Arquitetura e Organização de Computadores

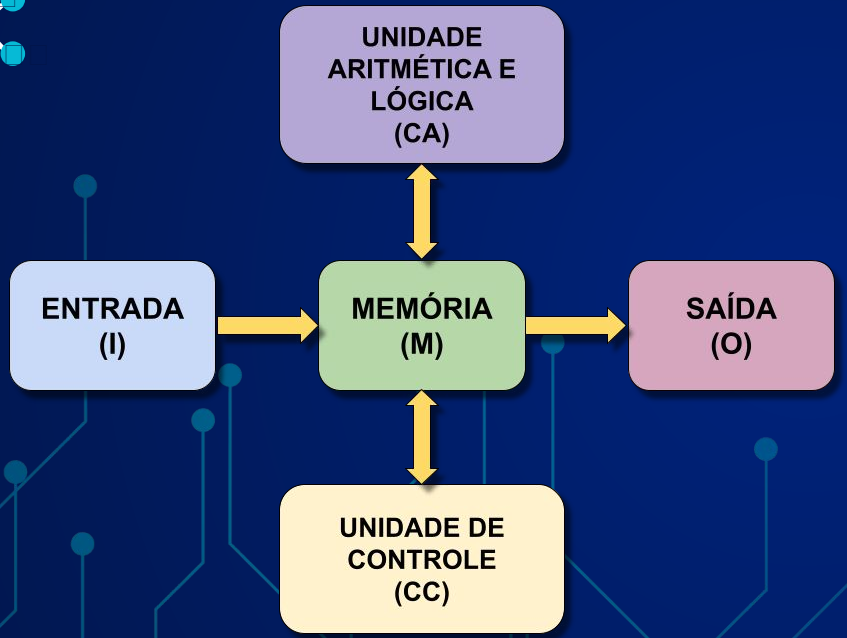
Uma Introdução

Gabriel P. Silva – José Antonio Borges

Organização do computador e do processador

Capítulo 3

3.1 Arquitetura de Von Neumann



Von Neumann

A proposta de computador controlado pelo programa armazenado em memória foi um dos conceitos fundamentais apresentados por Von Neumann

Modelo de Von Neumann

- **Unidade Aritmética e Lógica**: executa as operações aritméticas e lógicas.
- **Unidade de Controle**: responsável pelo sequenciamento das operações, transferência dos dados e instruções e pelo controle das demais unidades do computador.
- **Memória**: armazena as instruções, os dados de entrada, as tabelas de referência, e os resultados intermediários dos programas em execução.

Modelo de Von Neumann

- **Entrada**: É a unidade que transfere a informação (numérica ou não) do meio externo. Todas as transferências devem ser feitas para a memória e nunca diretamente para a unidade de controle.
- **Saída**: É a unidade que transfere a informação (numérica ou não) para o meio externo. Todas as transferências devem ser feitas da memória para o meio externo, e nunca diretamente da unidade de controle.

Modelo de Von Neumann

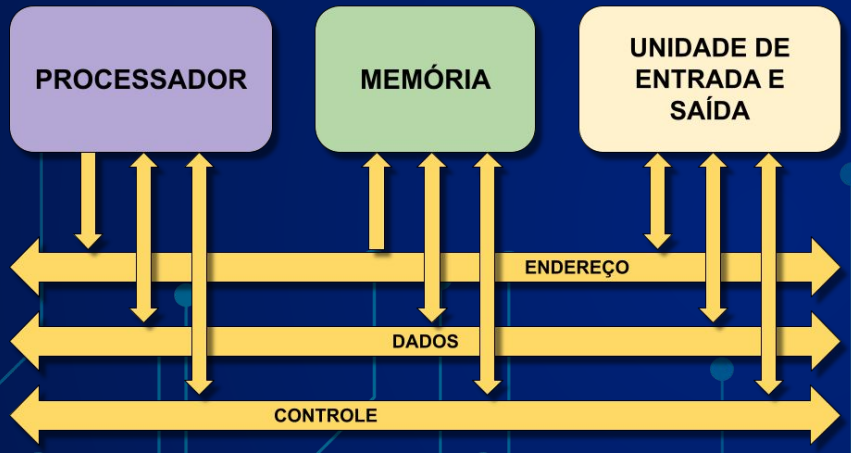
- Von Neumann sugere o uso da numeração binária para a representação interna dos números, ao invés da numeração decimal.
- O componente eletrônico básico era o E-element, um dispositivo que recebe e emite sinais por uma linha conectada a ele, de forma equivalente ao axônio de um neurônio.
- Poderia ser implementado com 1 ou 2 tubos de vácuo (válvulas) e combinado para criar circuitos mais complexos para a unidade aritmética, memória e unidade de controle.

Harvard

Na arquitetura Harvard as instruções e dados do programa são armazenados em memórias distintas.

