1º Semestre de 2007/2008

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira



Arquitectura de Computadores I 2007/08

## Aulas 2 & 3

- Definições e convenções
- Conceitos fundamentais em Arquitectura de Computadores
  - O Computador como sistema digital
  - Os elementos básicos de um computador
  - Modelos de Harvard e Von Neumann
  - O ciclo básico de execução de uma instrução
- Arquitectura de Computadores
  - *Instruction Set Architecture*
  - Organização
  - Níveis de Representação

- Algumas definições e convenções

| | Utilização normal | Utilização como potência de 2 |
|---|---|---|
| **K** (kilo) | $10^3$ =1.000 | $2^{10}$=1.024 |
| **M** (mega) | $10^6$ =1.000.000 | $2^{20}$=1.048.576 |
| **G** (giga) | $10^9$ =1.000.000.000 | $2^{30}$=1.073.741.824 |
| **T** (tera) | $10^{12}$=1.000.000.000.000 | $2^{40}$=1.099.511.627.776 |

  - Potências de 10: utilizadas, por ex., para expressar frequência, taxas de transferência
  - Potências de 2: usadas para expressar quantidade de informação; Ex.: capacidade de um dispositivo de armazenamento (memória, disco, ...)





- Algumas definições e convenções (cont.)

| | Europe | USA |
|---|---|---|
| *Million* | $10^6$ | $10^6$ |
| ***Billion*** | $\mathbf{10^{12}}$ | $\mathbf{10^9}$ |

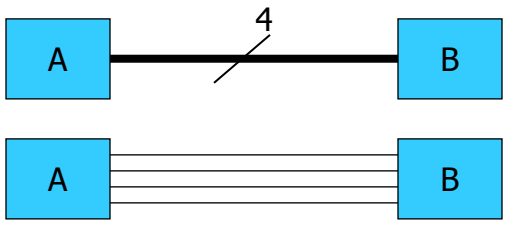
  - **byte**: grupo de 8 bits
  - **nibble**: ½ de 1 byte, i.e., 4 bits
  - **word**: grupo de bits que podem ser processados simultaneamente. Tipicamente: 16, 32, 64, 128 bits (no contexto do MIPS, 1 word = 32 bits)

- Algumas definições e convenções (cont.)
  - Representação de quantidades numéricas:
    - Binário: 10100011B, $10100011_2$
    - Hexadecimal: AF26H, 0xAF26
    - Decimal: 2004, $2004_{10}$
  - Unidades:
    - Segundos: s
    - Bits: b
      - (Ex. 1Mb/s ≡ 1 Mega bit/s)
    - Bytes: B
      - (Ex. 40MB ≡ 40 Mega bytes)
    - Words: w

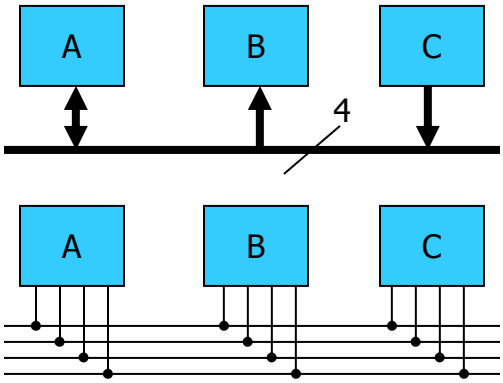




- Barramento *(bus)*= colecção de fios agrupados segundo uma dada função; cada fio transporta informação relativa a 1 bit. Ex. – barramento de 4 bits:



- Barramento partilhado (***shared bus***)= barramento que interliga diferentes blocos funcionais





