Arquitetura de Computadores

MiEI – 2016/17

DI-FCT/UNL

Aula 3

C vs Java

Java	**C**
object-oriented	function-oriented
strongly-typed	can be overridden
method overloading	no function overloading
classes for name space	single name space (mostly), file-oriented
exceptions, exception handling	if (f() < 0) {error} or OS signals
String is a type	Just char [], with '\0' end
Standard classes lib	Standard functions lib

C vs Java

- A execução começa sempre função main()
- A definição de funções é semelhante à dos métodos, mas:
  - Não são de instâncias de objetos
  - Não é preciso criar nada para usar, nem existe `this` (São como os métodos `static`)
- Retorno de função: `return;` ou `return val;`
- Blocos de código e scope:

```
{ declarações;
  instruções;
}
```


C vs Java

- Construções de controlo:

```
if (exp) {  inst;
} else {  inst;
}

while ( exp ) {
  inst;
}

for ( inst; exp; inst ) {
  inst;
}
```

```
switch (expr) {
  case val: inst; break;
  ....
  default: inst;
}


do {
   inst;
} while (exp);
```

C tem pre-processador

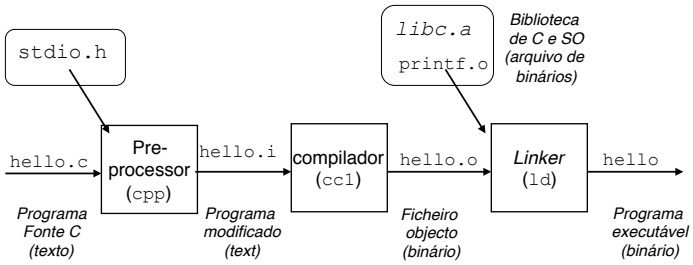
- Diretivas iniciadas por #
  - #define XPTO 234 - no ficheiro, substitui XPTO por 234
  - Permite pseudo-constantes, e mais...
  - #include “file.h” - inclui o file.h da diretoria corrente
  - #include <file.h> - inclui o file.h, da diretoria *standard*
  - Permite introduzir as definições de funções e tipos (interfaces) fora do nosso ficheiro como das bibliotecas
  - Também chamados ficheiros de *headers*

Processo de compilação de C

- Exemplo: hello.c
- compilação do programa:

`cc –o hello hello.c`

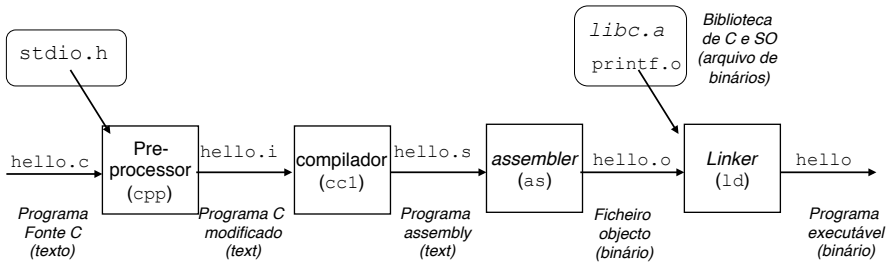
```
#include <stdio.h>
int main() {
   printf(”Hello world!\n”);
   return 0;
}
```



- compilação do programa:

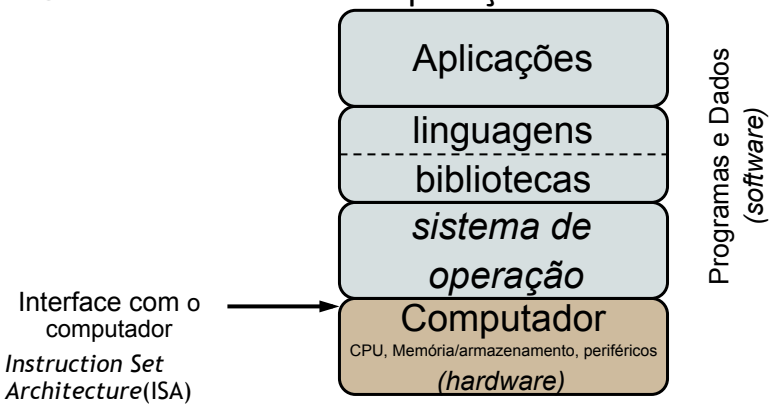
```
cc –o hello hello.c
```

- Alguns compiladores podem ter mais passos intermédios:



