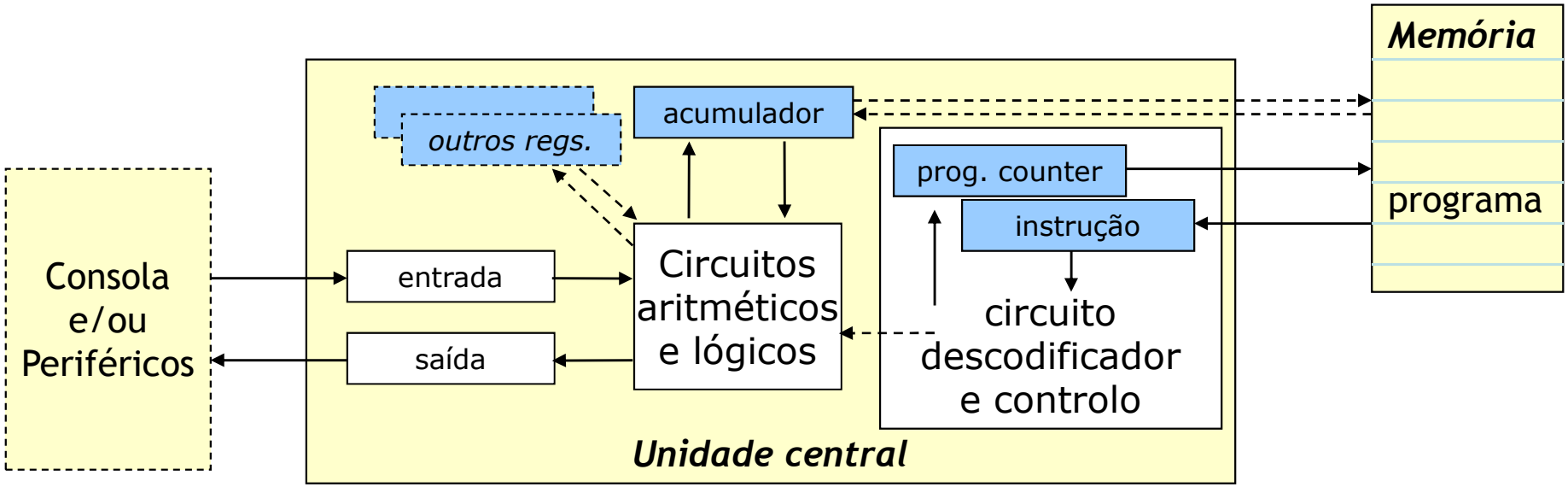
Processador

- Controlo global das operações do computador e responsável pela execução das instruções

- Contém:
  - ✓ Unidade de controlo: obtém, descodifica e interpreta as instruções (uma de cada vez)
  - ✓ Unidade aritmética e lógica: ALU – *Arithmetic and Logic Unit*
  - ✓ Conjunto de registadores (ou registos): células de memória locais ao CPU, para dados a usar pelas instruções e para valores intermédios

Processador

- Arquitectura de von Neumann
  - ✓ Von Neumann (e outros) propõem que o programa seja codificado numa memória interna tal como os dados
  - ✓ Não há distinção entre dados e código
  - ✓ A unidade de controlo é reponsável pelo acesso e descodificação das instruções



John von Neumann

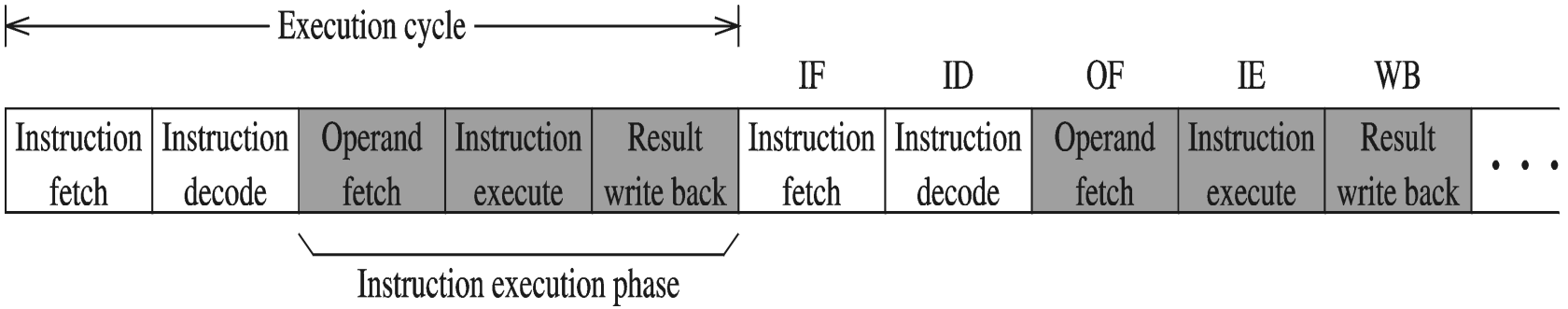
- Matemático Húngaro (naturalizado Americano), considerado por alguns o pai da arquitectura dos computadores binários modernos
  - ✓ Contribuiu nos domínios da física quântica, economia, estatística, **computação**
  - ✓ Autómato celular
  - ✓ Máquina de von Neumann
  - ✓ **EDVAC** (***E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer*)

1903 - 1957

Processador – Ciclo de Funcionamento

- Ciclo de funcionamento:
  - ✓ Fetch (obter instrução da memória)
  - ✓ Decode (descodificar instrução)
  - ✓ Execute (obter operandos executar instrução e guardar resultado)