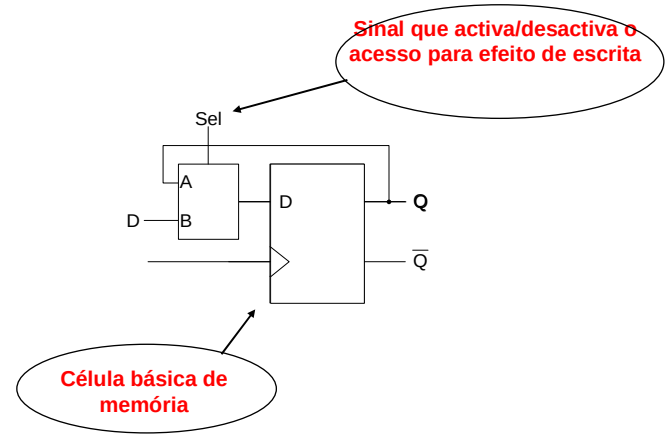


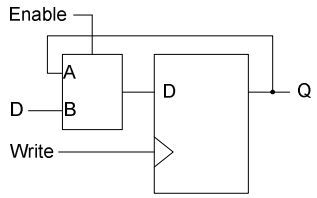

- **Questão número 1:**
  - **Há alguma relação entre Sistemas Digitais e Arquitectura de Computadores?**
- Resposta:
  - Arquitectura de Computadores não é mais do que uma das áreas (porventura a maior) de aplicação directa dos conceitos, técnicas e metodologias apreendidas em Sistemas Digitais.
  - Em Arquitectura de Computadores, contudo, trabalhamos num nível de abstracção diferente, recorrendo na maior parte das vezes a blocos funcionais complexos com cuja síntese, normalmente, não temos que nos preocupar.

- **Questão número 2:**
  - **Será fácil perceber a relação entre a Arquitectura de Computadores e os conhecimentos adquiridos em Sistemas Digitais?**
- Resposta:
  - Nada melhor do que um exemplo para estabelecer essa ligação. Consideremos, para o efeito, o caso dum bloco funcional a que chamaremos "Memória"
  - Uma "Memória", em Arquitectura de Computadores, é um dispositivo com capacidade para armazenar informação digital binária (conjuntos de bits) que deve perdurar no tempo.
  - Observemos então, como "construir" uma memória à custa de blocos básicos bem conhecidos dos sistemas digitais (uma memória de **8** **x** **8**, i.e, **8 posições** de **8 bits**)





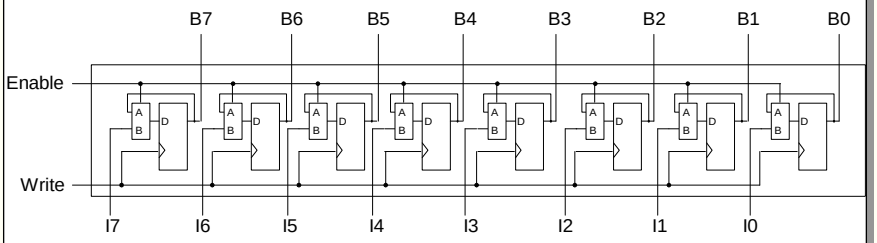






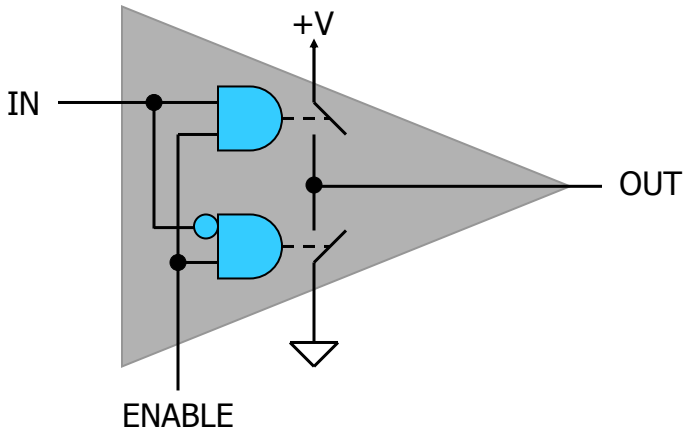



Agrupando em paralelo oito células básicas de memória podemos construir um registo com capacidade para armazenar um byte



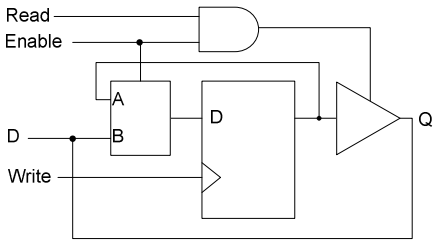

A utilização de portas *tri-state* permite a utilização de um único conjunto de linhas para ler e escrever o registo