3.2 Modelo de Barramento de Sistema

Modelo de Barramento de Sistema

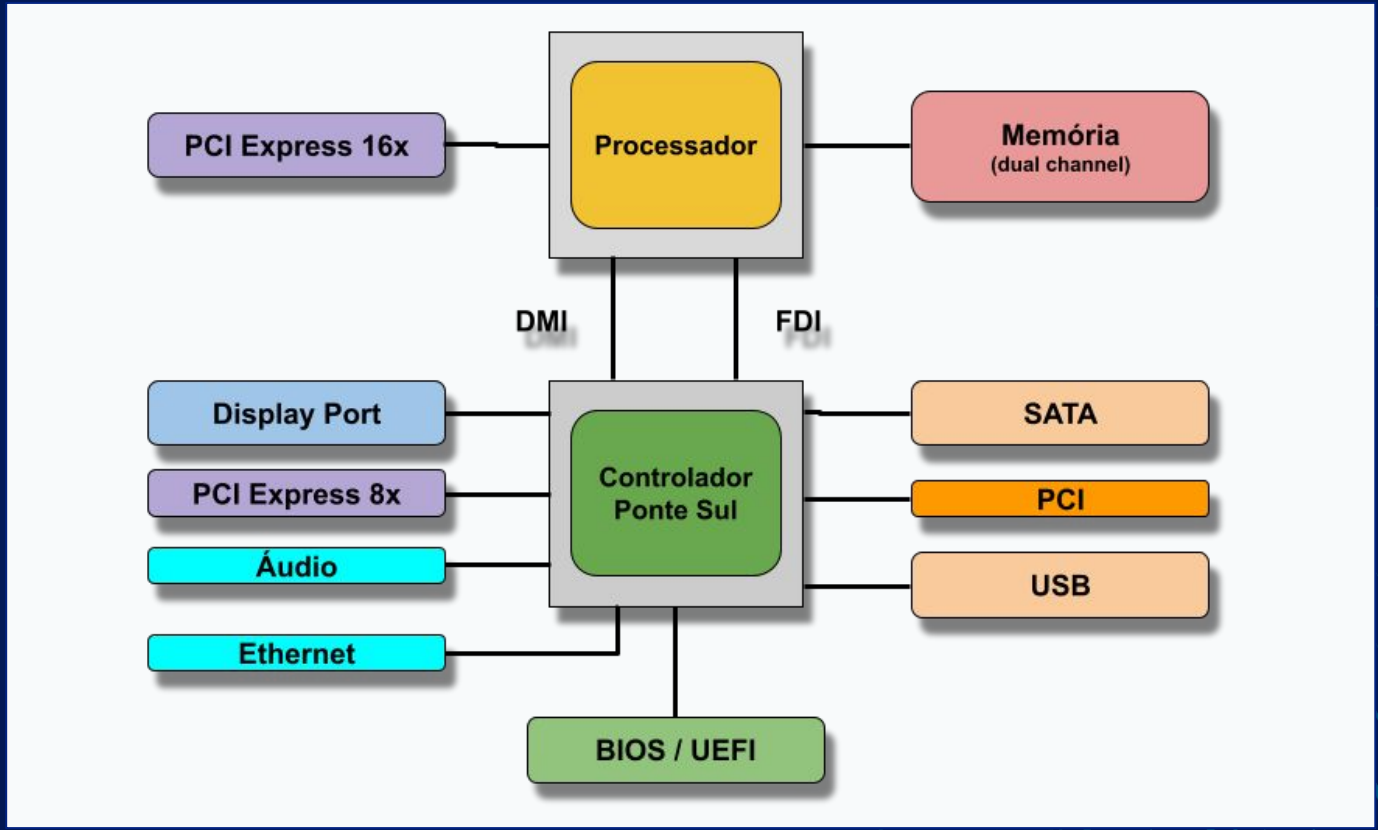


O barramento de sistema é composto pelos barramentos de endereço, dados e controle. O barramento de endereços transporta os sinais de endereço através de fios ou trilhas metálicas, conectando o processador, memória e a unidade de entrada/saída.

Modelo de Barramento de Sistema

- Nesse modelo, a unidade de controle e a unidade aritmética, e também novos elementos de memória chamados de registradores, são agregados em um único componente que recebe o nome de processador.
- A unidade de entrada e a unidade de saída são apresentadas agora também como um único módulo, chamado de unidade de entrada/saída.
- A memória continua sendo vista com uma unidade independente, com as mesmas funções da arquitetura de Von Neumann, ou seja, armazenamento de dados e instruções dos programas em execução.

Componentes do Computador Moderno



Componentes do Computador Moderno

- Nos computadores pessoais modernos não há um barramento de sistema explícito, mas apenas um controlador (chipset) de E/S que faz a intermediação entre o processador e os dispositivos de entrada e saída.
- Na mesma pastilha do processador estão embutidos também o controlador e o barramento de acesso à memória principal; o controlador e/ou a interface de interconexão à placa de vídeo; e ainda um terceiro barramento que faz a interface com o controlador de E/S externo.

Processador

O processador é responsável pela execução dos programas carregados na memória do computador

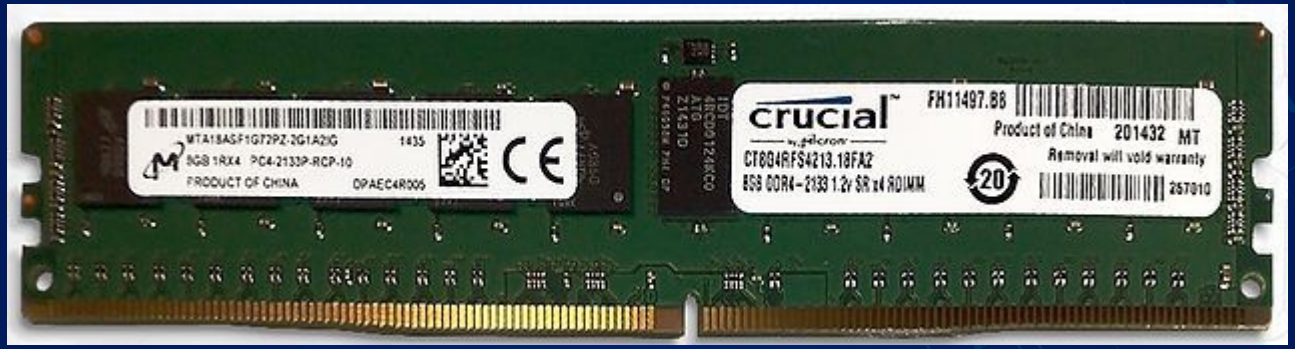
Funções do Processador

- Buscar instruções e dados na memória.
- Programar a transferência de dados entre a memória e os dispositivos de entrada/saída.
- Decodificar as instruções.
- Executar as operações aritméticas e lógicas.
- Realizar o tratamento das exceções na execução das instruções.
- Responder aos sinais enviados por dispositivos de entrada/saída, tais como interrupções e sinais de erro.

Memória

- A memória principal é utilizada para armazenar os programas (instruções e dados) que vão ser processados durante a operação normal do computador.
- A menor unidade de informação que pode ser manipulada individualmente na memória é o byte, um conjunto de 8 bits, sendo que cada byte possui um endereço distinto na memória.
- Várias tecnologias foram utilizadas como memória do computador ao longo do tempo, sendo hoje em dia utilizada a memória semicondutora de estado sólido.

Memória


crucial
FH11497.B6
Product of China 201432 MT
Removal will void warranty
CT8G4RFS4213.18FA2
8GB 1RX4 PC4-2133P-RCP-10
PRODUCT OF CHINA

Memória

- A unidade de memória do computador é formada uma parte volátil, chamada de memória primária ou principal, e por outra parte não volátil.
- As informações armazenadas na memória principal podem ser alteradas durante a execução de um programa, mas são mantidas apenas enquanto o computador estiver ligado, sendo perdidas quando for desligado.

Memória

- A memória não volátil, que mantém o seu conteúdo mesmo quando o computador é desligado, é utilizada para armazenar os programas responsáveis por iniciar o funcionamento do computador, realizar os testes iniciais e copiar o sistema operacional do disco para a memória principal.
- Nos primeiros computadores pessoais compatíveis com o IBM/PC, essa memória recebia o nome de BIOS (Basic Input/Output System), relativa ao nome do programa que ela armazenava.

Entrada e Saída

- São funções da unidade de entrada e saída
  - - Permitir a comunicação entre os seres humanos e o computador.
  - - Converter a informação analógica de/para o formato digital.
  - - Fornecer ou obter dados para alimentar o processador e a memória.
  - - Receber informações do processador e/ou memória para exibição ou armazenamento permanente.

