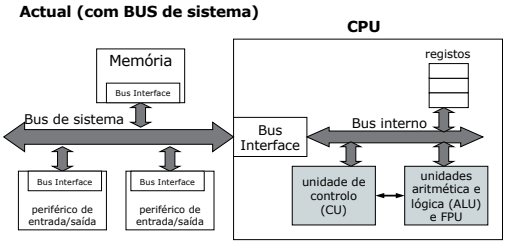
Arquitetura de Computadores

MIEI – 2017/18
DI-FCT/UNL
Aula 7

Arquitectura de Von Neumann (2)

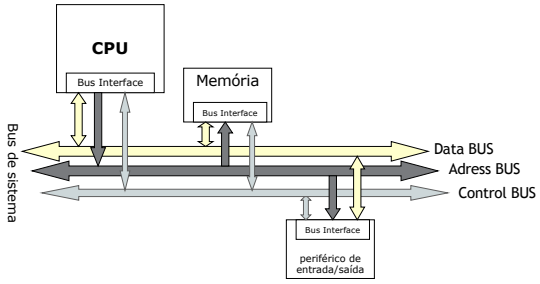


Os componentes têm diferentes ritmos de funcionamento
Diferentes relógios (clocks)

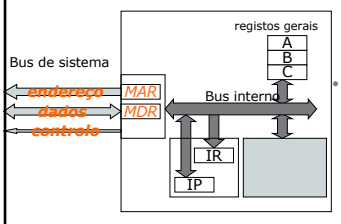
Processador - CPU

- 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

BUS de sistema



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

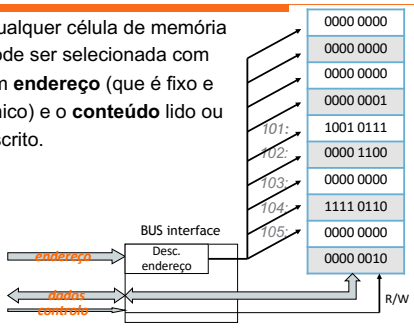
RTN/RTL *Register Transfer Notation/Language*

Bus de sistema

- 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)

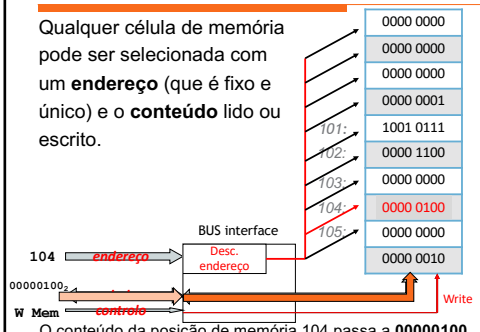
Memória central (RAM): endereços e conteúdos

Qualquer célula de memória pode ser selecionada com um **endereço** (que é fixo e único) e o **conteúdo** lido ou escrito.



Memória central (RAM): endereços e conteúdos

Qualquer célula de memória pode ser selecionada com um **endereço** (que é fixo e único) e o **conteúdo** lido ou escrito.



O conteúdo da posição de memória 104 passa a **00000100_2**

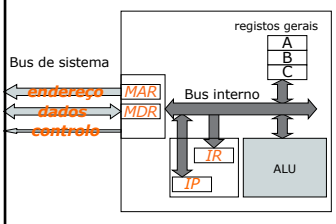
Memória central

- Guarda grupos de bits que representam instruções ou dados
  - Cada célula pode ser um byte (8 bits), uma word ("palavra"), outra dimensão …
- É acedida como um vetor:
  - Mem[0]…Mem[n-1] (capacidade = n células)
  - O endereço da célula corresponde ao índice i em Mem[i]
  - O endereço é representado em binário, como um número inteiro sem signal
  - Pode ser lido/escrito em grupos de células
  - O acesso é **direto**: dá-se i para aceder a partir de Mem[i]
    - →*RAM: Random Access Memory*

CPU: 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

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

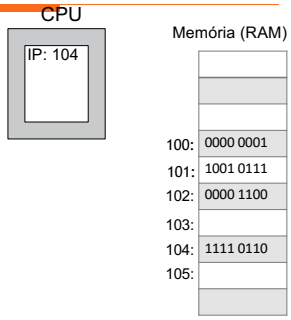
Nota: IP também é chamado PC

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

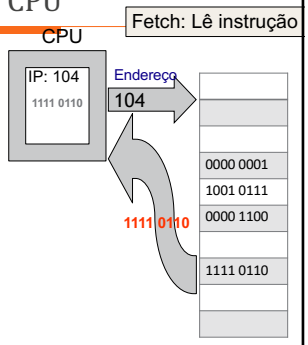


Funcionamento do CPU

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

```
SOMA 100, 101, 102
```

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

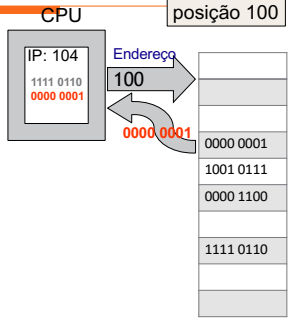


Funcionamento do CPU

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

```
SOMA 100, 101, 102
```

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

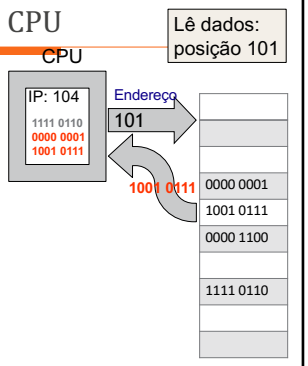


Funcionamento do CPU

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

```
SOMA 100, 101, 102
```

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



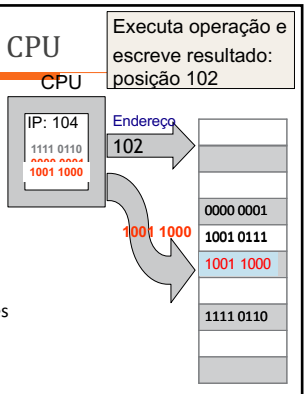
Funcionamento do CPU

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

```
SOMA 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). Um registo geral (acumulador)

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

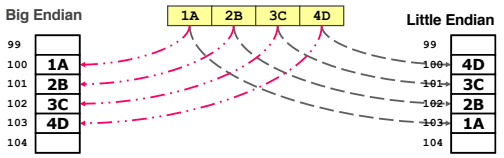
- Exemplo *clock*: 1GHz → 1 000 000 000 ciclos/s
  - Admitindo a execução de 1 instrução p/ciclo (1ns)

	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

Memória: Ordem dos bytes das palavras

- 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
  - Os acessos a endereços alinhados com as palavras é normalmente vantajoso
- A ordem dos bytes (*endianness*) é particularmente importante quando se partilha informação:
  - Ex: os ficheiros correspondem a sequências de bytes
  - Transferir dados por interfaces com o exterior (eg. Rede)

Variáveis na memória

- Variáveis são designações simbólicas para posições de memória:

```
int A = 5;
int B = -2;  // 0xFFFFFFFE
```

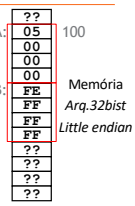


Tabela de símbolos do compilador:

A: 100 : 4 bytes

B: 104 : 4 bytes

Variáveis na memória

- Variáveis são designações simbólicas para posições de memória:

```
int A = 15213;   // 0x00003B6D
int B = -15213;
long C = 15213;
long long D = 15213;
```

Tabela de símbolos do compilador:

A: 100 : 4 bytes
B: 104 : 4 bytes
C: 108 : 4 bytes
D: 112 : 8 bytes