- Estrutura interna de uma *gate* Tri-State

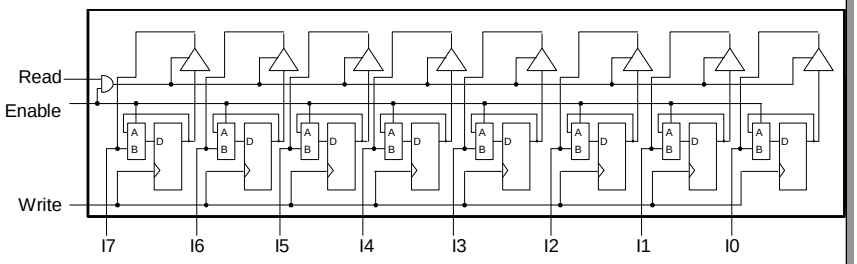






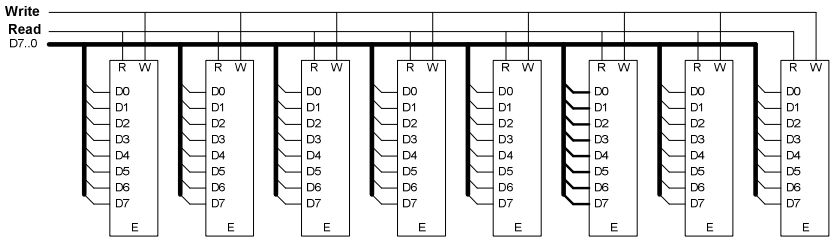
Agrupando em paralelo oito células básicas de memória podemos construir um registo com capacidade para armazenar um byte

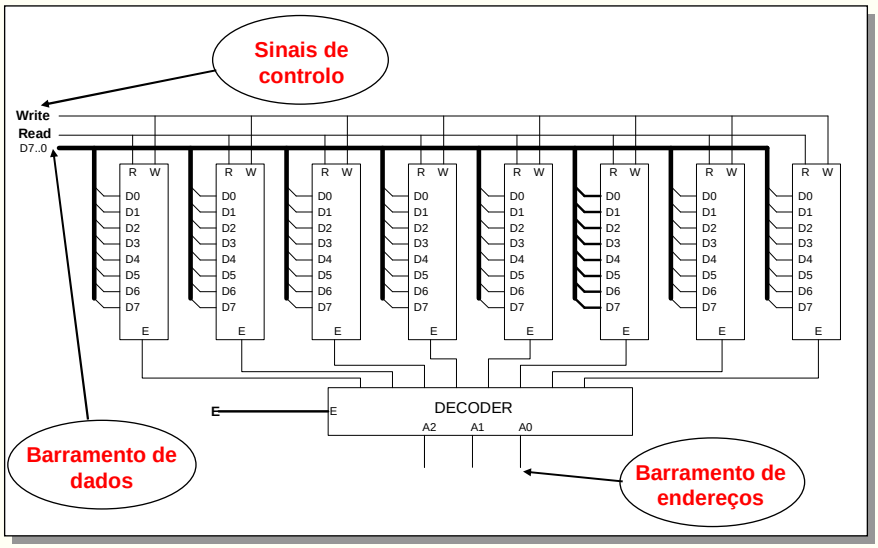



A utilização de portas *tri-state* permite a utilização de um único conjunto de linhas para ler e escrever o registo