Teclado de computador

Entrada e Saída

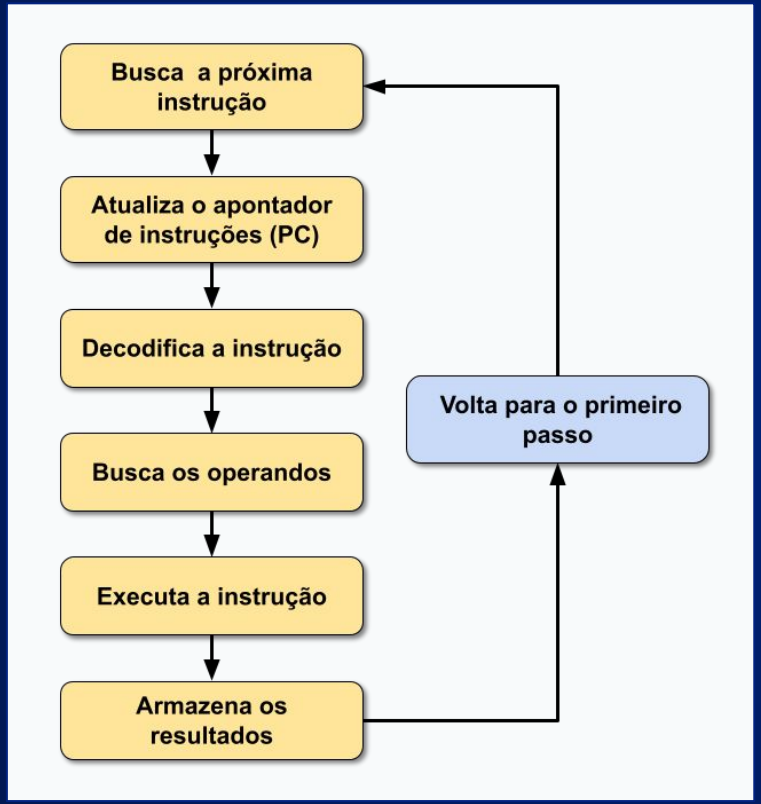
- A memória secundária ou memória de massa é onde os programas e dados, incluindo aqueles do sistema operacional, são armazenados de uma forma persistente no computador.
- Uma das características da memória secundária é o alto volume de dados e o baixo custo de armazenamento por byte quando comparado com a memória principal.
- São exemplos os discos magnéticos, unidades de estado sólido, fitas magnéticas, discos ópticos, pendrives, entre outros.

3.3 O Funcionamento do Processador

Processador

- O processador é o componente eletrônico no computador responsável pela interpretação das instruções em linguagem de máquina e controle de todos os demais dispositivos do computador, com o objetivo de realizar as tarefas determinadas pelas aplicações do usuário.
- Para executar um programa armazenado na memória, o processador realiza constantemente a busca de instruções no endereço de memória indicado pelo apontador de instruções (PC).

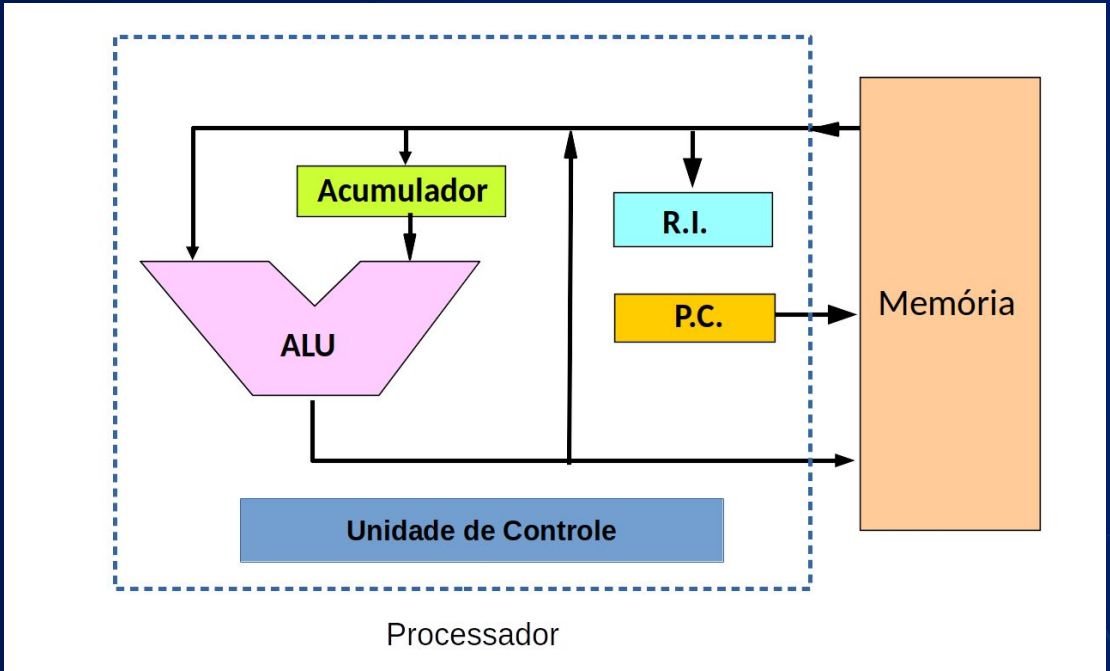
Ciclo de Busca de Instrução



Processador

- A organização interna de um processador para a execução de suas instruções pode variar bastante.
- Internamente possuem no mínimo uma unidade aritmética e lógica, unidade de controle, caminho de dados e registradores.
- Enquanto os processadores mais básicos contam com pelo menos um acumulador (AC), os modelos mais avançados podem ter dezenas ou até mesmo centenas de registradores para armazenar os resultados temporários das operações.

Processador simples com acumulador



Tamanho em bits da Arquitetura

- A largura da arquitetura de um processador (que pode ser de 8, 16, 32 ou 64 bits) é definida pela largura em bits do maior **operando inteiro** que pode ser processado de uma única vez (em um ciclo de máquina) pela UAL.
- Como consequência direta, os registradores têm a mesma largura em bits, para fornecer e receber os operandos também de uma única vez.

Tamanho em bits da Arquitetura

- A largura em bits da arquitetura de um processador:

  - **NÃO** é definida pelo tamanho em bits da instrução;
  - **NÃO** é definida pela largura do barramento de dados interno ou externo;
  - **NÃO** é definida pela largura em bits dos operandos da unidade de ponto flutuante;
  - **NÃO** é definida pela largura em bits do apontador de instruções (PC) ou do barramento de endereços.

Unidade de Controle

- Busca a instrução na memória e coloca no registrador de instruções (RI).
- Faz a decodificação da instrução que está no RI:
  - - Determina qual o tipo de operação vai ser realizada pela U.A.L.
  - - Determina quantos e quais são os operandos de leitura, e qual o registrador de destino, se houver.
- Lê os operandos necessários para a execução da instrução e os coloca na entrada da U.A.L.
- A unidade de controle lê o resultado da saída da U.A.L. e envia para o destino correto.