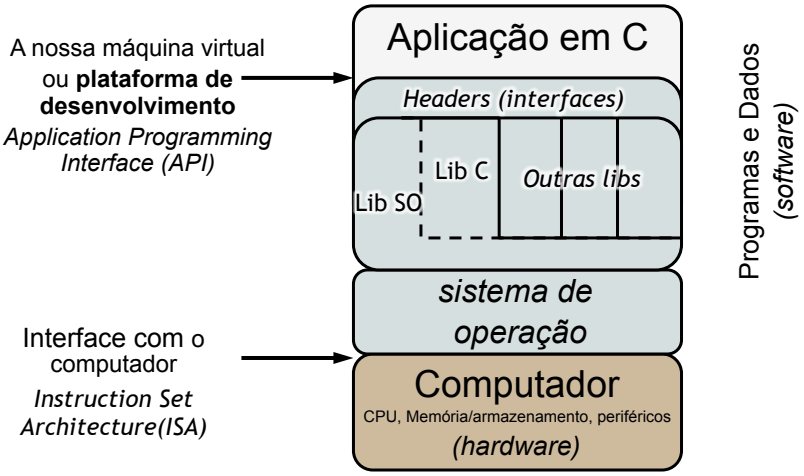
Níveis nos sistemas informáticos

- Num sistema de computação:



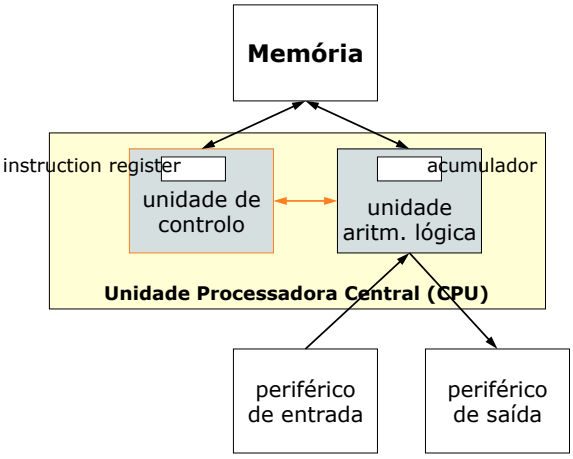
Níveis nos sistemas informáticos

- Num sistema de computação com C:



Arquitectura de Von Neumann (década 1940)