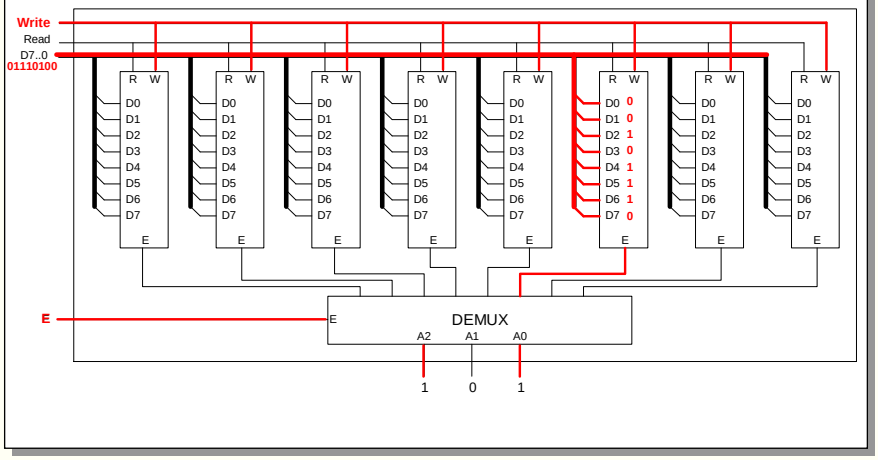




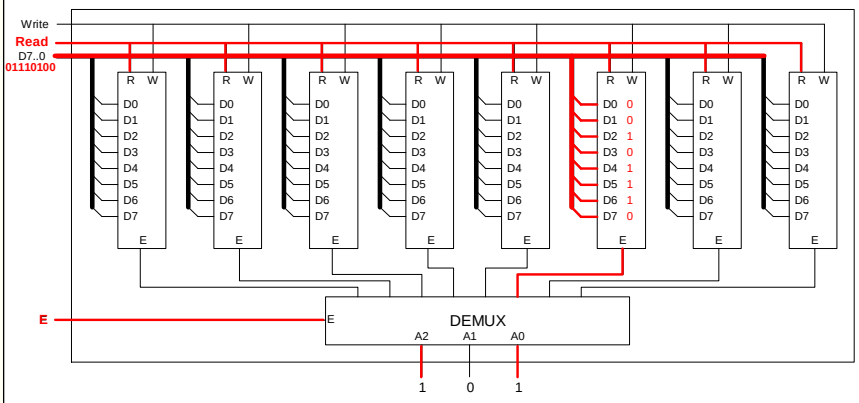





- Escrita na memória



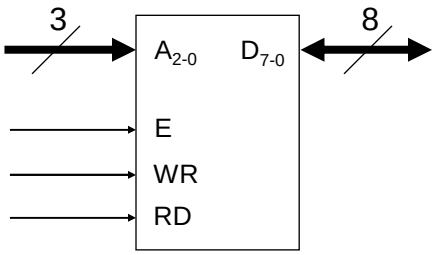
- Leitura da memória








- Bloco funcional correspondente à memória de 8x8



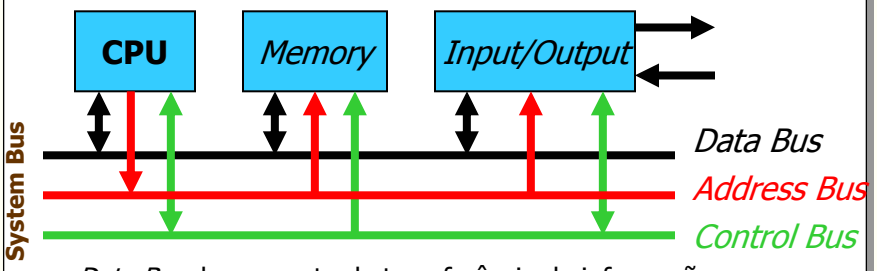
Agora que do verbo (o operador Nand) conseguimos construir uma "Memória", estamos em condições de tentar perceber quais os blocos básicos que constituem uma arquitectura genérica, sem perder de vista que por debaixo da abstracção dos diagramas de blocos se encontra sempre um sistema digital!





- **As unidades fundamentais que constituem um computador são:**
  - **Unidades de entrada** – permitem a recepção de informação vinda do exterior (dados, programas) e que é armazenada em memória
  - **Unidades de saída** – permitem o envio de resultados para o exterior
  - **Memória** – armazenamento de:
    - Programas
    - Dados para processamento
    - Resultados
  - **CPU** – processamento da informação através da execução do programa armazenado em memória

- Arquitectura básica de um sistema computacional (modelo de von Neumann, simplificado)



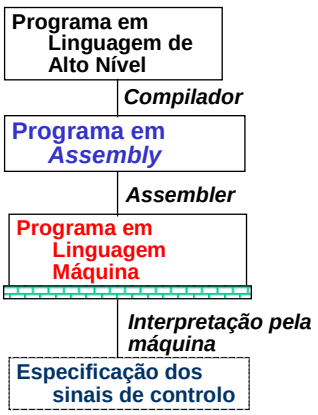

- *Data Bus*: barramento de transferência de informação (CPU↔memória, CPU↔Input/Output)
- *Address Bus*: identifica a origem/destino da informação
- *Control Bus*: sinais de protocolo que especificam o modo como a transferência de informação deve ser feita





- **Endereço** (*address*) – um número (único) que identifica cada posição de memória. Os endereços são contados sequencialmente, começando em 0
  - Exemplo: o conteúdo da posição de memória 2000 é 0x32
- **Espaço de endereçamento** (*address space*) – a gama total de endereços que o CPU consegue referenciar (depende da dimensão do barramento de endereços).
  - Exemplo: um CPU com um barramento de endereços de 16 bits pode gerar endereços na gama: 0x0000 a 0xFFFF (i.e., 0 a $2^{16}$-1)
  - Qual o espaço de endereçamento (⇒maior memória possível) de um processador com endereços de 32 bits?