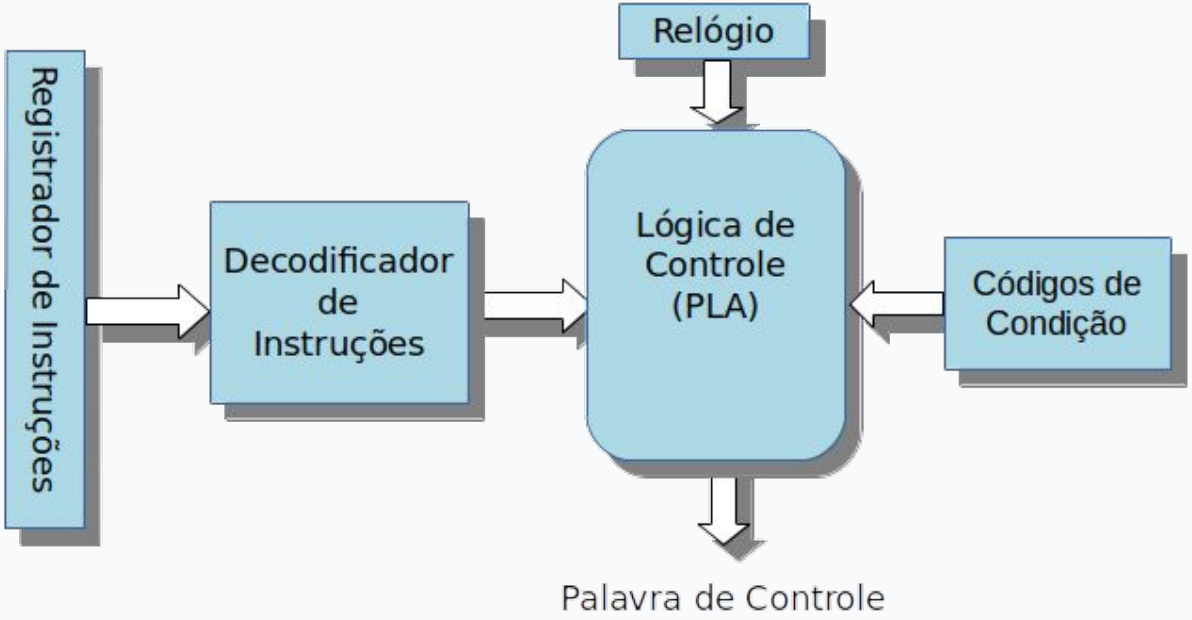
Unidade de Controle

- A unidade de controle gera um conjunto de sinais, chamado de palavra de controle, para comandar os circuitos lógicos responsáveis pelo funcionamento do processador.
- A operação desses circuitos é coordenada pelo relógio do processador que, junto com a palavra de controle, faz o acionamento de registradores, unidades funcionais, multiplexadores e outros circuitos para a correta execução das instruções.
- As unidades de controle podem ser de dois tipos: controladas por microprograma ou diretamente pelo hardware.

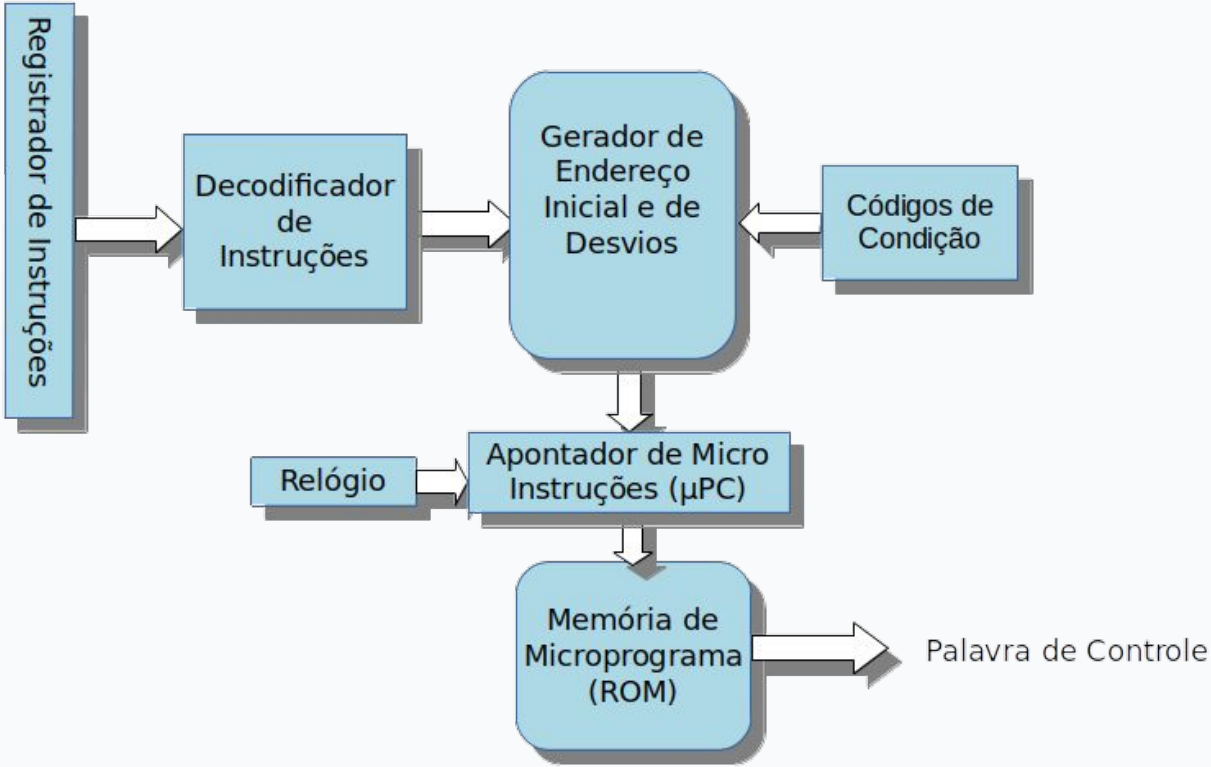
Unidade de Controle

- As unidades de controle **microprogramadas** são características das arquiteturas do tipo **CISC**.
- A complexidade das instruções demanda o uso de uma memória interna ao processador, que contém um microprograma responsável pela decodificação das instruções.
- O controle diretamente pelo **hardware** é encontrado normalmente nas arquiteturas do tipo **RISC**.
- Por ter instruções mais simples a decodificação das instruções pode ser feita diretamente em “hardware” com uso de uma PLA, que emula um circuito combinacional de portas lógicas.

Unidade de Controle por Hardware



Unidade de controle microprogramada



Modos de Endereçamento

- Em uma arquitetura de processador, os operandos das instruções podem ser constantes, ou variáveis armazenadas em registradores ou na memória.
- Alguns modos de endereçamento possíveis são:
  - - Imediato
  - - Registrador ou acumulador
  - - Direto
  - - Indireto via registrador
  - - Indireto via memória
  - - Indexado
  - - Pilha

Modos de Endereçamento

- **Imediato:** o operando é uma constante definida diretamente no código da instrução ou nos bytes subsequentes.
- **Registrador ou acumulador:** O operando está armazenado no acumulador ou em um dos registradores. O acumulador é referenciado de forma implícita, ou seja, não aparece no código da instrução de máquina. No caso do registrador, o número do registrador está definido no código da instrução.
- **Direto ou absoluto:** O operando está armazenado em um endereço de memória definido no código da instrução ou nos bytes subsequentes. É necessário realizar outro acesso para buscar o operando na memória.