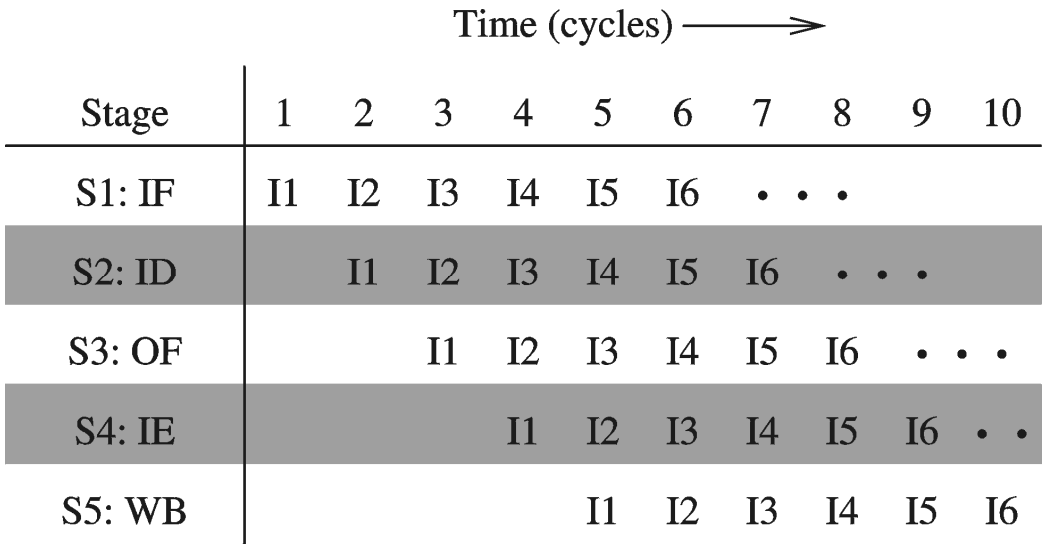
- Instruções executadas sequencialmente



Processador - Pipelining

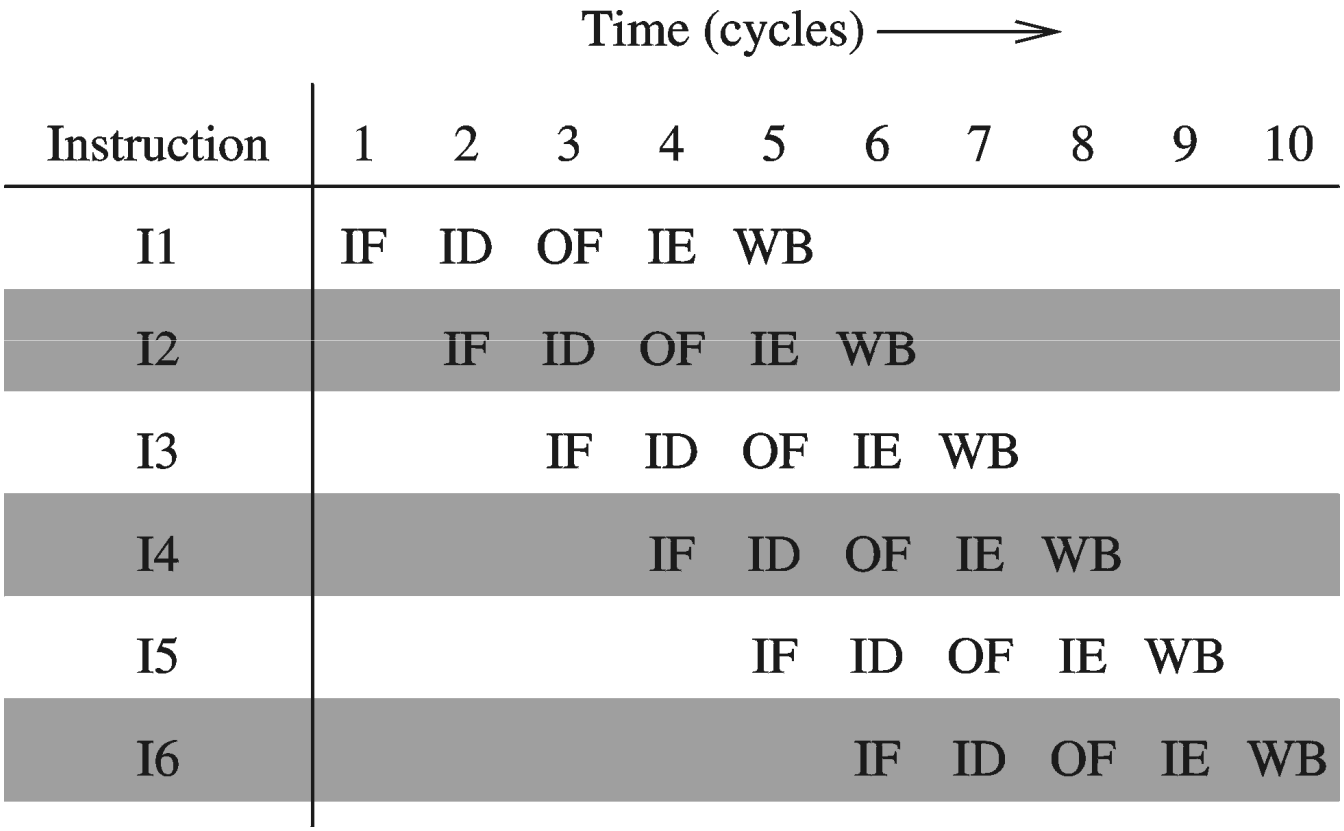
Pipelining

- ✓ Colmatar o problema gerado pelo fosso entre a velocidade do processador e da memória
- ✓ Sobreposição de comunicação (acesso a memória) com computação
- ✓ Aumenta o throughput → número de instruções concluídas por unidade de tempo



Processador - Pipelining

- Uma outra perspectiva



Processador - Complexidade

- A complexidade do processador é influenciada por:
  - ✓ Tipos de instruções
  - ✓ Número e tipo de operandos
  - ✓ Modos de endereçamento dos operandos
  - ✓ etc.
- O desempenho do processador é influenciado por essa complexidade:
  - ✓ Descodificação mais complexa (recurso a micro-código)
  - ✓ Instruções de tamanho variável
  - ✓ Resolução do endereço dos operandos e obtenção do seus valores mais complexa/demorada
- Mais complexidade → mais circuitos
  - ✓ CPU maior, mais lento, consumindo mais energia, etc…

Processador – CISC

- Até aos anos 70/80 a abordagem foi suportar directamente no processador os mais variados tipos de instruções que os programas podem necessitar
  - ✓ As mais variadas operações aritméticas e lógicas…
- Cada instrução suporta os mais variados operandos que o programa pode necessitar
  - ✓ Registos, memória (com vários modos de endereçamento), etc.
- A prioridade é reduzir o tamanho dos programas
  - ✓ Claro que também se procura reduzir o tempo de execução de cada instrução
- A abordagem referida por CISC – Complex Instruction Set Computer

Processador - RISC

- RISC – Reduced Instruction Set Computer
- Nova abordagem (anos 70/80) no desenho dos CPU. Simplificar para conseguir melhor desempenho:
  - ✓ Suportar um pequeno conjunto de instruções: as mais usadas
  - ✓ Instruções de tamanho fixo: Fetch mais simples e eficiente
  - ✓ Descodificação mais simples e eficiente
  - ✓ Menos instruções a optimizar, a execução pode ser mais eficiente
  - ✓ Usar espaço no CPU para mais registos e mais cache
  - ✓ Permitir explorar mais optimizações…

Processador – CISC vs. RISC

CISC

- Muitas instruções
- Muitos modos de endereçamento
- Instruções demoradas
  - Muitas acedem a memória
- Poucos registos
- Normalmente requer um microprograma para traduzir as instruções para outras mais simples

RISC

- "Poucas" instruções
- Poucos modos de endereçamento
- Arquitecturas load/store
  - Só as instruções load e store acedem a memória
- Muitos registos
- A menor complexidade do CPU abre a oportunidade para optimizações: diminuir o consumo de energia, optimizar a execução (ex: pipelines), etc.

Processador – CISC vs. RISC

- Computar: C = A + B

- CISC:

  load R1, [A]
  add R1, [B]
  store [C], R1

  ou mesmo:

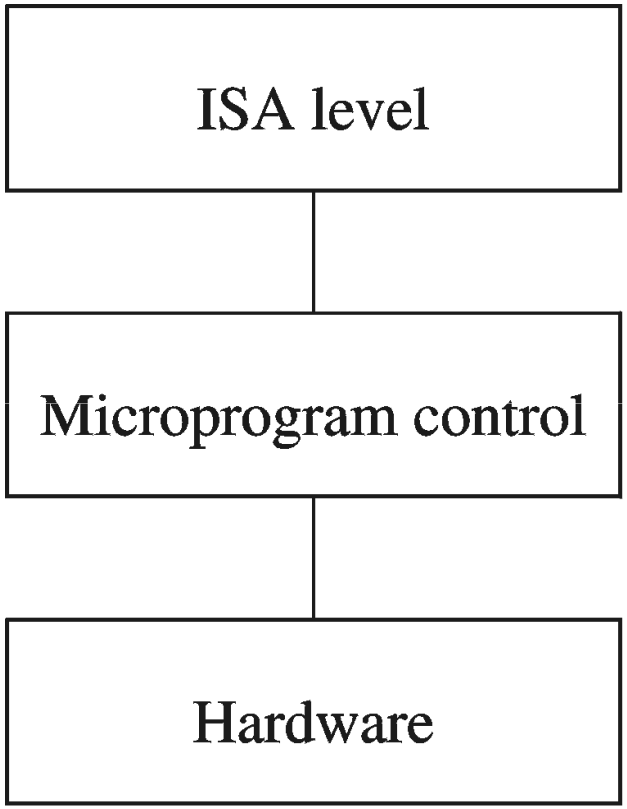
  add [C], [A], [B]