- Níveis de Representação




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

lw  $15, 0($2)
lw  $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111

ALUOP[0:3] <= InstReg[9:11] & MASK
```

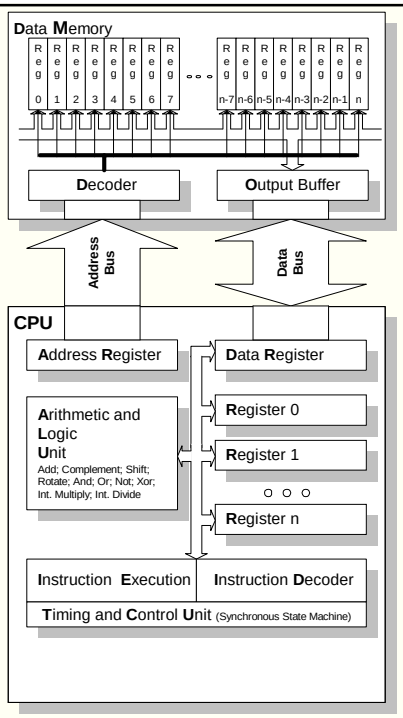




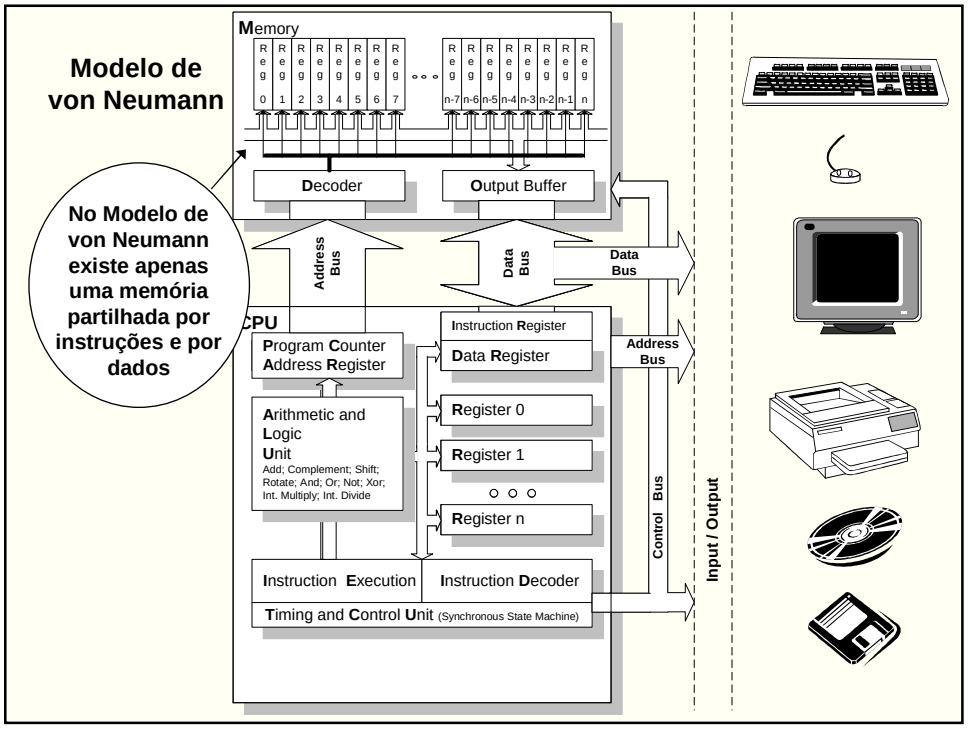
- Independentemente do tipo de CPU e da sua estrutura interna, qualquer instrução deverá ter uma codificação que permita responder às seguintes questões:
  - Qual a operação a realizar ?
  - Qual a localização dos operandos (se existirem) ?
    - podem estar em registos internos do CPU ou na memória externa. No 1º caso deverá ser especificado o número de um registo; no 2º um endereço de memória
  - Onde colocar o resultado ?
    - reg. Internos / memória
  - Qual a próxima instrução ?
    - em condições normais é a instrução seguinte na sequência e, portanto, não é, normalmente, explicitamente mencionada
    - em instruções que alteram a sequência de execução a instrução deverá fornecer o endereço da próxima instrução a ser executada

- O CPU consiste, fundamentalmente, em duas secções:
  - Secção de dados (*data path*): elementos operativos:
    - Registos internos
    - Unidade Aritmética e Lógica (ALU) – Add, Sub, And, Or...
  - Unidade de controlo: responsável pela coordenação dos elementos do *data path*, durante a execução de um programa
    - Máquina de estados síncrona (estado seguinte é função do estado actual e das entradas)
    - As entradas correspondem a informação retirada de cada uma das instruções lidas da memória



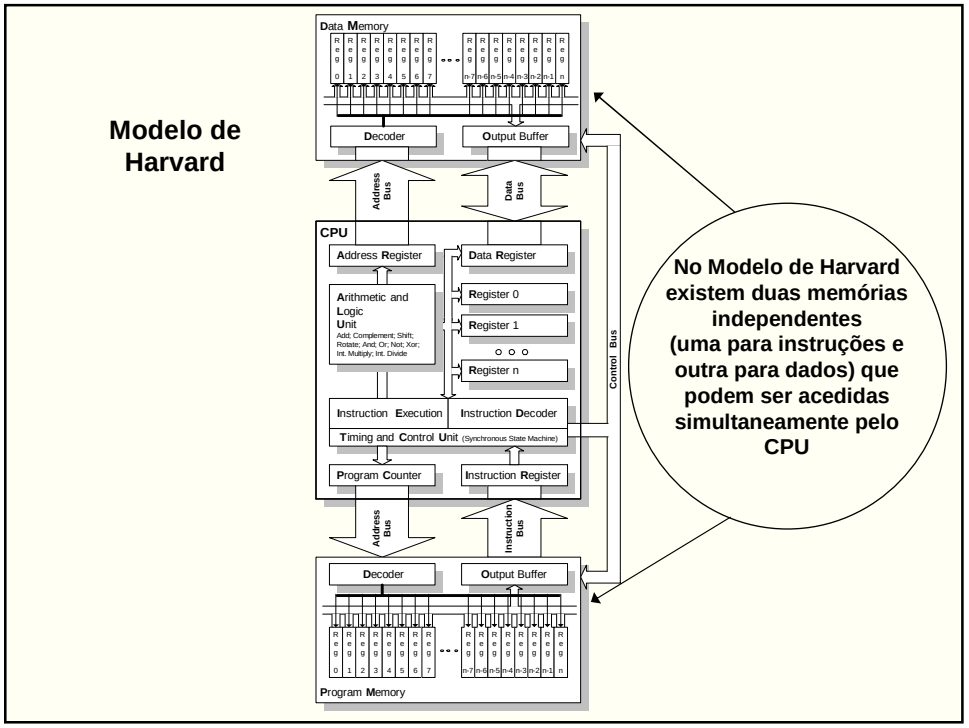


## Modelo de von Neumann



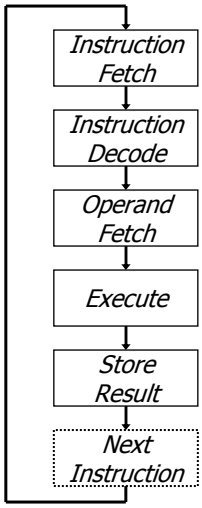