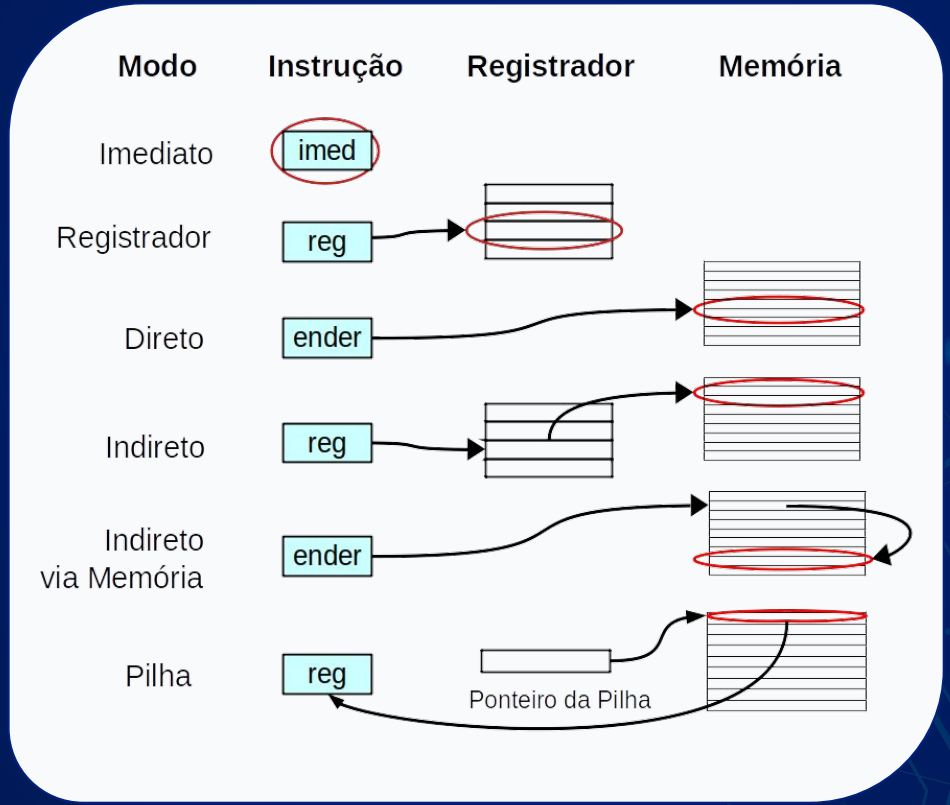
Modos de Endereçamento

- **Indireto via registrador:** o endereço do operando está armazenado no registrador definido no código da instrução. É necessário acessar este endereço de memória para ler/escrever o operando.
- **Indireto via memória:** o endereço de memória do operando está armazenado em um endereço de memória definido no código da instrução ou nos bytes subsequentes. São necessários dois acessos à memória: um para recuperar o endereço do operando e outro para buscar o operando em si.

Modos de Endereçamento

- **Indexado:** o endereço do operando é obtido pela soma do conteúdo de dois registradores definidos no código da instrução de máquina. É necessário acessar este endereço de memória para ler/escrever o operando.
- **Pilha:** o apontador de pilha (SP) aponta para uma estrutura em memória (uma pilha) onde o operando ou operandos são acessados de uma forma implícita, ou seja, os operandos de origem e destino da instrução estão, implicitamente, colocados no topo da pilha e não estão definidos no código da instrução.

Modos de Endereçamento



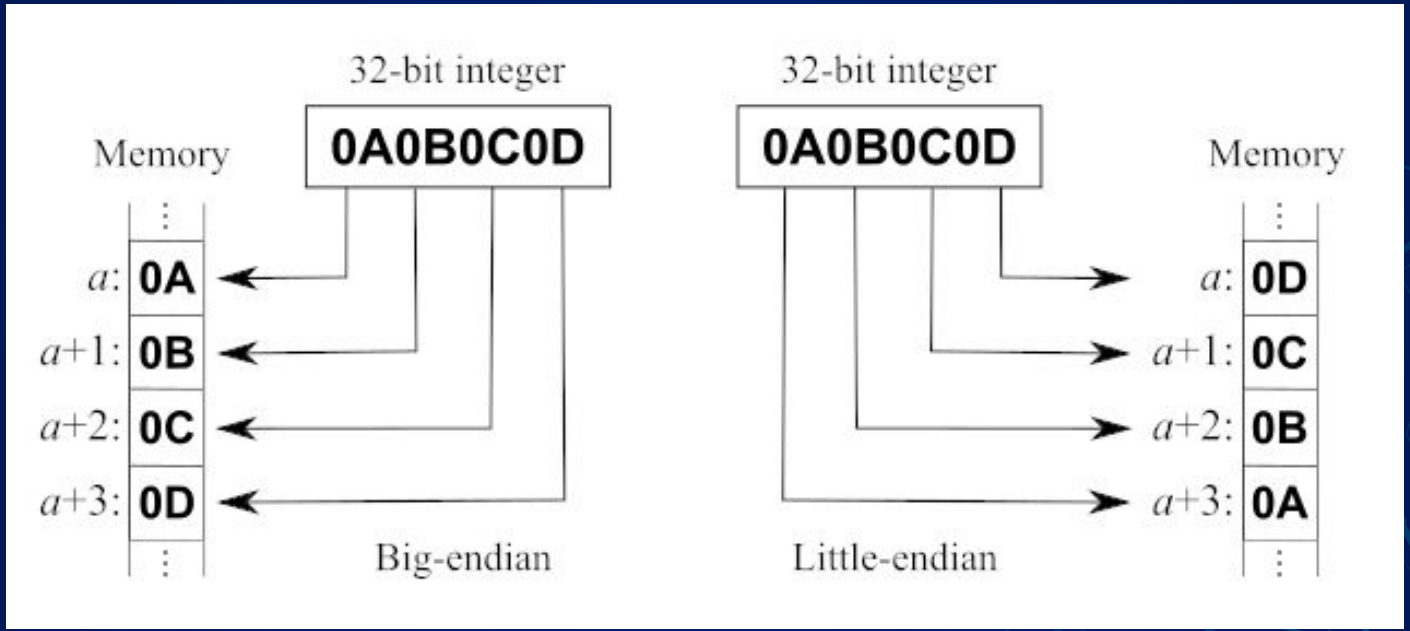
Ordenação dos Bytes na Memória

- Armazenar variáveis com tamanho maior que um “byte” na memória pode ser feita de duas maneiras diferentes, que são incompatíveis entre si.
- A ordenação “big-endian” armazena o byte mais significativo de um operando no menor endereço de memória e o byte menos significativo no maior endereço.
- A ordenação “little-endian”, por outro lado, armazena o byte menos significativo no menor endereço de memória e byte mais significativo no maior endereço.

Ordenação dos Bytes na Memória

- Historicamente, os computadores não se comunicavam e a troca de programas e dados entre eles era bastante esporádica.
- Assim, cada fabricante adotou um desses formatos, sem muitas preocupações de compatibilidade com os padrões de outros fabricantes.
- Contudo, com o surgimento da internet, foi necessário definir um padrão para que essa troca de dados fosse possível.
- Assim, computadores com processadores de ordenação diferente, ao utilizar a internet para se comunicarem, devem converter seus dados para o formato “big-endian”.

Ordenação dos bytes na Memória



https://www.wikiwand.com/en/Endianness

3.4 TIPOS DE ARQUITETURA DE PROCESSADOR

Tipos de Arquitetura

- Os processadores podem ser construídos de formas diferentes, que se distinguem principalmente pela maneira como os operandos das instruções são acessados e armazenados internamente. Os principais tipos de arquitetura são:
  - - Acumulador
  - - Pilha
  - - Registrador-Memória
  - - Memória-Memória
  - - Registrador-Registrador (Load-Store)

