Conceitos Básicos de Arquitetura de Computadores e Circuitos Digitais

Conceitos Gerais

1. Processamento de Dados

- O computador pode ser definido como uma máquina capaz de coletar, manipular e dar resultados da manipulação de informações. Por ter essas características, o computador já foi chamado de equipamento de processamento eletrônico de dados.

- A manipulação das informações coletadas é chamada de processamento e as informações iniciais são chamadas dados, por isso é comum vermos a expressão processamento de dados.

- Dados e informações não são sinônimos. Informações são dados organizados e/ou processados

2. Organização vs. Arquitetura de Computadores

- O **organização de computadores** é uma área da computação dedicada ao estudo dos detalhes físicos do computador, como material usado na construção do processador e memória, frequência de clock, etc.

- Os usuários e até mesmo os desenvolvedores, via de regra, não precisam conhecer profundamente a organização do computador que trabalham.

- Já o conhecimento da **arquitetura de um computador** é mais relevante a um programador, pois suas características possuem impacto direto no desenvolvimento de um software.

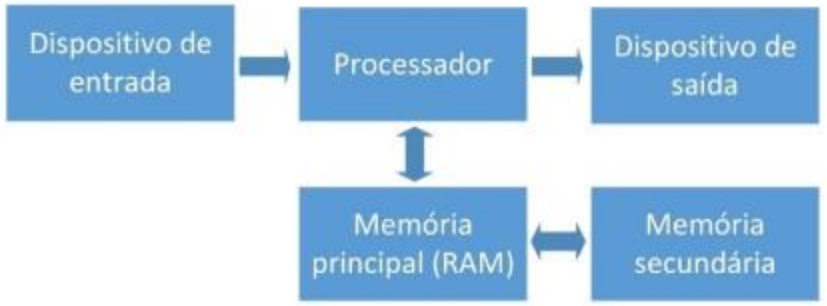
- Fazem parte da arquitetura conceitos como conjunto de instruções do processador (ex.: ADD, SUB, entre outras), tamanho da palavra (quantidade de bits utilizada para transferência entre o processador e a memória - ex.: palavra de 32 bits), etc.

3. Arquiteturas Clássicas

- Um **sistema de computação** é um conjunto de componentes que são integrados para funcionar como se fosse um único elemento, com objetivo realizar o processamento de dados.

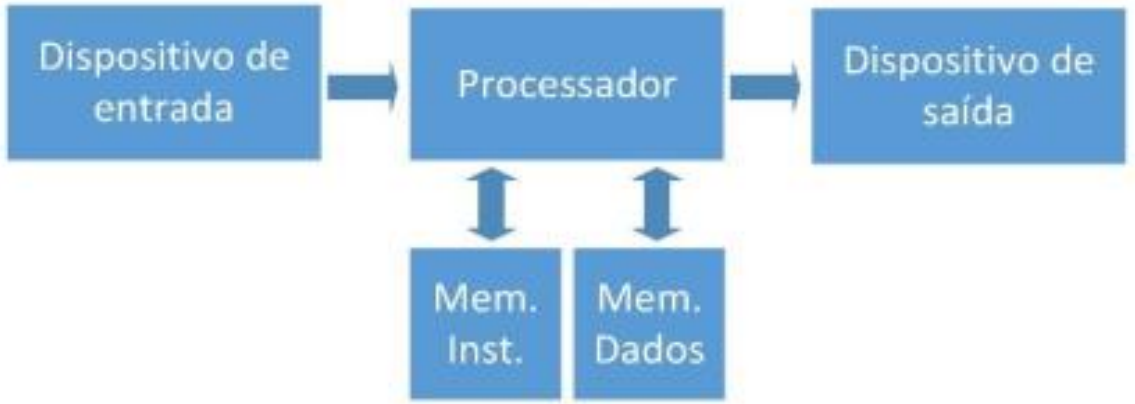
- Os primeiros computadores surgiram contendo apenas dispositivos de entrada (ex.: teclado), processador e dispositivos de saída (ex.: monitor de vídeo).

- O engenheiro John von Neumann melhorou a arquitetura inicial, acrescentando a memória (principal e secundária) para armazenar programas e dados, tornando o processamento muito mais rápido e eficaz. Tal arquitetura é conhecida como **Arquitetura de von Neumann**.