No Modelo de von Neumann existe apenas uma memória partilhada por instruções e por dados

## Modelo de Harvard




No Modelo de Harvard existem duas memórias independentes (uma para instruções e outra para dados) que podem ser acedidas simultaneamente pelo CPU

- Ciclo básico de execução de uma instrução




***Fetch-execute cycle***





- O que é a Arquitectura de Computadores?

  – Arquitectura de Computadores =
  Arquitectura do Conjunto de Instruções (ISA) +
  Organização da Máquina

Conjunto (*Set*) de Instruções: a colecção de todas as operações que o processador pode executar

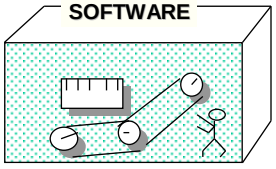
**ISA** – *Instruction Set Architecture*

- Arquitectura do Conjunto de Instruções (também designada por "modelo de programação"):
  - Uma importante abstracção que representa a interface entre o h/w e o nível mais básico de s/w
  - Descreve tudo o que o programador necessita de saber para programar correctamente, em linguagem máquina, um determinado processador
  - Descreve a funcionalidade, independentemente do h/w que a implementa. Pode assim falar-se de "arquitectura" e "implementação de uma arquitectura" (Ex. Processadores AMD compatíveis com Intel x86)

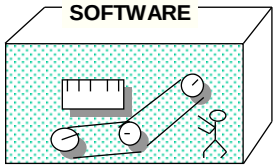




- Arquitectura do Conjunto de Instruções
  - ... os atributos de um sistema computacional tal como são vistos pelo programador, i.e. a estrutura conceptual e o comportamento funcional, de forma distinta e independente da organização do fluxo de informação e dos respectivos elementos de controlo, do desenho lógico e da implementação física. – Amdahl, Blaaw, and Brooks, 1964
  - Conjunto de Instruções
  - Organização da memória programável