Tipos de Arquitetura

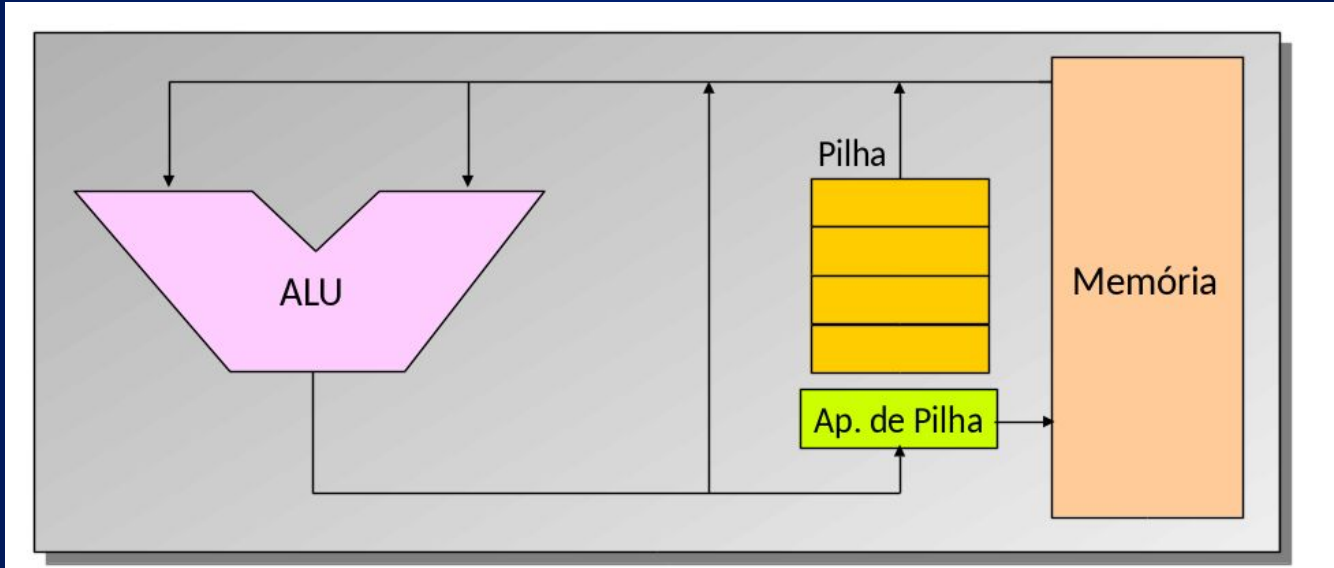
- Acumulador: um operando (em registrador ou memória), o acumulador é usado como operando implícito a maioria das vezes.
- Pilha: nenhum operando. Todos operandos são implícitos no topo da pilha.
- Registrador (load / store): três operandos, todos nos registradores. Loads e stores são as únicas instruções que fazem acesso à memória
- Registrador-Memória: dois operandos, um em memória.
- Memória-Memória: três operandos, podem todos estar na memória.

Tipos de Arquitetura

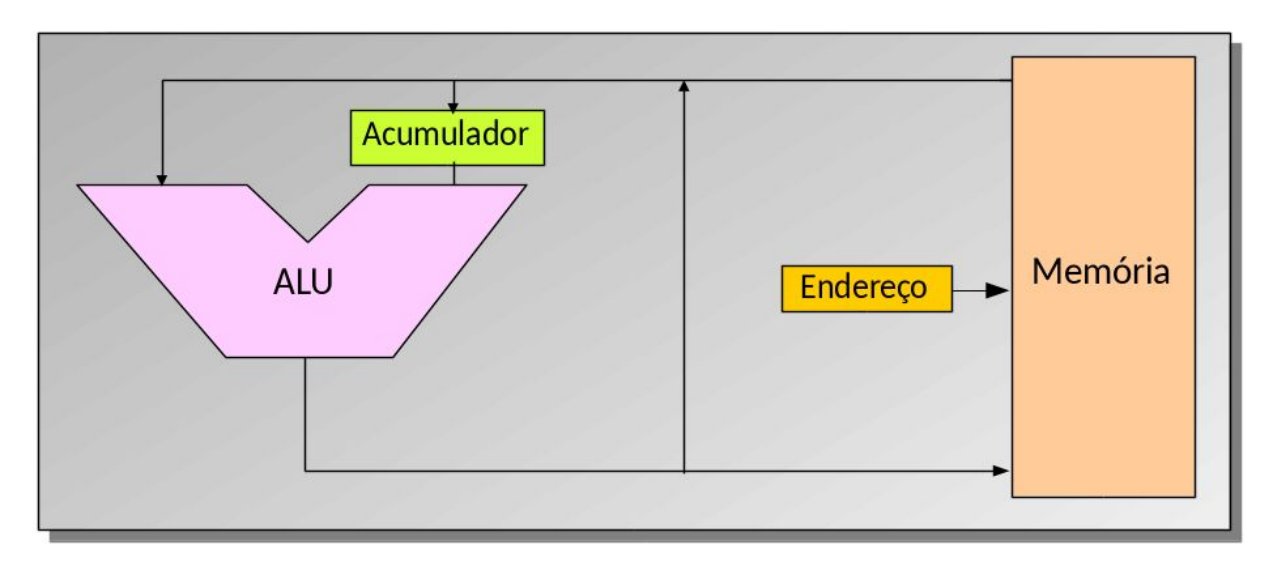
C := A + B;

Classificação				
Pilha	Acumulador	Memória-Memória	Registrador-Memória	Registrador-Registrador
PUSH A	LOAD A	ADD C, B, A	LOAD R1, A	LOAD R1, A
PUSH B	ADD B		ADD R1, B	LOAD R2, B
ADD	STORE C		STORE C, R1	ADD R3, R1, R2
POP C				STORE C, R3

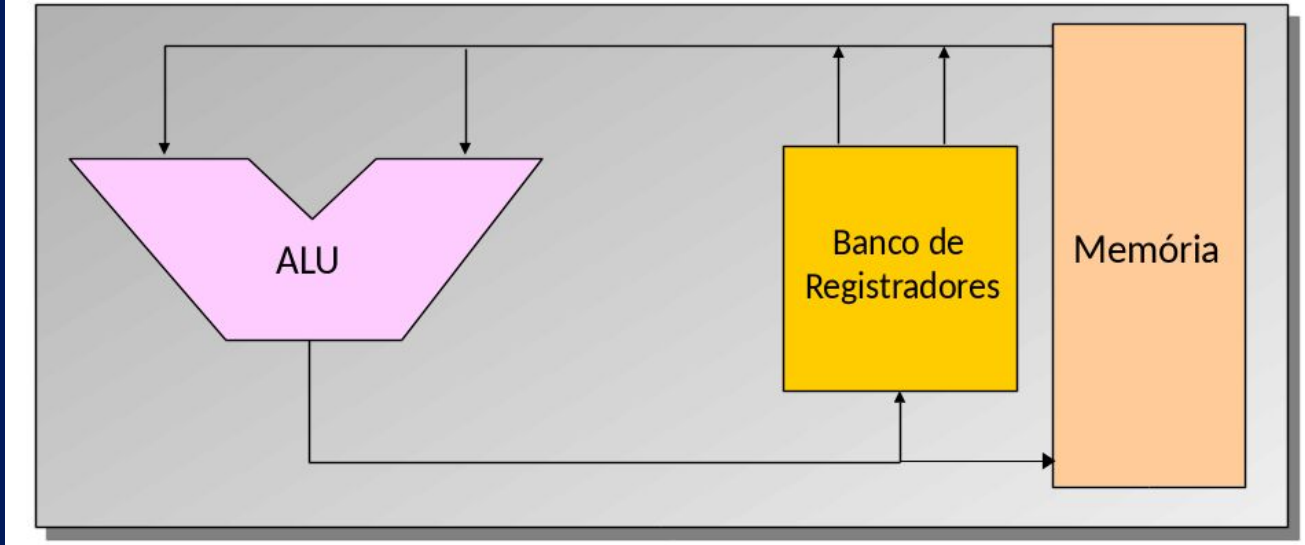
Arquitetura de Pilha



Arquitetura de Acumulador



Arquitetura de Registrador



3.5 RISC x CISC

Tempo de Execução de um Programa

$$T_p = C_i \times T_c \times N_i \quad (3.1)$$

- T_p = tempo de execução do programa
- C_i = ciclos gastos em média por instrução
- T_c = tempo de duração de um ciclo
- N_i = número de instruções do programa

Arquiteturas CISC

- Uma forma de melhorar o desempenho é diminuir o número total de instruções (Ni) gastas para traduzir um programa da linguagem de alto nível para a linguagem de máquina.
- Assim sendo, instruções cada vez mais complexas eram implementadas no conjunto de instruções do processador.
- Essa aproximação foi utilizada pelas arquiteturas CISC (Complex Instruction Set Computer), utilizada nas primeiras arquiteturas de processadores.