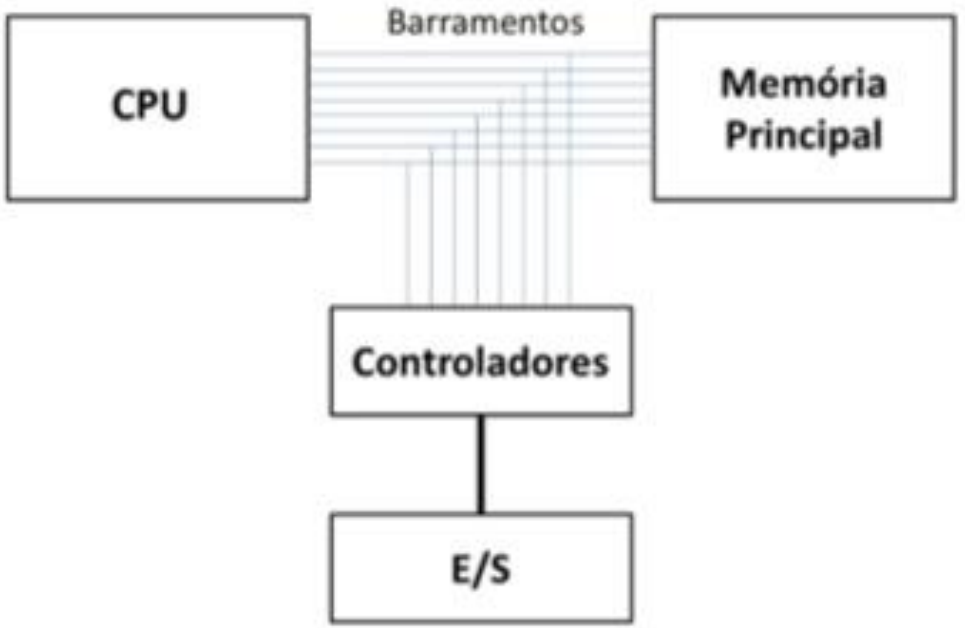
3. Arquiteturas Clássicas

- Um melhoramento da Arquitetura de von Neumann é a Arquitetura de Harvard. Ela possuir duas memórias diferentes e independentes em termos de **barramento** e **ligação** ao processador.
- Sua principal característica é o acesso à **memória de dados** de modo separado em relação à **memória de instruções (programa)**, o que é tipicamente adotado pelas memórias cache da atualidade.
- Com essa separação de **dados** e **instruções** em memórias e barramentos separados, o processador consegue acessar as duas **simultaneamente**, obtendo um desempenho melhor



3. Arquiteturas Clássicas

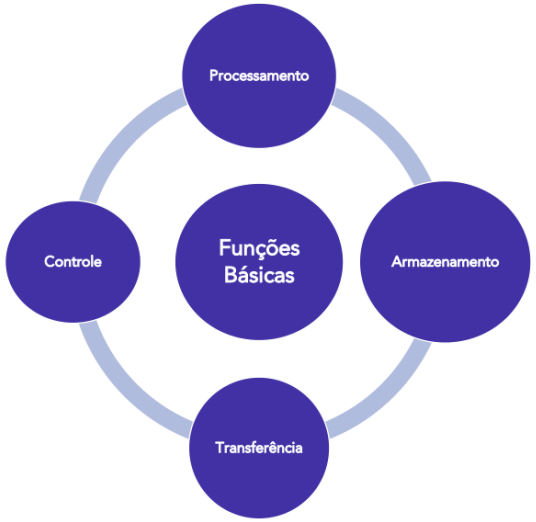
- Como podemos observar na figura, um computador atual continua utilizando a essência da Arquitetura de von Neumann e/ou a Arquitetura de Harvard.
- Os barramentos são os responsáveis pela comunicação entre o processador, a memória principal e os dispositivos de entrada e saída



3. Arquiteturas Clássicas

- De uma forma geral, são funções básicas de um computador:

1. **Processamento de dados**: realizado pelo processador
2. **Armazenamento de dados**: pode ocorrer de forma temporária ou de longo prazo
3. **Transferência de dados**: ocorre através de sistemas de interconexão (barramento do sistema), permitindo a comunicação da CPU com dispositivos de entrada e saída.
4. **Controle:** uma unidade de controle gerencia os recursos do computador (CHIPSET).



4. Arquiteturas de Processadores

- Quando o assunto é saber qual a melhor arquitetura de processador, sempre há polêmica. Por exemplo, muitos defendem que os “Macs” são mais rápidos por terem chips RISC. Mas o que é RISC? E CISC? Quais vantagens e desvantagens?

- Um processador **CISC** (Complex Instruction Set Computer) é capaz de executar várias centenas de instruções complexas diferentes, sendo extremamente versátil. Ex: 386, 486.