  - Número e tipos de Registos
  - Tipos de dados e estruturas de dados
    - Codificação e representação
  - Modos de endereçamento e de acesso a dados e instruções
  - Formato das Instruções
  - Condições de Excepção



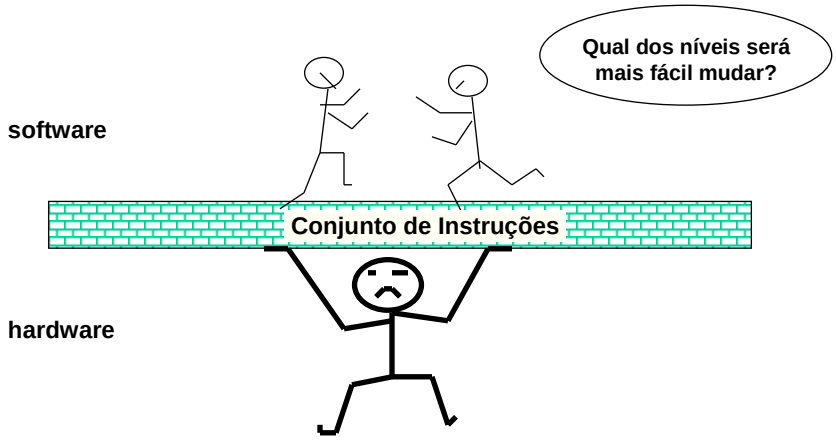
- Objectivos da Arquitectura do Conjunto de Instruções
  - Implementação eficiente em hardware
  - Implementação simples em hardware
  - Fácil de entender e programar
  - Compiladores eficientes








- Conjunto de Instruções: um Interface Crítico

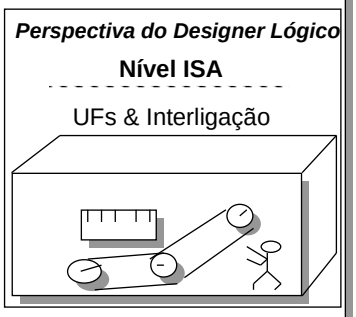


- Exemplos de ISAs (*Instruction Set Architecture*)
  - IBM S/360 1964
  - VAX 1977
  - Intel x86 (8086, 80286, 80386, 80486,Pentium, MMX, ...) 1978
  - SGI MIPS (MIPS I, II, III, IV, V) 1986
  - HP PA-RISC (v1.1, v2.0) 1986
  - Sun Sparc (v8, v9) 1987
  - Digital Alpha (v1, v3) 1992

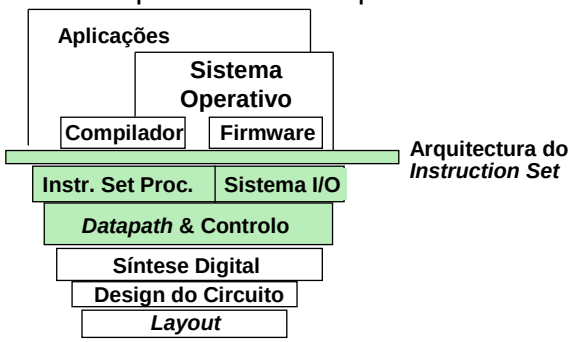




- Organização da máquina
  - Características operativas e de performance das principais unidades funcionais
    - (ALU, Registos, *Shifters*, Unidades Lógicas, ...)
  - De que modo esses componentes são interligados
  - Fluxo de informação entre componentes
  - Lógica e meios através dos quais esse fluxo é controlado
  - Coreografia das Unidades Funcionais para implementar a ISA



- Arquitectura de Computadores será portanto...

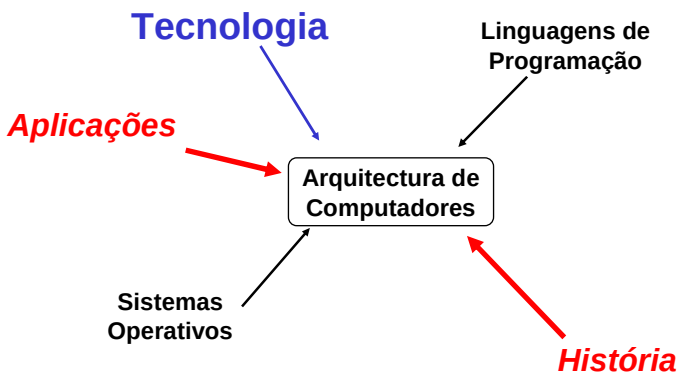



  - Coordenação de muitos níveis de abstracção...
  - Sob o efeito de um conjunto de forças em permanente mutação.
  - Design, Medida, e Avaliação

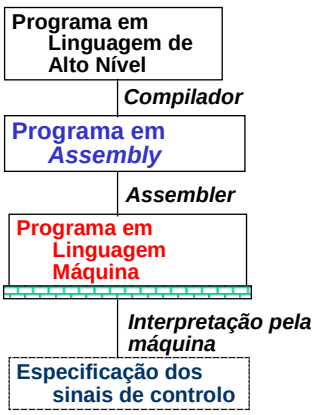




- Forças envolvidas em Arquitectura de Computadores



- Níveis de Representação