Arquiteturas RISC

- No início da década de 80, surgiu nas universidades americanas (Berkley e Stanford especificamente) uma nova proposta de arquitetura para os processadores.
- Nesse novo conceito, buscava-se reduzir o tempo de execução dos programas através da diminuição do número de ciclos gastos para executar cada instrução e também no tempo de duração de cada ciclo de máquina, com uso de um conjunto de instruções o mais simples possível.
- Essa abordagem foi utilizada pelas arquiteturas RISC (Reduced Instruction Set Computer), utilizada nos modernos processadores.

Características das Arquiteturas CISC

- Instruções complexas demandando um número grande e variável de ciclos de máquina para sua execução.
- Uso de diversos modos de endereçamento de operandos.
- Instruções com formato muito variável.
- Diferentes tipos de instruções podem referenciar operandos na memória principal.
- Cada fase do processamento da instrução pode ter duração variável em função da complexidade.

Características das Arquiteturas RISC

- Instruções mais simples demandando um número fixo de ciclos de máquina para sua execução.
- Uso de poucos modos simples de endereçamento de operandos.
- Poucos formatos diferentes de instruções.
- Apenas as instruções de “load” e “store” referenciam operandos na memória principal.
- Cada fase de processamento da instrução tem a duração fixa igual a um ciclo de máquina.

Consequências das Arquiteturas CISC

- Implementação com uso de pipeline é difícil. **(-)**
- A taxa média de execução das instruções era bastante superior a 1 CPI. **(-)**
- A unidade de controle é em geral microprogramada. **(-)**
- Códigos compactos podem ser gerados pelos compiladores. **(+)**

Consequências das Arquiteturas RISC

- Implementadas facilmente para uso do pipeline. **(+)**
- A taxa média de execução de instruções de máquina é próxima ou menor que 1 CPI. **(+)**
- A unidade de controle é em geral “hardwired”. **(+)**
- Processo de compilação é complexo e requer cuidados especiais para otimização do desempenho do código gerado. **(-)**

3.6 Processador Sapiens

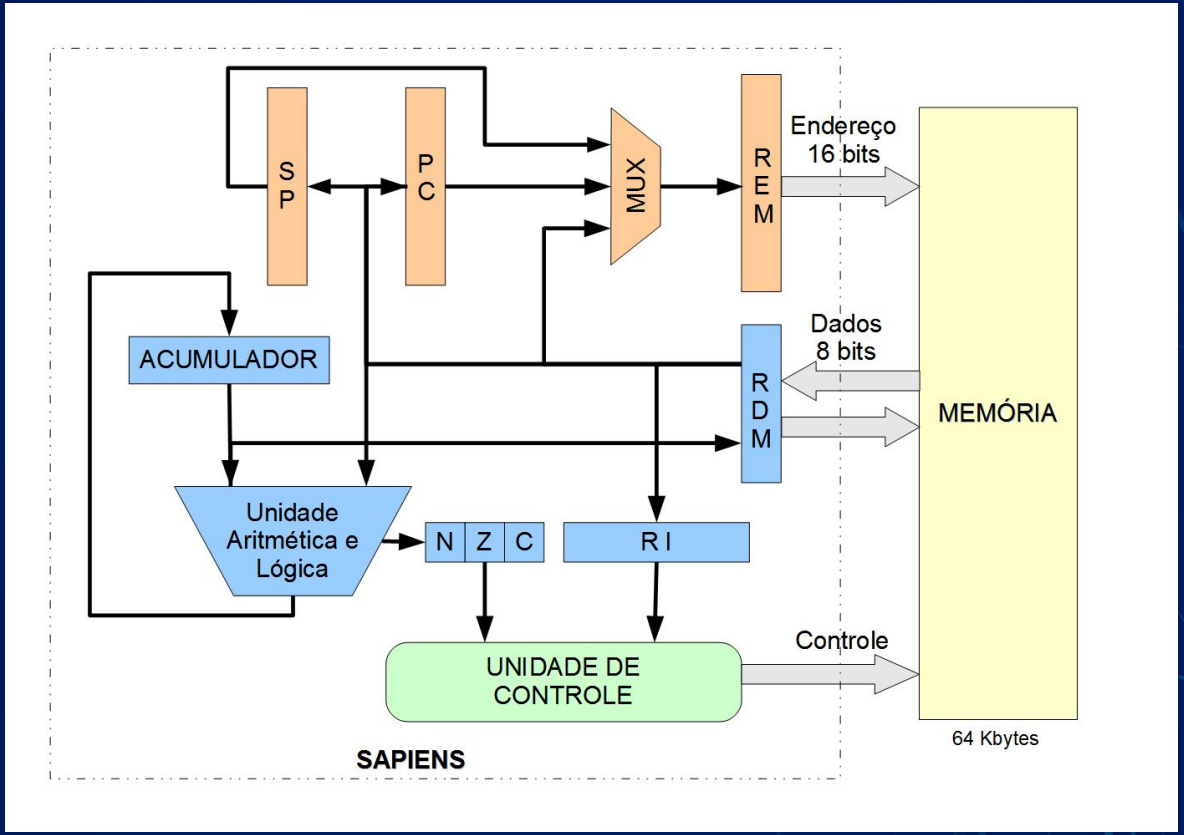
Processador Sapiens

- O processador Sapiens é um processador didático desenvolvido para introduzir de forma gradual os aspectos centrais do funcionamento e programação de um processador.
- Suas principais características são:
  - - Arquitetura de acumulador de 8 bits.
  - - Código de instrução com 8 bits.
  - - Apontador de instruções (PC) de 16 bits.
  - - Apontador de pilha (SP), também de 16 bits.
  - - Códigos de condição C (carry), N (negativo) e Z (zero).

Processador Sapiens

- Suas principais características são:
  - - Modos imediato, direto e indireto de acesso aos operandos.
  - - A ordenação dos bytes na memória é little-endian.
  - - Possui instruções de IN e OUT para realizar operações de entrada e saída.
  - - Instrução de TRAP para realizar operações mais elaboradas de E/S.
  - - Instruções de movimentação de pilha, deslocamento do registrador, soma e subtração com vai-um/vem-um, entre outras.