- RISC:

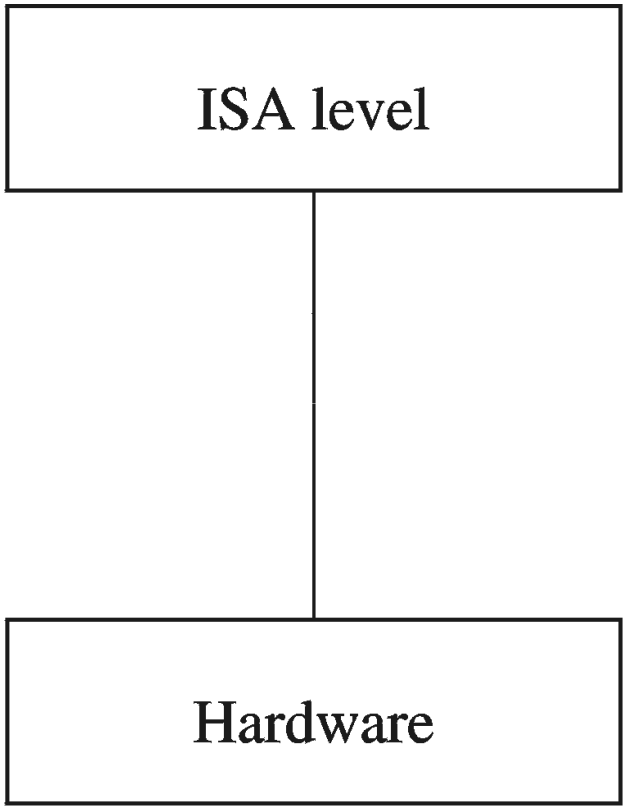
  load R1, [A]
  load R2, [B]
  add R1, R2
  store [C], R1

- Qual será mais eficiente?
  - ✓ RISC se cada instrução demorar menos tempo!!

Processador – CISC vs. RISC



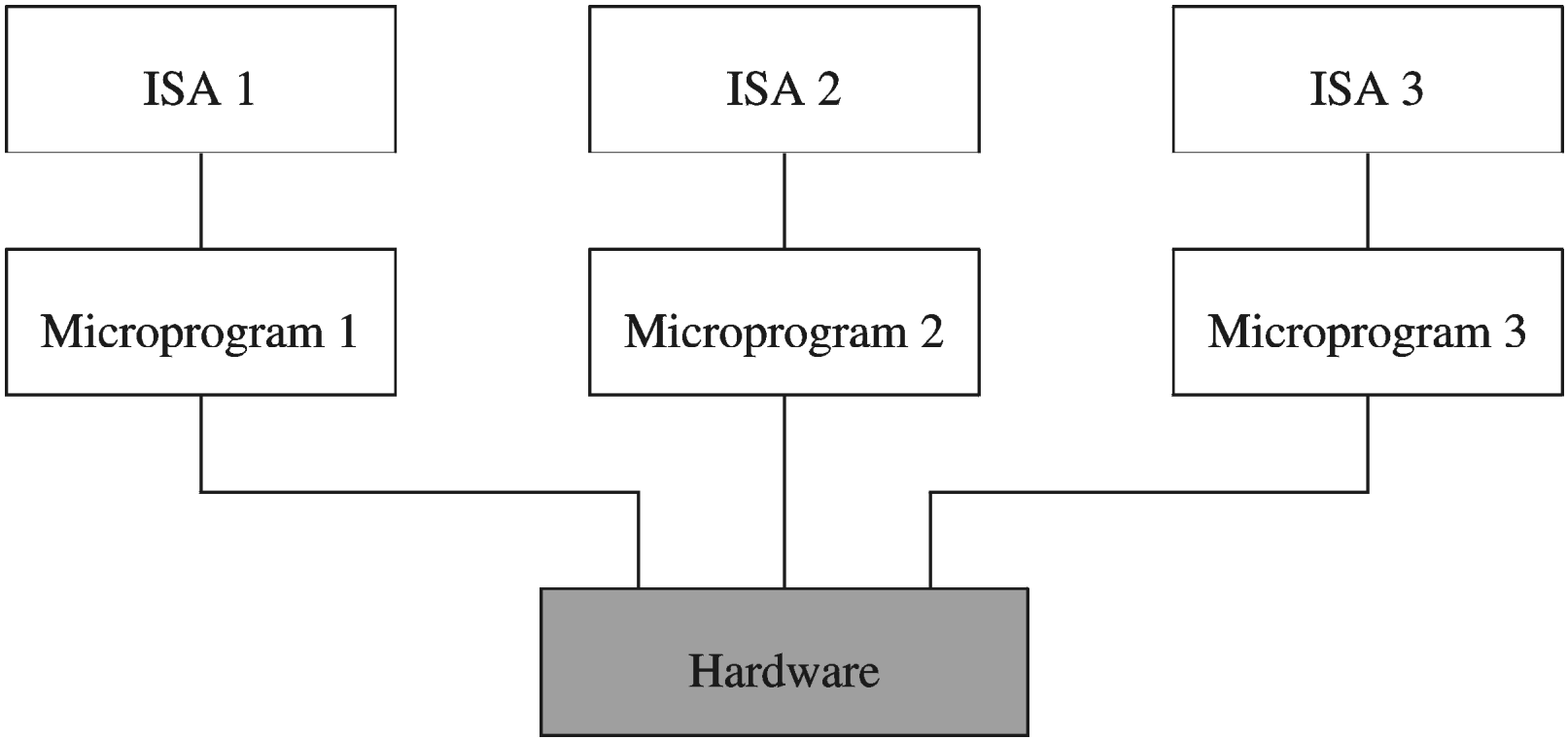
(a) CISC implementation



(b) RISC implementation

Processador

- Diferentes microprogramas podem implementar diferentes ISAs (processadores lógicos) no mesmo processador físico



Memória

- Guarda instruções e dados, sob a mesma representação simbólica, em células que contêm agrupamentos de bits:
  - ✓ byte (8 bits) ou palavra (normalmente da dimensão ou (sub)múltiplo do bus de dados)
- É acedida como um vector:
  - ✓ Mem[0]… Mem[n-1] (capacidade = n bytes)
  - ✓ O endereço da célula corresponde ao índice i em Mem[i]
  - ✓ O endereço é representado em binário, como um número inteiro sem sinal
  - ✓ O valor de Mem[i] é o do conteúdo da célula
  - ✓ O acesso é directo: dá-se i para aceder a Mem[i]
    - ✓ (RAM: Random Access Memory)