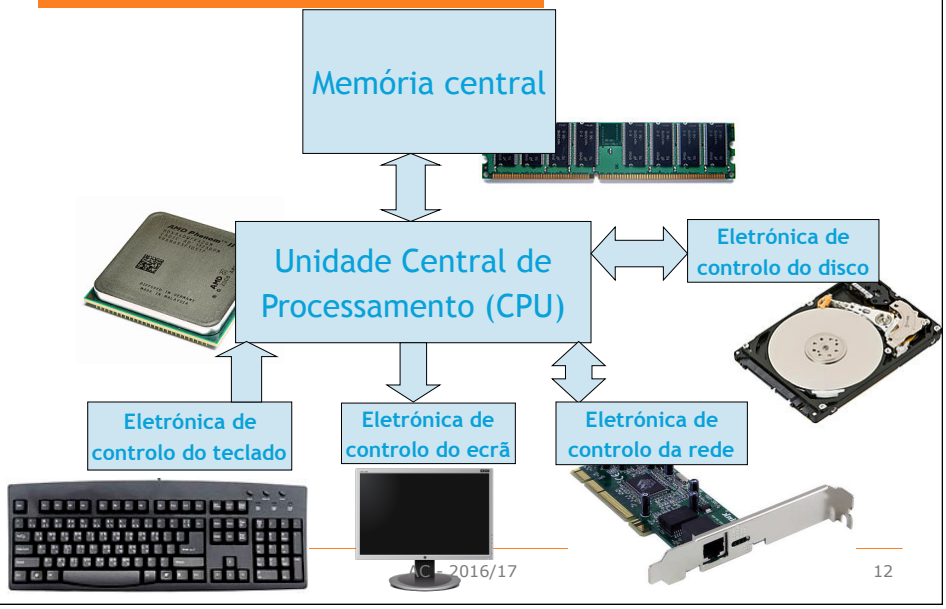
versão original



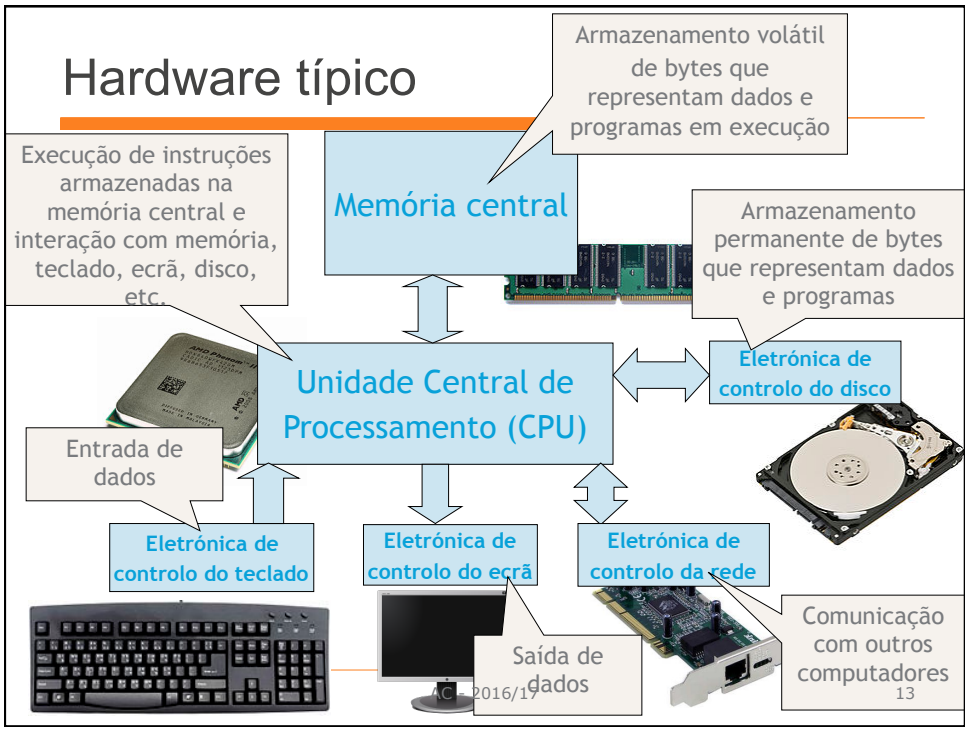
Como guardar um inteiro ou um float na memória?

Como os transferir entre unidades e entre periféricos?

Hardware típico



Hardware típico

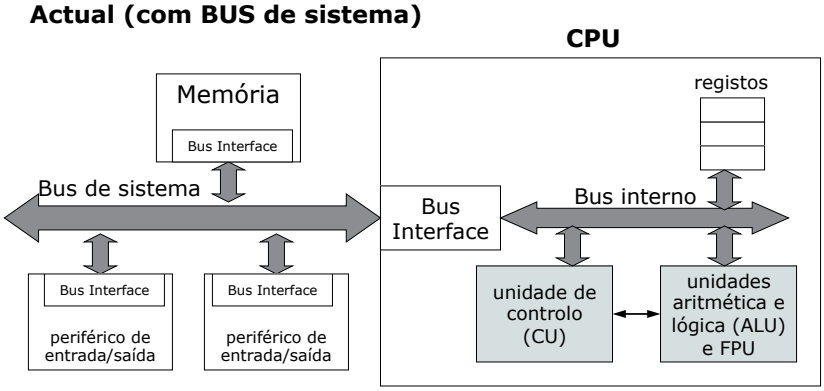


Ligações entre componentes

- Como os componentes comunicam?
  - componentes internos do CPU; componentes externos e CPU
- Quantas ligações?
  - Tantas quantos os bits a transportar de cada vez
- Cada componente ligado a todos os outros?
  - Complexo e dispendioso!
- Os componentes partilham um meio único: o BUS
  - Seguem um protocolo para partilhar o BUS

Arquitectura de Von Neumann (2)

Actual (com BUS de sistema)