```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

```
lw  $15, 0($2)
lw  $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)
```

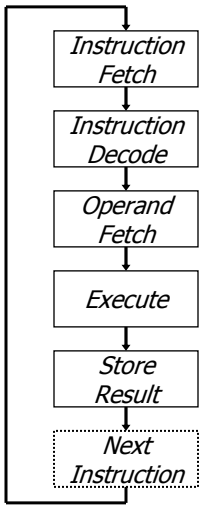
```
0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111
```

```
ALUOP[0:3] <= InstReg[9:11] & MASK
```





- Arquitectura do conjunto de instruções define:
  - Formato e codificação das instruções
    - como são descodificadas?
  - Localização de operandos e resultados
    - onde?
    - quantos operandos explícitos?
    - como localizar?
    - quais podem residir na memória externa?
  - Tipo e dimensão dos dados
  - Operações
    - quais devem ser suportadas?
  - Instruções auxiliares
    - jumps, conditions, branches
    - fetch-decode-execute (implícito)!



- Formato das instruções
  - Tamanho variável
    - Código mais pequeno
    - maior flexibilidade
    - *Instruction fetch* em vários passos
  - Tamanho fixo
    - *Instruction fetch* e *decode* mais simples
    - Mais simples de implementar em *pipeline*





- Número de registos
  - Vantagens de um número pequeno de registos
    - Menos hardware
    - Acesso mais rápido
    - Menos bits para identificação do registo
    - Mudança de contexto mais rápida
  - Vantagens de um número elevado de registos
    - Menos acessos à memória
    - Variáveis em registos
  - Certos registos podem ter restrições de utilização

- Localização dos operandos
  - Acumulador
    - Resultado das operações é armazenado num registo especial designado de acumulador
  - Baseados em *Stack*
    - Operandos e resultado armazenados numa *stack* de registos
  - *Register-Memory*
    - Operandos residem em registos ou em memória
  - *Load-store architecture*
    - Operandos residem em registos de uso geral.





- Conjunto de Instruções do MIPS
  - Instruções de tamanho fixo (32 bits)
  - 32 registos de uso geral ($0 valor fixo zero)
  - *Load-store architecture*