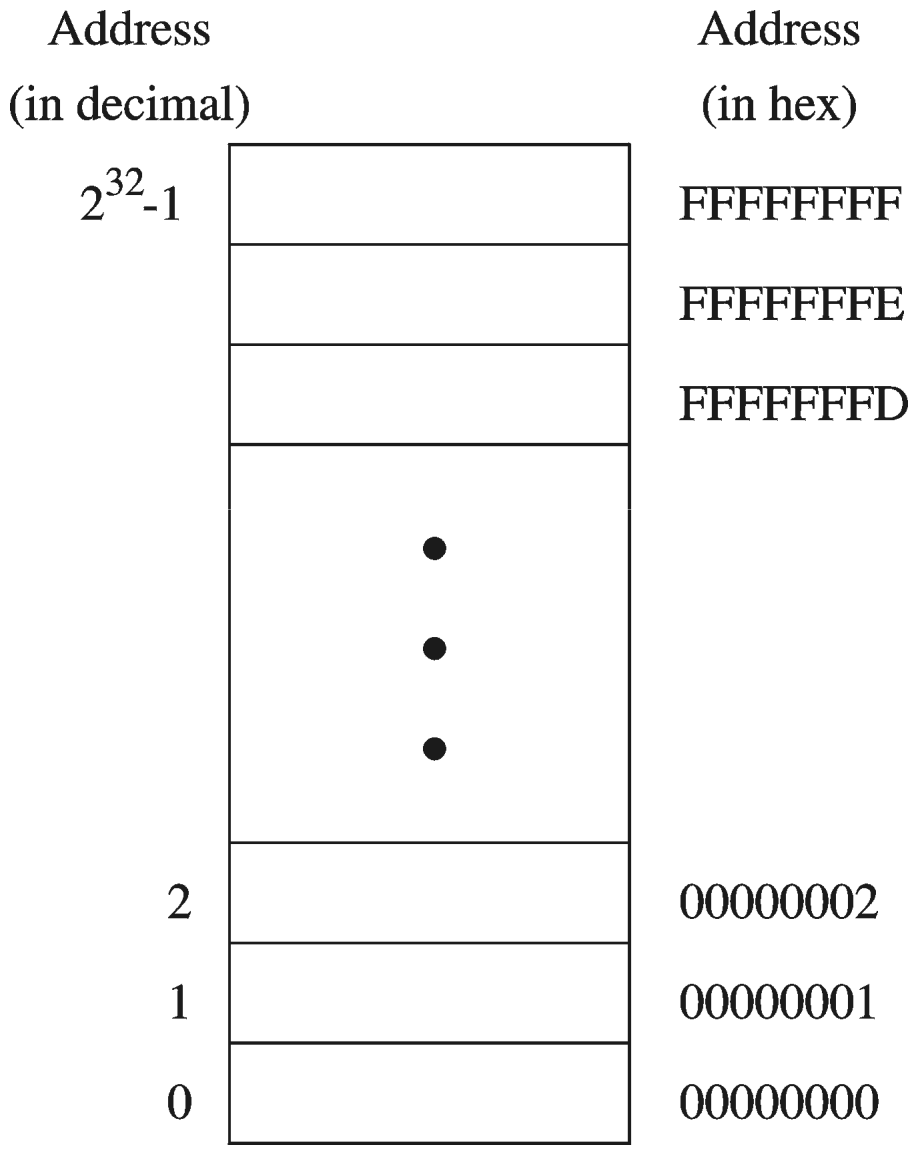
Memória – Espaço de Endereçamento

- Em quase todos os processadores a memória é **endereçado ao byte**, ou seja:
  - ✓ Cada posição do vector contém um byte, ou
  - ✓ Cada cada byte tem o seu próprio endereço
- Espaço de endereçamento – quantidade de memória a que se pode aceder a partir de um programa
  - ✓ O programador tem a perspectiva dada pelo ISA
    - ✓ O espaço de endereçamento é limitado número de bits do endereço
  - ✓ Por exemplo o Pentium a 32bits tem endereços a 32 bits
    - ✓ Espaço de endereçamento comporta 2^{32} células de memória
    - ✓ A memória é endereçada ao byte, logo o total de memória endereçável é de 2^{32} * 1 byte = 4 Gbytes

Memória - Espaço de Endereçamento

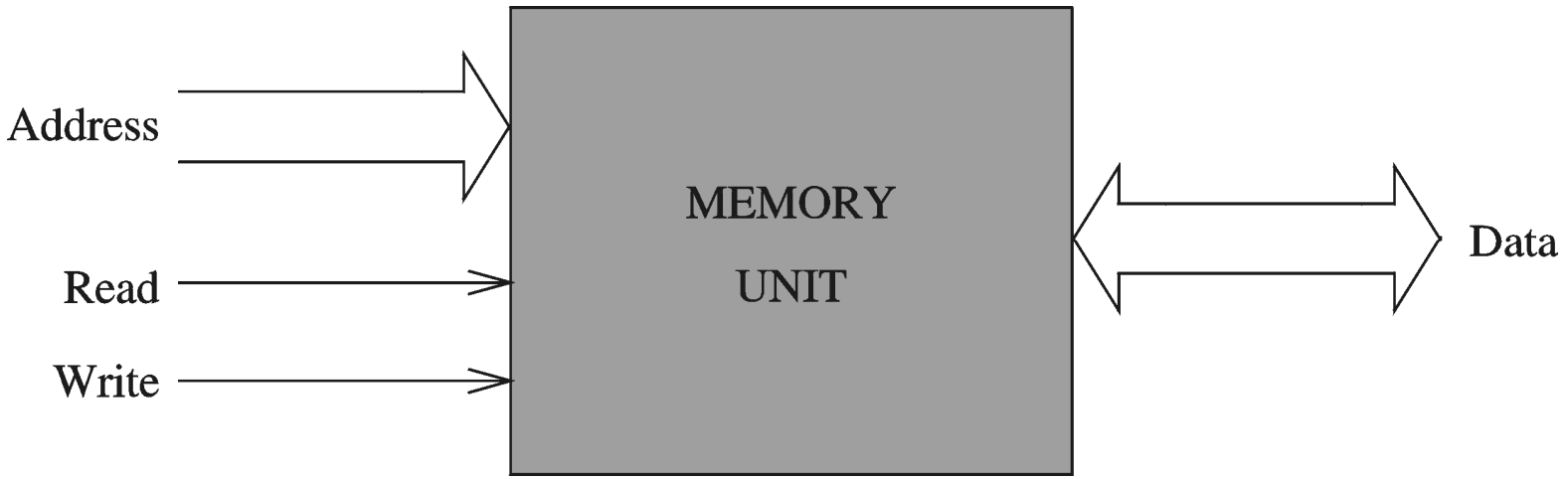
- Portanto o número de bits do endereço determina a capacidade máxima de memória endereçável
- No entanto é o número de linhas no bus de endereços determina a capacidade máxima de memória realmente acessível
- Por exemplo o ISA do Intel Core 2 a 64 bits, o AMD64/Intel64, fornece endereços de 64 bits
  - ✓ Logicamente endereça 16Ebytes (Exa = 2^{60})
  - ✓ No entanto, por enquanto, não se justifica um bus de endereços de 64bits. Actualmente tem apenas 48
  - ✓ A quantidade de memória realmente acessível é 256Tbytes (Tera = 2^{40})

Memória – Espaço de Endereçamento



Memória – Unidade de Memória

- Unidade de Memória
  - ✓ Endereço
  - ✓ Dados
  - ✓ Sinais de controlo
    - ✓ Leitura (Read)
    - ✓ Escrita (Write)