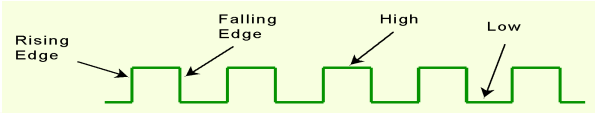


Os componentes têm diferentes ritmos de funcionamento

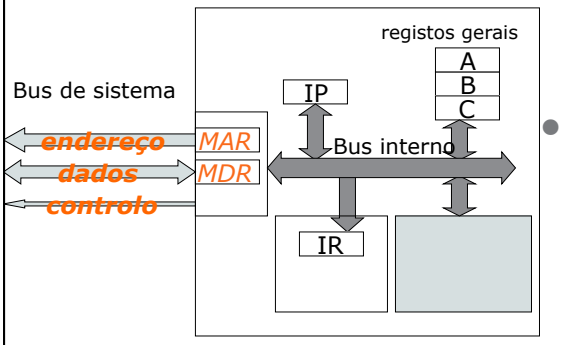
Diferentes relógios (clocks)

Relógio nos Circuitos Síncronos

- Os circuitos funcionam ao ritmo de um relógio:
  - marca com impulsos eléctricos o ritmo de funcionamento de todos os componentes (transições de estado)
  - Podem existir diferentes relógios
- Cada relógio gera um determinado número de impulsos por segundo: ciclos/segundo ou frequência (Hz)
- Marcam quando os componentes podem interactuar
  - Garante que os vários sinais (bits) são enviados e recebidos no instante devido
    - Ex: um componente não lê o que está no BUS antes que algo válido lá tenha sido colocado



Interface com o bus

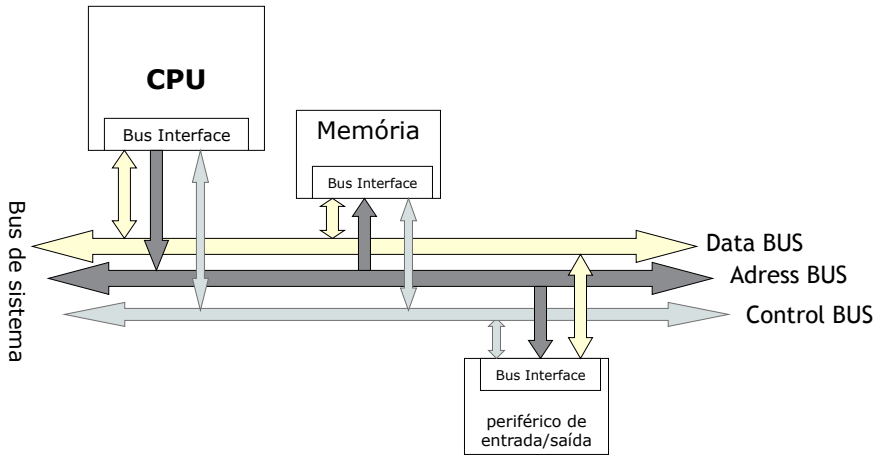


- O CPU tem mais do que é visível ao nível do código da máquina (*Instruction Set Architecture* -ISA)
- Mais registos e Micro Acções internas
  - exemplo:

    Store Register to memory:
    MAR ← endereço mem.
    MDR ← conteúdo do reg.
    controlo ← escr. memória

  - **RTN/RTL *Register Transfer Notation/Language***

BUS de sistema



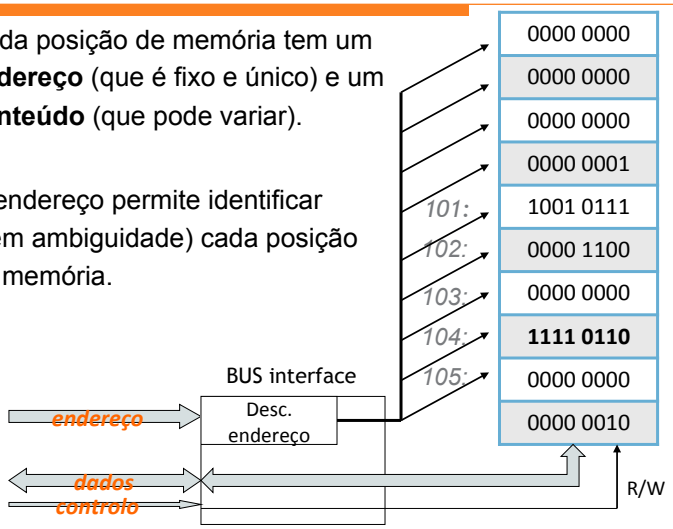
Bus de sistema

- Conjunto de linhas paralelas, cada uma codificando um bit
  - Número de linhas define a **largura** do Bus
  - 1 linha para dados → ligação **série**
- Bus de endereços:
  - conjunto de linhas que codificam o interlocutor
  - para identificar a célula de memória ou a unidade periférica
- Bus de dados:
  - codificam os dados a transferir
- Bus de controlo:
  - para coordenar as transferências e as interacções entre unidades,
  - sinais de controlo e comandos (por exemplo: ler/escrever, memória/periférico)