- Alguns fabricantes decidiram seguir o caminho contrário, criando o padrão **RISC** (Reduced Instruction Set Computer). Os processadores RISC são capazes de executar apenas algumas poucas instruções simples, por isso são mais simples e baratos.

- Por terem um menor número de circuitos internos, o RISC pode trabalhar com frequências mais altas. Alguns exemplos de processadores RISC são Sparc (Sun), Mips (Silicon Graphics), Power (IBM) e Alpha (DEC).

4. Arquiteturas de Processadores

- Agora surgiu uma pergunta, como um chip que é capaz de executar algumas poucas instruções pode ser considerado por muitos, mais rápido do que outro que executa centenas delas?

- A grande questão é que um processador RISC é capaz de executar suas poucas instruções muito mais rapidamente.

- A ideia principal é que apesar de um processador CISC ser capaz de executar centenas de instruções diferentes, apenas algumas são usadas frequentemente, o que parece ser um desperdício, não?

- Os processadores atuais incorporam características das duas arquiteturas. Mas no mundo acadêmico, é importante saber a diferença dos dois.

4. Arquiteturas de Processadores

- O processador CISC possui as seguintes características:

- Possui grande quantidade de instruções, com múltiplos modos de endereçamento;
- O conceito de microprogramação facilitou o projeto de instruções complexas (do 386 para o 486 foram implementadas instruções sem recriar a arquitetura)
- As instruções são completas e eficientes;
- Instruções de máquina de “alto nível” (complexidade semelhante à dos comandos de alto nível).
- Formato de 2 operandos é o mais comum, ex.: ADD AX, mem;
- Uso dos modos: Registrador para registrador, Registrador para memória, Memória para registrador;
- Instruções com largura variável;
- Instruções requerem múltiplos ciclos de relógio para completar a execução
- Poucos registradores, devido ao pouco espaço no chip
- Existe possibilidade de acesso a operandos na memória;
- Há registradores especializados: controle (flags), segmento (ponteiro da pilha) etc.

4. Arquiteturas de Processadores

- O processador RISC possui as seguintes características:

- Possui poucas instruções e todas possuem a mesma largura;
- Execução otimizada de chamada de funções;
- Menor quantidade de modos de endereçamento;
- Uso intenso de pipelining, pois é mais fácil implementar o paralelismo quando se tem
- instruções de mesmo tamanho;
- Execução rápida de cada instrução (uma por ciclo de relógio);
- Processadores RISC não requerem microcódigos (sobra mais espaço no chip);
- Menos acesso à memória principal,
- Maior quantidade de registradores (menos acesso a memória principal)

4. Arquiteturas de Processadores

- A tabela abaixo mostra características de diferentes processadores. Vamos classifica-los como RISC ou CISC:

Características	MIPS R4000	RS/6000	VAX11/780	INTEL 486
Quantidade de instruções	94	183	303	235
Modos de endereçamento	1	4	22	11
Largura de instruções (bytes)	4	4	2-57	1-12
Quantidade de registradores de uso geral	32	32	16	8

4. Arquiteturas de Processadores

- Olhando pela quantidade de instruções, o que apresenta 94 parece ser RISC (poucas instruções) e o 303 CISC (muitas instruções), mas os outros dois são próximos e fica a dúvida.

- Analisando os modos de endereçamento, fica evidente que os dois primeiros são RISC (poucos modos), enquanto os dois últimos possuem bem mais.

- Pela largura de instruções fica mais claro ainda que os dois primeiros são RISC, pois possuem uma largura fixa de instruções (4 bytes), enquanto os outros dois possuem instruções de diversos tamanhos (2 a 57 bytes um deles e o outro entre 1 e 12 bytes).

- E para arrematar nossa análise, os dois que achamos que são RISC possuem mais registradores (32 cada um deles), enquanto os outros dois processadores possuem menos registradores.

5. Linguagem de Máquina e de Montagem

- A grande maioria dos profissionais de TI trabalham com linguagens de programação de alto nível (Python, Java, C#, etc.).

- Para se trabalhar com estas linguagens não é necessário ter conhecimento aprofundado da arquitetura do processador e do computador como um todo.

- Entretanto, se um desenvolvedor precisa programar com linguagem de máquina (o termo correto é linguagem de montagem), será necessário conhecer mais profundamente a arquitetura do computador (registradores, memória, tipos de dados aceitos, estrutura da ULA).