Processador Sapiens



Formato da Instrução

A instrução pode ter um tamanho entre um a 3 bytes de comprimento, dependendo do modo de endereçamento dos operandos

Conjunto de Instruções

Instruções Aritméticas	
Instrução	Ciclo de Execução
ADD ender ADD @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] AC ← AC + MEM(REM) PC ← PC + 2 Atualiza N, Z, C
ADD #imed	AC ← AC + MEM(PC) PC ← PC + 1 Atualiza N, Z, C
ADC ender ADC @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] AC ← AC + MEM(REM) + C PC ← PC + 2 Atualiza N, Z, C
ADC #imed	AC ← AC + MEM(PC) + C PC ← PC + 1 Atualiza N, Z e C
SUB ender SUB @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] AC ← AC - MEM(REM) PC ← PC + 2 Atualiza N, Z , C
SUB #imed	AC ← AC - MEM(PC) PC ← PC + 1 Atualiza N, Z, C
SBC ender SBC @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] AC ← AC - MEM(REM) - C PC ← PC + 2 Atualiza N, Z, C
SBC #imed	AC ← AC - MEM(PC) - C PC ← PC + 1 Atualiza N, Z, C

Instruções de soma, soma com carry, subtração e subtração com borrow, com operandos endereçados no modo imediato, direto e indireto.

Instruções Lógicas	
Instrução	Ciclo de Execução
OR ender OR @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] AC ← AC or MEM(REM) PC ← PC + 2 Atualiza N, Z
OR #imed	AC ← AC or MEM(PC) PC ← PC + 1 Atualiza N, Z
XOR ender XOR @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] AC ← AC xor MEM(REM) PC ← PC + 2 Atualiza N, Z
XOR #imed	AC ← AC xor MEM(PC) PC ← PC + 1 Atualiza N, Z
AND ender AND @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] AC ← AC and MEM(REM) PC ← PC + 2 Atualiza N, Z
AND #imed	AC ← AC and MEM(PC) PC ← PC + 1 Atualiza N, Z
NOT	AC ← not AC Atualiza N, Z
SHL	AC ← AC << 1; C = AC(7); AC(0) = 0 Atualiza N, Z e C
SHR	AC ← AC >> 1; AC(7) = 0; C = AC(0) Atualiza N, Z , C
SRA	AC ← AC >> 1; AC(6) = AC(7) ; AC(7) = AC(7); C = AC(0) Atualiza N, Z , C

Conjunto de Instruções

Instruções de ‘ou’, ‘ou exclusivo’, ‘e’, com operandos endereçados no modo imediato, direto e indireto, além de ‘not’, deslocamento à esquerda, deslocamento lógico à direita e deslocamento aritmético à direita, sobre o dado no acumulador.

Instruções de Transferência de Controle	
Instrução	Ciclo de Execução
JMP ender JMP @ender	REM ← MEM(PC)[1] Se modo indireto PC ← MEM(REM)[1] Senão PC ← PC + 2
JN ender JN @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] if N = 1 then PC ← REM else PC ← PC + 2
JP ender JP @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] if (N = 0 and Z = 0) then PC ← REM else PC ← PC + 2
JZ ender JZ @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] if Z = 1 then PC ← REM else PC ← PC + 2
JNZ ender JNZ @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] if Z = 0 then PC ← REM else PC ← PC + 2
JC ender JC @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] if C = 1 then PC ← REM else PC ← PC + 2
JNC ender JNC @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] if C = 0 then PC ← REM else PC ← PC + 2
JSR ender JSR @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] SP ← SP – 2 MEM(SP) ← PC[1] PC ← REM
RET	PC ← MEM(SP)[1] SP ← SP + 2

Conjunto de Instruções

Instruções de desvio incondicional, desvios condicionais: se negativo, se positivo, se zero, se não zero, se carry, se não carry, chamada de subrotina e retorno de subrotina.

Conjunto de Instruções

Instruções de Transferência de Dados	
Instrução	Ciclo de Execução
STA ender STA @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] MEM(REM) ← AC PC ← PC + 2
STS ender STS @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] MEM(REM) ← SP[1] PC ← PC + 2
LDA ender LDA @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] AC ← MEM(REM) PC ← PC + 2 Atualiza N, Z
LDA #imed	AC ← MEM(PC)[1] PC ← PC + 2 Atualiza N, Z e C
LDS ender LDS @ender	REM ← MEM(PC)[1] Se modo indireto REM ← MEM(REM)[1] SP ← MEM(REM)[1] PC ← PC + 2
LDS #imed	SP ← MEM(PC)[1] PC ← PC + 2
IN ender	REM ← MEM(PC)[1] AC ← IO (REM) PC ← PC + 2
OUT ender	REM ← MEM(PC)[1] IO (REM) ← AC PC ← PC + 2
PUSH	SP ← SP - 1 MEM(SP) ← AC
POP	AC ← MEM(SP) SP ← SP + 1 Atualiza N, Z

Instruções de armazenamento, soma com carry, subtração e subtração com borrow, com operandos endereçados no modo imediato, direto e indireto.

Conjunto de Instruções

Instruções vs Códigos de Condição	
Instrução	**Código de Condição**
LDA	N, Z
ADD	N, Z, C
SUB	N, Z, C
ADC	N, Z, C
SBC	N, Z, C
AND	N, Z
OR	N, Z
XOR	N, Z
NOT	N, X
SHR	N, Z, C *
SRA	N, Z, C *
SHL	N, Z, C **
POP	N, Z

As instruções que afetam os códigos de condição estão listadas ao lado.
As demais instruções não modificam o valor dos códigos de condição.

* - a flag de carry recebe o bit menos significativo do acumulador.
** - a flag de carry recebe o bit mais significativo do acumulador.

Conjunto de Instruções

- Além dessas, temos as seguintes instruções especiais:
  - - **NOP**: que não faz nada.
  - - **HALT**: que pára a execução do programa.
  - - Ambas não tem operandos.
  - - **TRAP**: utilizada para emulação de rotinas de E/S pelo simulador SapienS. O código do tipo de serviço é passado como parâmetro no acumulador. Os dois bytes seguintes à instrução definem o endereço, no modo direto ou indireto, dos parâmetros adicionais para a instrução.

Conjunto de Instruções

- TRAP (código do tipo de serviço):
  - # 1 – Leitura de um caractere da console.
  - # 2 – Escrita de um caractere ASCII na console.
  - # 3 – Leitura de uma linha inteira da console.
  - # 4 – Escrita de uma linha inteira na console.
  - # 5 – Rotina de temporização em milissegundos.
  - # 6 – Rotina para tocar um tom.
  - # 7 – Rotina para retornar um número pseudo aleatório entre 0 e 99 no acumulador.
  - # 8 – Semente inicial da rotina de número aleatórios.

Operações de E/S

- Os dispositivos virtuais de entrada e saída do simulador SimuS são acessíveis com uso das instruções IN e OUT.
- Alguns desses dispositivos compartilham o mesmo endereço de E/S, sendo diferenciados apenas pela operação de entrada ou saída.

- Painel de chaves: **IN 00**
- Display hexadecimal: **OUT 00**
- Status do painel de chaves: **IN 01**
- Visor de 16 caracteres: **OUT 02**
- Teclado: **IN 02**
- Status do teclado: **IN 03**
- Limpeza do visor: **OUT 03**

Obrigado !

Arquitetura e Organização de Computadores: Uma Introdução

Mais recursos em:
https://simulador-simus.github.io