Memória central (RAM): endereços e conteúdos

Cada posição de memória tem um **endereço** (que é fixo e único) e um **conteúdo** (que pode variar).

O endereço permite identificar (sem ambiguidade) cada posição da memória.



O conteúdo da posição de memória com o endereço 104 é **11110110**

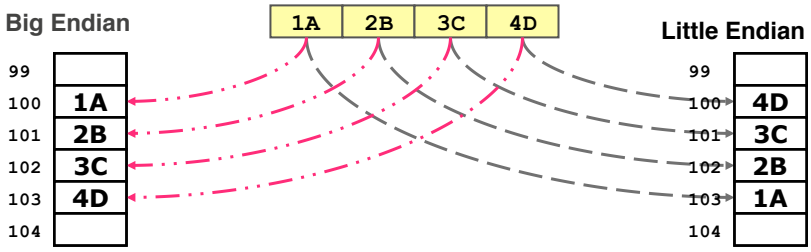
Memória central

- Guarda instruções e dados, sob a mesma representação simbólica, em células que contêm agrupamentos de bits:
  - 1 bit, byte (8 bits), word (palavras) de outra dimensão …
- É acedida como um vetor:
  - Mem[0]…Mem[n-1] (capacidade = n palavras)
  - 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 **direto**: dá-se i para aceder a Mem[i]

    → *RAM: Random Access Memory*

Ordem dos bytes das palavras na memória

- Como é que os bytes que compõem um registo são guardados em memória endereçada ao byte?
- 2 convenções:
  - Exemplo colocar o valor $1A2B3C4D_{16}$ (439041101_{10}) na posição de memória 100:



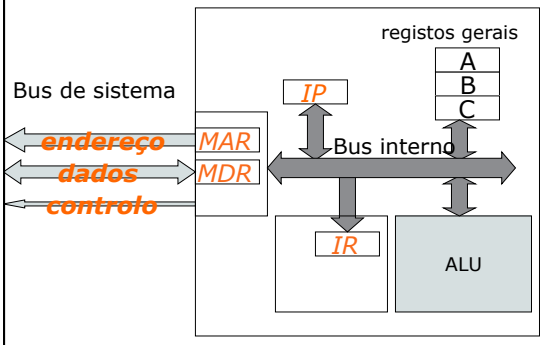
Palavra vs byte, Little vs Big

- Na memória endereçada ao byte
  - O endereço indica o 1º byte da palavra
  - O acesso a qualquer byte pode levar-nos ao "meio" de uma palavra
- A ordem dos bytes (*endianness*) é particularmente importante quando se partilha informação:
  - Os ficheiros são organizados como sequências de bytes
  - Transferir dados por interfaces com o exterior (eg. Rede)

Processador - CPU

- Controlo global das operações do computador e responsável pela interpretação 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*
  - Possivelmente uma FPU – *Floating Point Unit*
  - Conjunto de registos (ou registadores): células de memória locais ao CPU, a usar pelas instruções e para manter valores intermédios

Micro-acções: fetch



ciclo de funcionamento:
fetch (obter instrução da memória)
decode (descodifica)
execute (executa)