Memória - Acesso de leitura

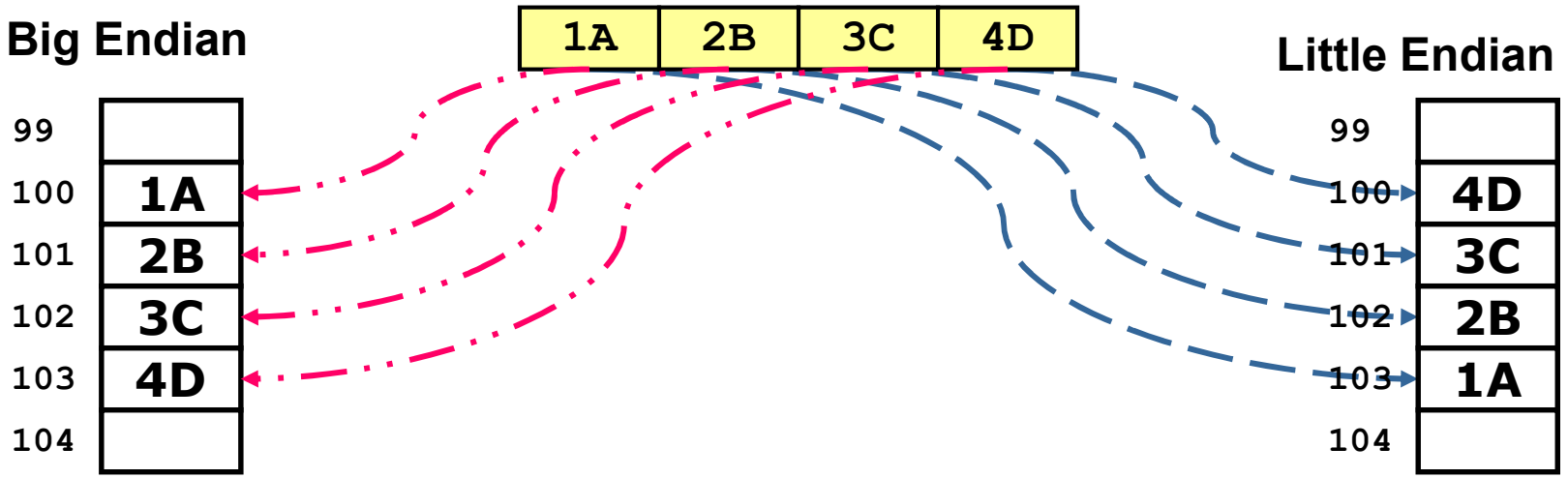
- Acesso de leitura
  1. Colocar um endereço nas linhas de endereço
  2. Activar as linhas de controlo para requerer uma operação de leitura
  3. Esperar que a memória retorne o valor e que o coloque no bus de dados (pode requerer ciclos de espera)
  4. Ler o valor do bus de dados
  5. Desactivar a linha de controlo para permitir que outro acesso inicie
- No Pentium cada acesso a memória gasta 3 ciclos de relógio
  - ✓ Ciclo 1 – passos 1 e 2
  - ✓ Ciclo 2 – passo 3
  - ✓ Ciclo 3 – passos 4 e 5

Memória - Acesso de escrita

- Acesso de escrita
  1. Colocar um endereço nas linhas de endereço
  2. Colocar um valor nas linhas de dados
  3. Activar as linhas de controlo para requerer uma operação de escrita
  4. Esperar que a memória escreva o valor (pode requerer ciclos de espera)
  5. Desactivar a linha de controlo para permitir que outro acesso inicie
- No Pentium cada acesso a memória gasta 3 ciclos de relógio
  - ✓ Ciclo 1 – passos 1 e 3
  - ✓ Ciclo 2 – passo 2
  - ✓ Ciclo 3 – passos 4 e 5

Memória - Ordem dos bytes

- Como é que os bytes que compõem um registo são guardados em memória endereçada ao byte?
- 2 convenções:
  - ✓ Exemplo colocar o valor $1A2B3C4D_{16}$ (43904110110) na posição de memória 100



Memória - Ordem dos bytes

- Em ambos os casos refere-se sempre os dados através do endereço mais baixo (100)

- Exemplos de arquitecturas little-endian
  - ✓ IA-32 e AMD64/Intel64

- Exemplos de arquitecturas big-endian
  - ✓ Configuração por omissão no MIPS e PowerPC

- Algumas arquitecturas suportam configuração
  - ✓ MIPS, Power, IA64, …

Memória - Ordem dos bytes

- Exemplo de como funciona uma transferência de dados entre big endian e little endian.
- A falta de standard no ordenamento de bytes é um dos maiores problemas na transmissão de dados.
- Suponhamos que temos em memória a string “Ola” e o número 315 em 32 bits:
  - ✓ A representação em big endian é:

‘O’	‘l’	‘a’	‘\0’
0	0	1	59

  - ✓ Em little endian é:

‘O’	‘l’	‘a’	‘\0’
59	1	0	0

  - ✓ Onde ‘O’ = 4F, ‘l’ = 6C, ‘a’ = 61 ‘\0’ = 00

Memória - Ordem dos bytes

- Como é que enviamos estes dados da primeira máquina (big endian) para a segunda (little endian)?
  - ✓ Sem tratamento (raw)?

‘0’	‘1’	‘a’	‘\0’
0	0	1	59

✖

    - ✓ Os 32 bits 0 0 1 59 não representam o número 315 em little endian, logo temos de ter algum tipo de tratamento
  - ✓ Inverter os bytes?

‘\0’	‘a’	‘1’	‘0’
59	1	0	0

✖

    - ✓ Apesar de o número ter sido bem recebido, se lermos os primeiros 4 bytes não obtermos a string “Ola”
  - ✓ É necessário anotar os dados com informação de tipo
    - ✓ Uma solução simples (mas ineficiente) é colocar um cabeçalho de descrição (tipo e comprimento) em cada transmissão

Memória - Ordem dos bytes

- Nesse caso a informação enviada é a seguinte:

  char 1 byte - 'O'
  char 1 byte - 'l'
  char 1 byte - 'a'
  char 1 byte - '\0'
  int 4 bytes - 0 0 1 59

- O receptor converte os dados para a sua representação interna:

'O'	'l'	'a'	'\0'
59	1	0	0

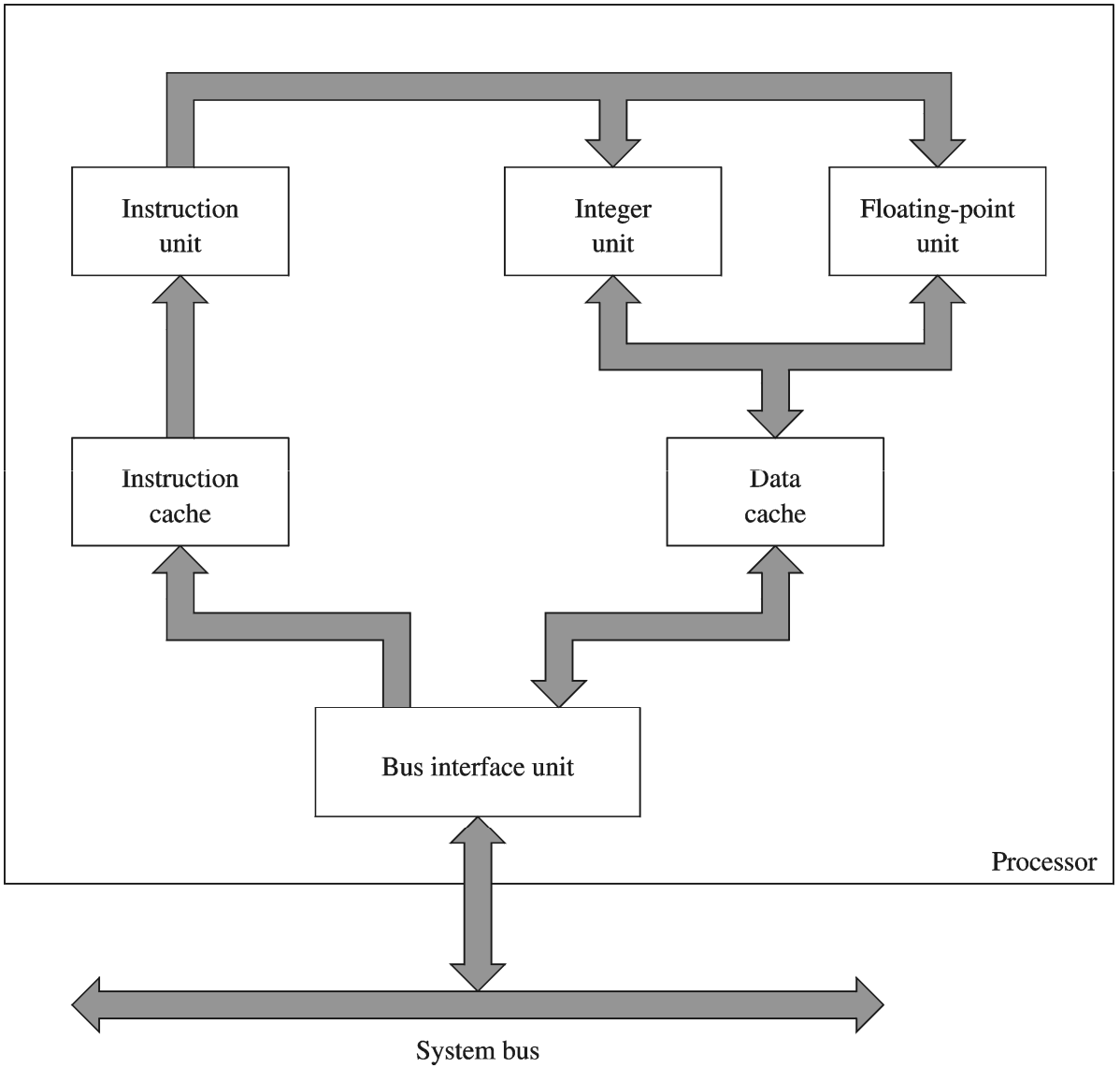
- Este tipo de mecanismo também resolve os problemas das diferentes representações para o mesmo tipo
  - ✓ Exemplo: **int** a 32 bits e **int** a 16 bits
- A comunicação entre máquinas heterogêneas será estudado a fundo em *Redes de Computadores*

Memória – Questões de Desenho

- As memórias são lentas
- Como colmatar o fosso entre a velocidade dos processadores e da memória?

- A solução passa por uma hierarquia de memórias – as memórias cache
  - ✓ Usar pequenas porções de memória rápida (mas cara)
  - ✓ Aparentar que estamos sempre a utilizar essa memória rápida
  - ✓ Fazer uso do conceito de localidade
  - ✓ E funciona!!!!
    - ✓ Consegue-se ter taxas de acerto de mais de 90% nas memórias cache

Memória – Questões de Desenho

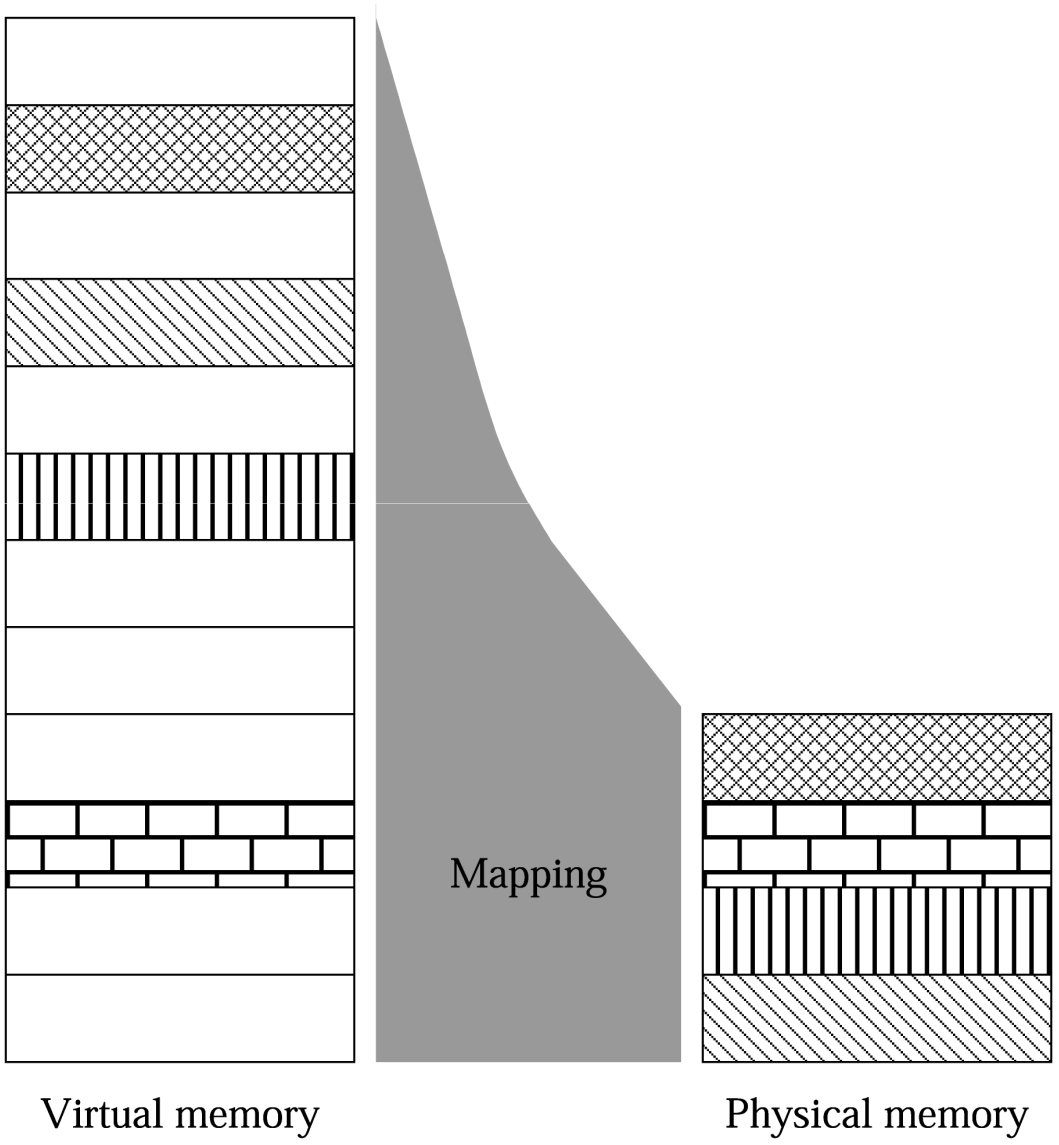


Memória – Questões de Desenho

- A memória instalada é normalmente insuficiente para suportar a execução do sistema operativo e de todas as aplicações
- De que é que serve poder endereçar 4GBytes de memória ou mais se só tenho 1GByte instalado?