5. Linguagem de Máquina e de Montagem

- Nas linguagens de alto nível existe o compilador que transformam o código digitado pelo programador em um código binário específico para determinada arquitetura. Quando se trabalha com linguagem de máquina, cabe ao programador fazer este trabalho.

- Os processadores já vem de fábrica com um conjunto de instruções definidas. Utilizando este conjunto de instruções aceitas por um determinado processador (ou família de processador), é possível programar em "baixo nível".

C	ASSEMBLY
`a = b;`	`MOV A, B`
`a = a + b;`	`ADD A, B`
`a = b + c;`	`ADD A, B, C`
`a = a - b;`	`SUB A, B`
`a = b - c;`	`SUB A, B, C`
`a = a*b;`	`MUL A, B`
`a = b*c;`	`MUL A, B, C`
`a = a/b;`	`DIV A, B`
`a = b/c;`	`DIV A, B, C`

5. Linguagem de Máquina e de Montagem

- Para implementação de um software utilizando código de máquina, cada instrução deve conter as informações exigidas pelo processador para a execução. Os elementos de uma instrução de máquina são:

1. Código de operação: especifica a operação a ser realizada (ex.: ADD, E/S). A operação é especificada por um código binário conhecido como código da operação (**opcode**);

2. Referência a operando fonte: operandos que são entradas para a operação;

3. Referência a operando de resultado: a operação deve produzir um resultado;

4. Referência à próxima instrução: informa ao processador onde buscar a próxima instrução depois que a execução da instrução atual finalizar. Na maioria das vezes a próxima instrução vem na sequência corrente. Neste caso, não é necessário especificar

5. Linguagem de Máquina e de Montagem

- Os **operandos fonte e resultado** podem estar armazenados em uma das seguintes áreas:

1. Memória principal ou virtual: assim como as referências à próxima instrução, o endereço da memória (principal ou virtual) deve ser fornecido;

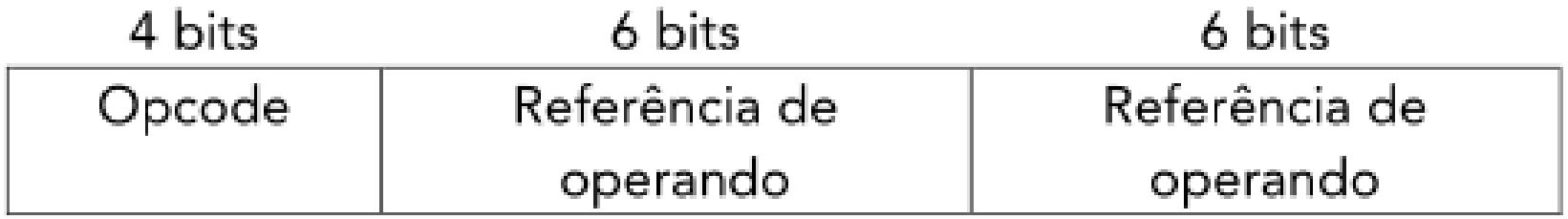
2. Registradores do processador: um processador tipicamente possui um ou mais registradores, os quais podem ser referenciados por instruções de máquina. Cada registrador recebe um nome ou número exclusivo, e a instrução deve conter o número do registrador desejado;

3. Imediato: o valor do aperando está contido em um campo na instrução que está sendo executada;

4. Dispositivo de E/S: a instrução precisa especificar o módulo e o dispositivo de E/S para a operação.

6. Representação de Instruções de Máquina

- Cada instrução é representada por uma sequência de bits. A instrução é dividida em campos que formam os 4 elementos da instrução. Um exemplo de instrução simples de 16 bits é:



6. Representação de Instruções de Máquina

- Para executar a instrução de máquina, o processador carrega a instrução presente no registrador de instrução (IR). Em seguida, é realizada a extração dos operandos para que a execução enfim seja realizada.

- Dificilmente um ser humano consegue trabalhar corretamente com representações binárias das instruções de máquina. Por essa razão, é utilizado uma representação simbólica das instruções.

- Um exemplo dessa prática foi utilizado para o conjunto de instruções do IAS, o primeiro computador eletrônico construído pelo Instituto de Estudos Avançados de Princeton (feito pelo próprio John Von Newman).

6. Representação de Instruções de Máquina

- Os opcodes são representados por abreviações, conhecidas por mnemônicos, os quais indicam a operação.

- Operandos também são representados simbolicamente, como por exemplo a instrução **ADD R, X**, que pode significar a soma do valor contido na localização X com o conteúdo do registrador R.