fetch: **IR = mem[IP]**
MAR ← IP
controlo ← lêr memória
MDR ← Mem[MAR]
IR ← MDR

Nota: IP também é chamado PC

Ciclo de Execução

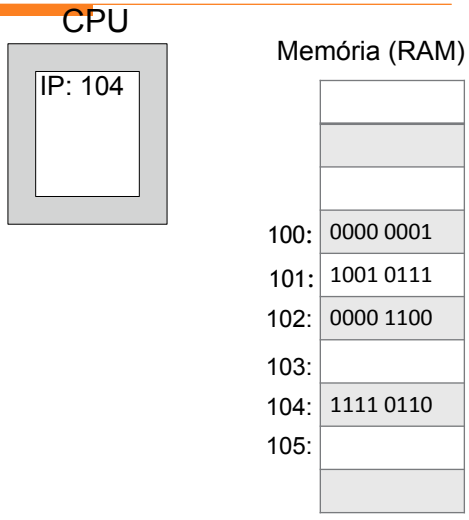
- A Unidade de controlo do CPU:
  - Obtém a próxima instrução de memória (usa um índice para o programa: PC ou IP) → ***fetch***
  - Incrementa IP (IP← IP + 1)
  - Descodifica a instrução → ***decode***
  - Emite os sinais de controlo, na micro-arquitectura, correspondentes ao encadeamento de acções necessárias para executar a instrução e as transferências de informação necessárias → ***execute***
  - No fim, volta ao início

Execução

- A execução de uma instrução pode envolver:
  - Operações aritméticas e lógicas (pela ALU) ou FP
    - A ALU e FPU operam sobre operandos na instrução ou em registos
    - ADD, OR, NOT, ...
  - Transferências reg/CPU ↔ Memória
    - Load/Store ou MOV
  - Transferências reg/CPU ↔ Periféricos
    - IN/OUT
  - Controlo da sequência de execução de instruções → **alterar o valor em IP**
    - Jump

Funcionamento do CPU

- O CPU executa sequencialmente as instruções guardadas na memória central
- Em cada momento, o CPU mantém a posição de memória da instrução a executar
  - ex: 104

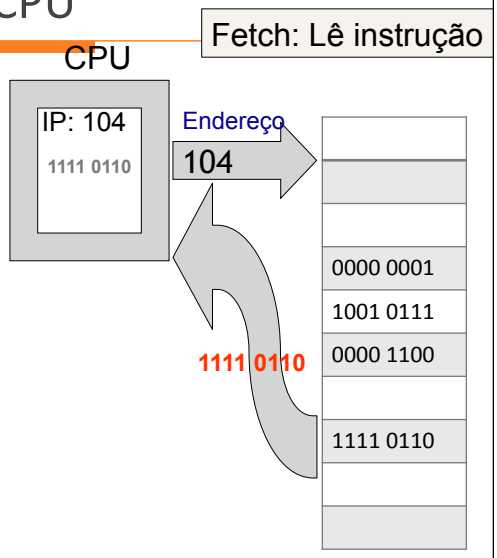


Funcionamento do CPU

- A instrução define a ação elementar a executar
  - Ações atuam sobre dados em registos, memória central ou num dispositivo de entrada/ saída
- Exemplo:

  somar 100 101 102

  Soma o conteúdo das posições 100 e 101 e armazena o resultado na posição 102

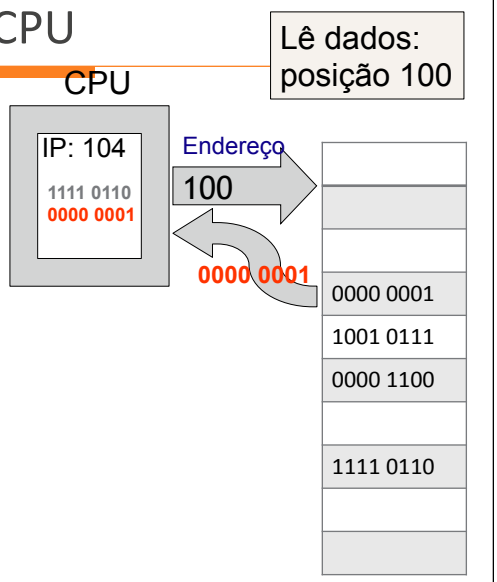


Funcionamento do CPU

- A instrução define a ação elementar a executar
  - Ações atuam sobre dados em registos, memória central ou num dispositivo de entrada/ saída
- Exemplo:

  somar 100 101 102

  Soma o conteúdo das posições 100 e 101 e armazena o resultado na posição 102

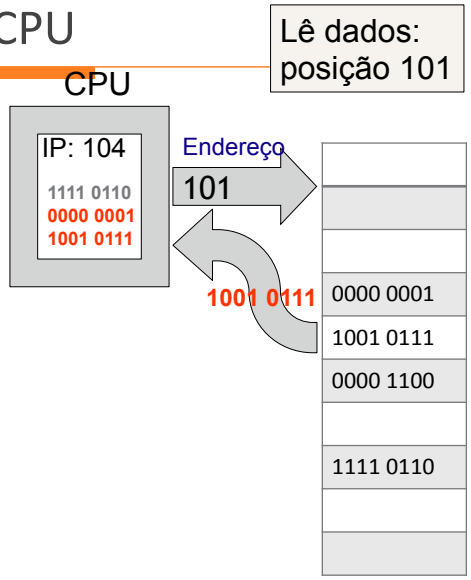


Funcionamento do CPU

- A instrução define a ação elementar a executar
  - Ações atuam sobre dados em registos, memória central ou num dispositivo de entrada/ saída
- Exemplo:

  somar 100 101 102

  Soma o conteúdo das posições 100 e 101 e armazena o resultado na posição 102

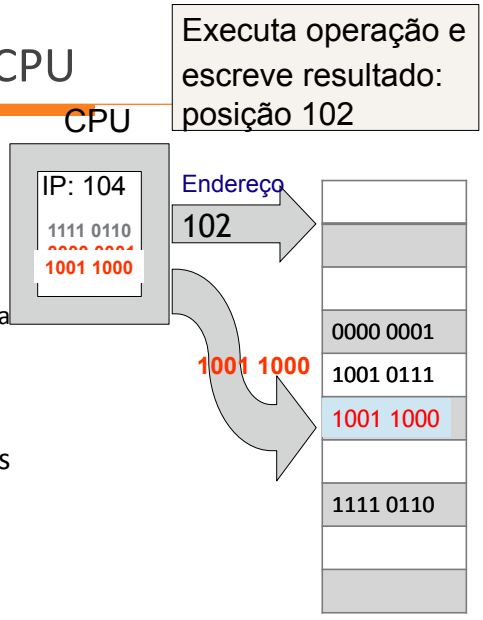


Funcionamento do CPU

- A instrução define a ação elementar a executar
  - Ações atuam sobre dados em registos, memória central ou num dispositivo de entrada/saída
- Exemplo:

  somar 100 101 102

  Soma o conteúdo das posições 100 e 101 e armazena o resultado na posição 102
- Este tipo de instrução não é real



Little Man Computer

- Instruções de tamanho fixo (3 algarismos), com zero ou um operando em memória (XX = endereço de memória)

Code	Name/assembly	Description
000	HLT	Halt/Stop
1XX	ADD	Accumulator = Accumulator+Operand
2XX	SUB	Accumulator = Accumulator-Operand
3XX	STA	Store Accumulator in the memory
5XX	LDA	Load the Accumulator from memory
... Etc ...		

- Exemplos:
  - http://peterhigginson.co.uk/lmc/
  - http://pddring.github.io/cpu-battle-tank/

Relógio vs velocidade do computador

- A frequência do relógio não é o único factor na velocidade de funcionamento de um computador!
- O tempo de execução de determinado programa depende de:
  - Número de instruções (tamanho do programa)
  - O que cada instrução faz
  - Tempo de execução de cada instrução (núm. ciclos de relógio)
  - Tempos de espera pela memória
  - Tempos de espera pelas Entradas/Saídas
  - E se estivermos a executar outros programas?

Tempos típicos

- CPU: cada instrução → 1, 2, … ciclos de relógio
- Memória: cada acesso → dezenas de ciclos
- Periféricos: cada E/S → muitas dezenas, centenas, ou milhares, de ciclos

Vendo noutras escalas

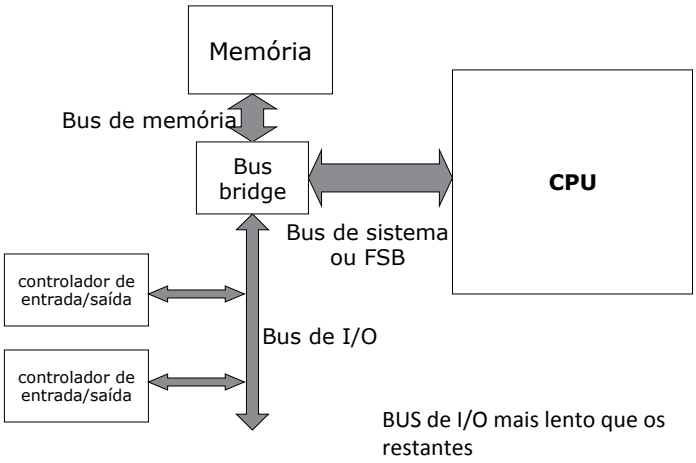
- Exemplo *clock*: 1GHz → 1 000 000 000 ciclos/s
  - Admitindo a execução de 1 instrução p/ciclo (1ns)