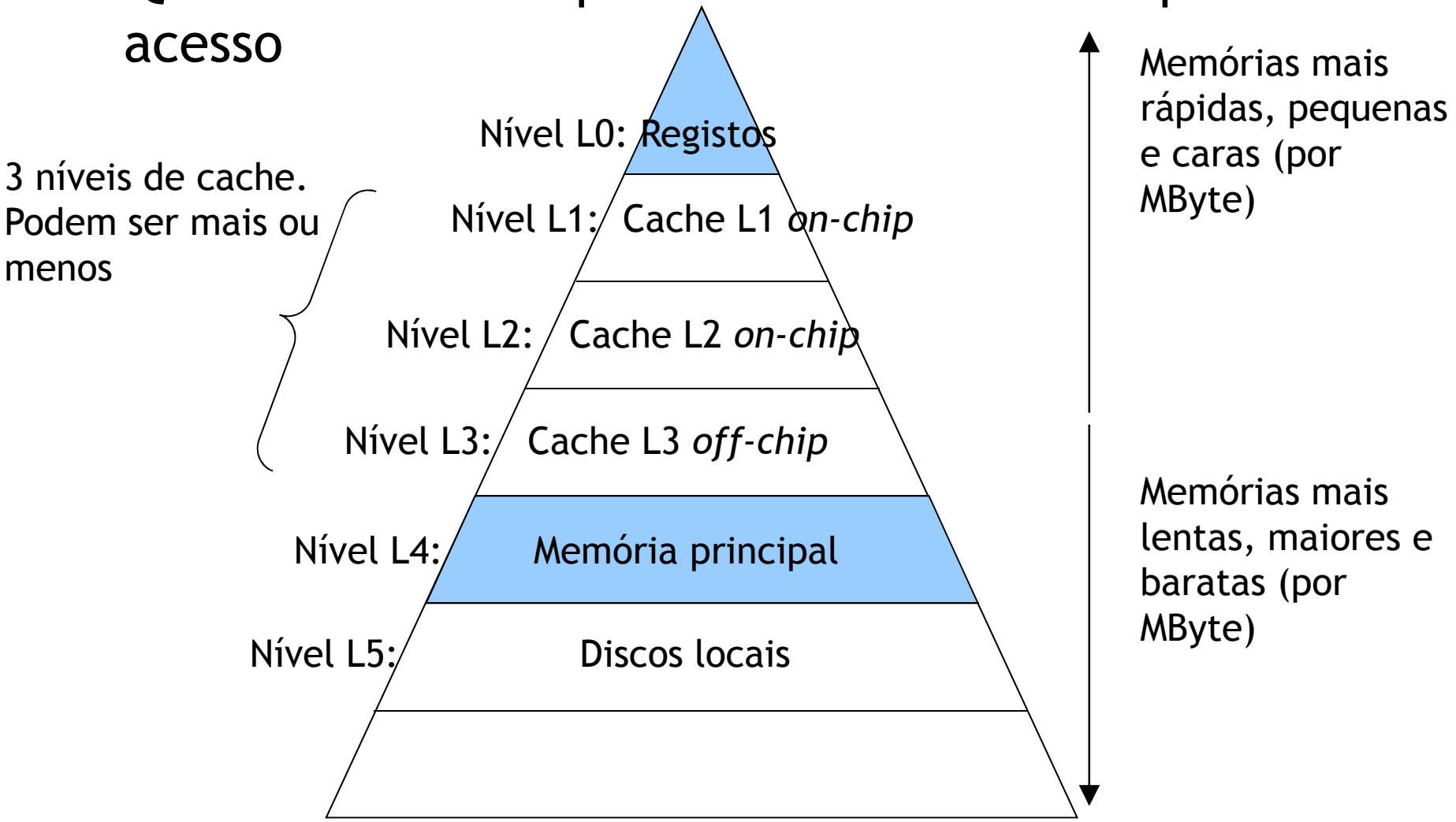
- A solução passa pela noção de memória virtual
  - ✓ Aparentar que se tem muito mais memória do que a que está na realidade instalada
  - ✓ O disco é utilizado para guardar os dados que não estão a ser utilizados naquele momento
  - ✓ A memória central passa a funcionar como uma cache para o disco

Memória – Questões de Desenho



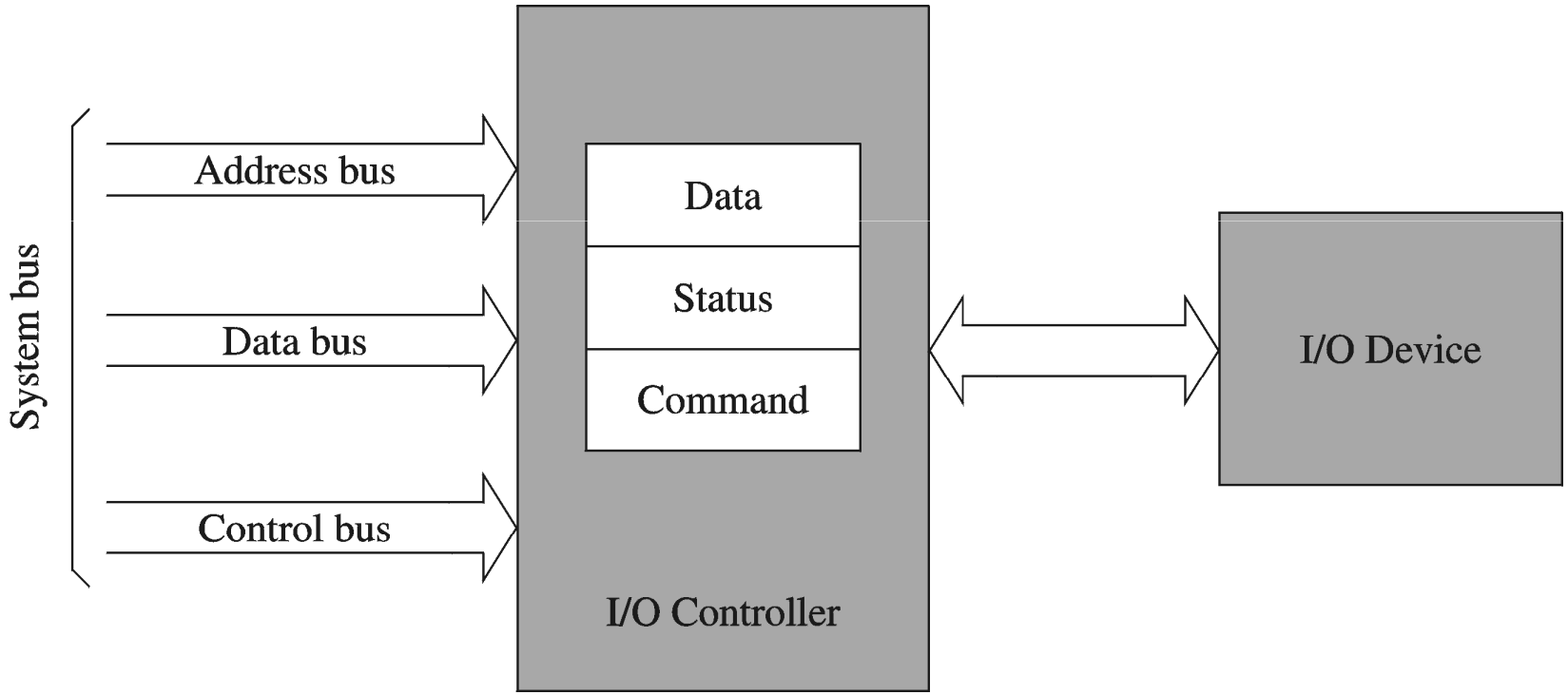
Memória - Hierarquia

- Quanto maior é capacidade maior é o tempo de acesso



Entradas/Saídas (Input/Output)