- Alguns exemplo comuns de mnemônicos de opcodes são:

ADD - Adição;
SUB - Subtração;
MUL - Multiplicação;
DIV - Divisão;
LOAD - Carrega dados da memória;
STOR - Armazena dados na memória.

7. Modos de Endereçamento

- O(s) campo(s) de endereço são relativamente pequenos. Para tornar possível referenciar um grande intervalo de locais da memória principal, uma série de técnicas de endereçamento foram desenvolvidas, sendo as principais:

1. **Endereçamento imediato**: é a forma mais simples de endereçamento, no qual o valor do operando está presente explicitamente na instrução. Como exemplo, a instrução MOV B, #5H move o valor 5 (em hexadecimal) para o registrador B. O valor 5 é um endereçamento imediato. É mais rápido (evita busca na memória), mas limita o tamanho do operando.

2. **Endereçamento direto**: Essa técnica era comum nos primeiros computadores, requer apenas uma referência à memória e nenhum cálculo especial. A limitação é que ela oferece um espaço de endereçamento limitado (pequeno)

7. Modos de Endereçamento

3. **Endereçamento indireto**: no endereçamento direto, o tamanho do campo de endereço é muito pequeno, o que limita o espaço de endereços. A solução é ter um campo de endereço fazendo referência ao endereço de uma palavra na memória, a qual possui o endereço completo do operando. A desvantagem é que a execução da instrução requer duas referências à memória para obter o operando, uma para obter apenas o endereço e a outra para obter o operando (valor) em si.

4. **Endereçamento por registradores**: semelhante ao endereçamento direto, porém o campo de endereço faz referência a um registrador em vez de um endereço de memória.

5. **Endereçamento indireto por registradores**: análogo ao endereçamento indireto, sendo que a diferença é que no lugar de referência à memória, existe referência a um registrador

6**. Endereçamento de pilha**: itens são adicionados e retirados do topo da pilha, sendo que há um ponteiro cujo valor é o endereço do topo. O ponteiro da pilha é mantido em um registrador. O modo de endereçamento de pilha é uma forma de endereçamento implícito, sendo que as instruções de máquina não necessitam incluir uma referência de memória, devem apenas operar no topo da pilha.

8. Linguagem de Montagem

- Um processador (CPU) entende e executa apenas instruções de máquina (opcodes e operandos em uma sequência binária de zeros e uns). Programar diretamente em linguagem de máquina seria muito complexo para um ser humano.

- Para facilitar o trabalho dos desenvolvedores, nomes simbólicos (mnemônicos) podem ser utilizados no lugar de cada instrução binária, e os endereços podem ser representados por meio de símbolos.

- É neste contexto que temos a **linguagem de montagem (Assembly)**, que pode ser traduzida para linguagem de máquina através de um montador (assembler).

- Cada instrução de Assembly é traduzida em uma instrução de máquina pelo montador (1:1). É uma linguagem dependente do hardware, ou seja, há uma linguagem de montagem diferente para cada tipo ou família de processador.

9. Compilador, Montador e Interpretador

- A maioria dos desenvolvedores utilizam linguagens de alto nível em seus projetos. Uma vez que o código está implementado, é necessário converter para linguagem de máquina, é neste ponto que diferentes técnicas podem ser utilizadas:

- **Compilador**: Um compilador tem como função traduzir o código fonte de uma linguagem de programação de alto nível (C, Pascal, etc.) para uma linguagem de programação de baixo nível (Assembly ou código de máquina). Dependendo do autor, o entendimento é que o compilador converte de alto nível para Assembly, outros autores entendem que a conversão é de alto nível para código objeto (binário).

- **Montador (Assembler)**: É responsável pela tradução de código feito em linguagem Assembly para código objeto. Dependendo da implementação, o montador pode estar integrado ao compilador (o compilador "faz tudo", tendo como entrada um código fonte e entregando como resultado final o código objeto).

9. Compilador, Montador e Interpretador

- **Ligador (linker ou link-editor):** Junta todos os arquivos objetos gerados na compilação, e gera o programa executável em linguagem de máquina. Por exemplo, o #include (linguagem de programação C) faz com que bibliotecas sejam incluídas no programa para que comandos que estão descritos nelas possam ser utilizados.

- **Interpretador**: É um programa que traduz em tempo de execução o código de programação de alto nível em código de máquina sem criar um arquivo executável. Esse processo é mais lento, se comparado com a compilação. Alguns exemplos de linguagens interpretadas são BASIC, PHP e Python.