	escala: 1ns → 1s	escala: 1ns → 1m
Registo: 1 ns	1 s	1 m
Memória RAM: 20 ns	20 s	20 m
Disco: 8 ms – 20 ms	3 a 7,6 meses	8 000 – 20 000 km

1 ms = 1 000 000 ns

Arquitecturas de computadores

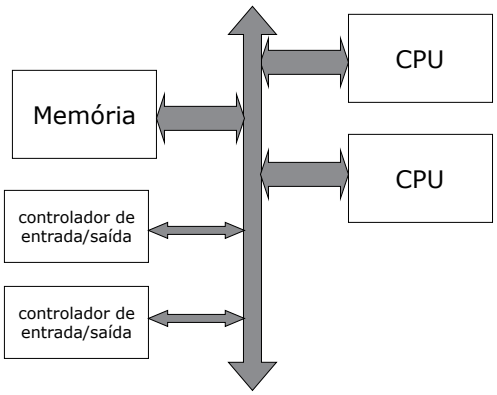
variantes



BUS de I/O mais lento que os restantes

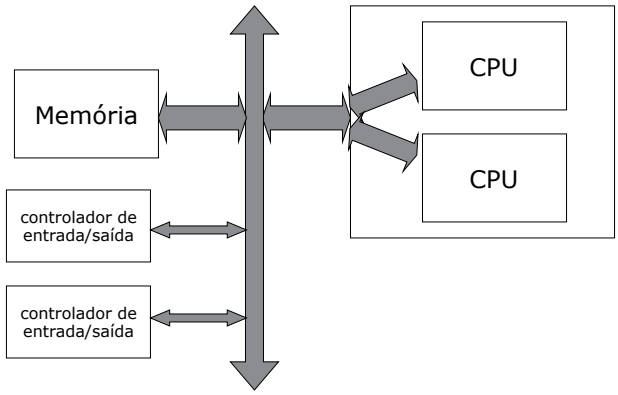
Arquitecturas de computadores (2)

variantes



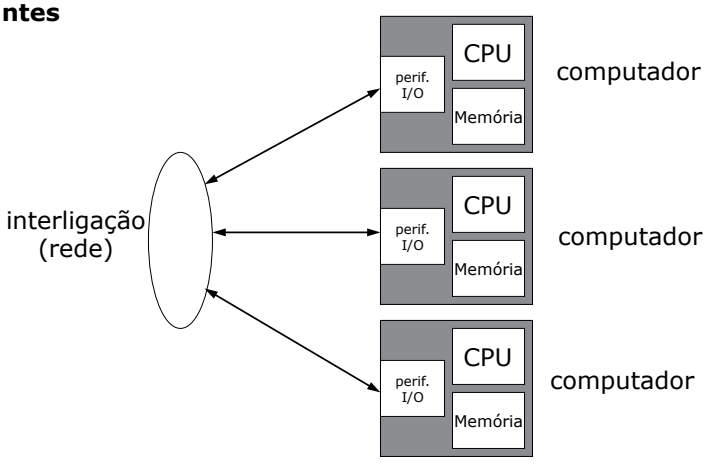
Arquitecturas de computadores (2)

variantes



Arquitecturas de computadores (3)

variantes



Arquitectura de Von Neumann (resumo)

- 3 tipos de componentes (interligados através de buses):
  - Processador central (CPU - Central Processing Unit)
  - Memória central
  - Unidades de Entrada/Saída (para periféricos de armazenamento e comunicação)
- O CPU suporta vários tipos de instruções:
  - Aritméticas e lógicas
  - Transferências entre componentes
  - Controlo da sequência de execução das instruções (“saltos”)
- Um modelo de execução muito simples:
  - A Memória armazena as instruções e os dados
  - Um acesso a Memória de cada vez
  - Interpreta uma instrução de cada vez
  - Um programa é uma sequência de instruções que vão mudando o estado dos componentes