- Os dispositívos de I/O são acedidos via um controlador que gere os detalhes de baixo nível



Input/Output - Controlador genérico

- Em geral, as interfaces têm três tipos de registos (portos ou portas):
  - ✓ Porta de dados: para guardar os dados enviados pelo CPU ou recebidos do exterior pelo dispositivo periférico
  - ✓ Porta de estado: indica informação sobre o estado corrente do periférico
    - ✓ Existência de dados para ler
    - ✓ Disponível para receber dados
    - ✓ Erros e outras informações
  - ✓ Porta de controlo: para receber comandos vindos do CPU (inicializar interface, posicionar a interface em estados para receber ou enviar, e outros comandos que desencadeiam a operação do periférico)

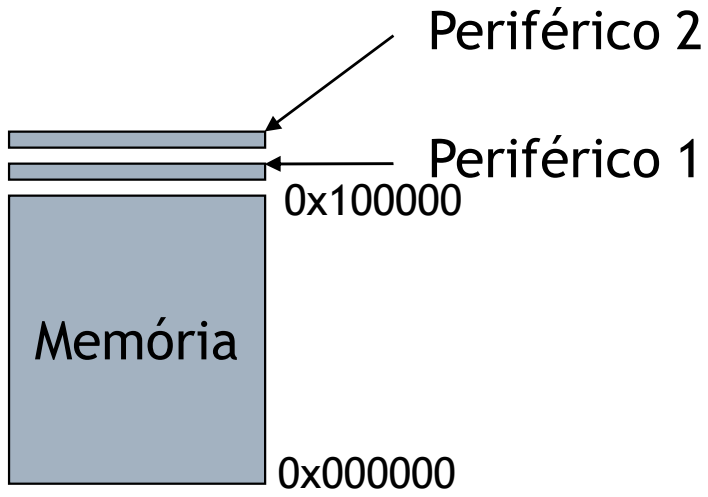
Input/Output – Endereçar I/O

- Existem duas formas para se endereçar o destinatário de uma operação de I/O
  - ✓ Mapeamento em memória – Memory Mapped I/O
    - ✓ Usado pela maior parte dos processadores
  - ✓ Espaço de endereçamento próprio
    - ✓ Usado pelo IA-32
    - ✓ O Pentium também suporta Memory Mapped I/O

Input/Output – Memory Mapped I/O

- Uma faixa de endereços corresponde à memória
- Cada controlador é configurado para responder a uma faixa de endereços fora do espaço da memória e a que nenhum outro controlador responde
- As faixas de endereços da memória e dos controladores são disjuntos

Entidade	Faixa de endereços
Memória	0x000000-0x0FFFFF
Periférico 1	0x100000-0x10000B
Periférico 2	0x10000C-0x100017



Input/Output – Memory Mapped I/O

- Espaço único
  - ✓ Não há instruções máquina especiais para acesso aos controladores.
  - ✓ São usadas as instruções usuais de movimentação de dados (por exemplo **mov** no IA-32) para endereços que pertencem à faixa de endereços de entrada/saída

- Exemplo:
  - ✓ Envio de 1 byte — OUT — para o periférico 2:

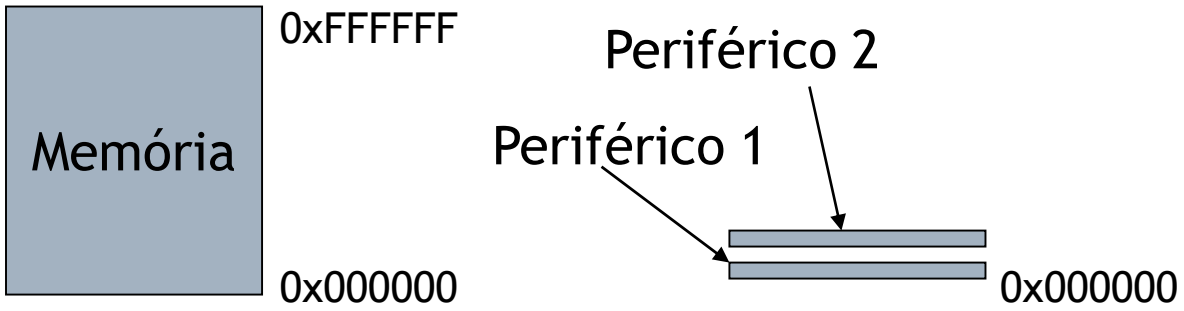
```
mov byte [0x10000C], 5
```

Input/Output – Espaço próprio

- Dois espaços de endereçamento separados
  - ✓ Um para a memória
  - ✓ Um para os controladores dos dispositivos
- Cada controlador é configurado para responder a uma faixa de endereços diferente
- A faixa de endereços da memória é sempre disjunta da dos periféricos
- O CPU necessita de instruções diferentes para separar os acessos a memória dos acessos aos periféricos

Input/Output – Espaço próprio

Entidade	Faixa de endereços
Memória	0x000000-0xFFFFFF
Periférico 1	0x000000-0x00000B
Periférico 2	0x00000C-0x000017



Input/Output – Espaço próprio

- Espaço separado
  - ✓ Há instruções máquina dedicadas ao acesso do espaço de endereços de entrada/saída (*ports*)
  - ✓ Exemplo da família Intel 80x86/Pentium
    - ✓ **IN** regA, port (leitura para o registo EAX/AX/AL)
    - ✓ **OUT** port, regA (escrita a partir do conteúdo do EAX/…)
    - ✓ Se port > 255 então usa-se um registo:

      IN regA, DX e **OUT** DX, regA

- O mesmo exemplo do acetato anterior:

```
mov al, 5
out 0x0C,al
```

Input/Output - Programação

- Existem vários métodos para programar um controlador de I/O:
  - ✓ Espera activa (programmed I/O ou polled I/O)
    - ✓ O programa utiliza um ciclo de espera activa (que gasta tempo de CPU) para ler os dados do controlador
    - ✓ A transferência tem de ser antecipada
  - ✓ Por interrupções (interrupt-driven I/O)
    - ✓ O controlador é programado para gerar uma interrupção (notificação) quando um certo, pré-definido evento ocorre
    - ✓ Lida com transferências não antecipadas
  - ✓ DMA (Direct Memory Access)
    - ✓ Permite transferir blocos de dados entre memória e periféricos sem intervenção do CPU

Interconexão - Bus

- Um Bus é um mecanismo de comunicação digital que permite a duas ou mais unidades funcionais transferir dados e sinais de controle
- Um computador pode conter vários “buses” que estão optimizados para um determinado objectivo
  - ✓ On-chip
    - ✓ Ligação ALU – registos
    - ✓ Bus de dados e endereços para ligar a memórias cache on-chip
  - ✓ Internos
    - ✓ PCI, AGP, PCMCIA
  - ✓ Externos
    - ✓ USB, IEEE 1394 (FireWire), série, …

Interconexão - Bus

- "Buses" são partilhados por múltiplas entidades
  - CPUs e periféricos, múltiplos CPUs, ...
- A norma do bus define um protocolo de acesso:
  - Transacções:
    - Sequência de operações para realizar uma dada actividade
      - Leitura/escrita de/para memória ou para um periférico
    - Exemplo de uma transacção
      - Pentium burst read – lê 4 posições de memória
  - Arbitragem:
    - Determina qual a entidade que pode usar o Bus num dado momento
  - Todos as entidades ligadas ao Bus têm de obedecer a este protocolo

Avanços Tecnológicos - CPU