Conceitos Básicos de Arquitetura de Computadores

Preprint · October 2020

DOI: 10.13140/RG.2.2.34872.78089

2 authors:

José Antonio Dos Santos Borges
Federal University of Rio de Janeiro
53 PUBLICATIONS **130** CITATIONS
SEE PROFILE

Gabriel P. Silva
Federal University of Rio de Janeiro
66 PUBLICATIONS **45** CITATIONS
SEE PROFILE

3

Conceitos Básicos de Arquitetura de Computadores

"Conto os versos de um poema, calculo a altura de uma estrela, avalio o número de franjas, meço a área de um país, ou a força de uma torrente – aplico, enfim, fórmulas algébricas e princípios geométricos – sem me preocupar com os louros que possa tirar de meus cálculos e estudos!"

– Malba Tahan, *O Homem que Calculava*

De uma forma simplificada, o computador é um dispositivo eletrônico cuja principal finalidade é o processamento da informação (na forma de textos, números, imagens e sons) conforme especificado pelo usuário com a utilização de um programa.

Um programa é um conjunto de instruções e dados em formato binário que são carregados na memória do computador, que definem uma forma de interação com o usuário e especificam as tarefas que serão realizadas pelo computador.

Os programas são inicialmente definidos pelos programadores com o uso de uma linguagem de alto nível (como C, Python, Fortran e outras) e que são posteriormente traduzidos pelo compilador para a linguagem de máquina, que é a linguagem que o processador entende.

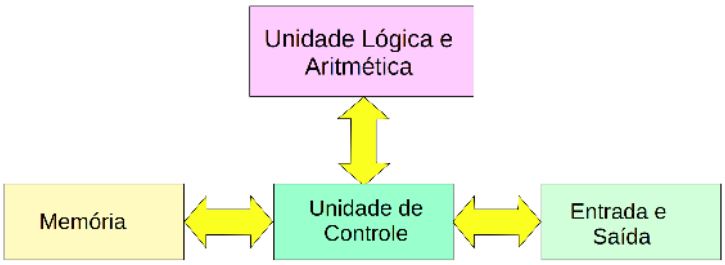
A seguir vamos ver, modernamente, como isso tudo começou.

3.1 ARQUITETURA DE VON NEUMANN

Para entendermos a importância da arquitetura Von Neumann vamos começar fazendo uma pergunta: qual a diferença entre uma calculadora e um computador?

Uma calculadora básica realiza apenas as funções pré-determinadas em seu teclado. Se desejarmos fazer um novo tipo de operação, isto só será possível com a modificação dos circuitos eletrônicos que compõem a calculadora, além do seu teclado, para a inclusão da nova função. Não há, portanto, flexibilidade para realizar alterações na calculadora para se adaptar a novas aplicações.

Figura 3.1 – Arquitetura de Von Neumann



O computador, por sua vez, é um equipamento que oferece a possibilidade de ser configurado facilmente para novas tarefas, de acordo com as necessidades de cada aplicação que for requerida pelo usuário.

A grande inovação da proposta de Von Neumann foi uma nova forma de arquitetura para o computador que permitisse um alto grau de flexibilidade, de forma a adaptá-lo facilmente para diversas aplicações.

O conceito de programa armazenado foi um dos conceitos fundamentais apresentados por Von Neumann que permitiu essa flexibilidade. Em seu modelo de computador foi introduzido o conceito de memória, um dispositivo de armazenamento temporário, para onde programas (instruções e dados) diferentes poderiam ser carregados a partir de uma unidade de entrada, para serem executados pela unidade aritmética e lógica, com os resultados sendo transferidos da memória para uma unidade de saída, tudo isso sob a coordenação de uma unidade de controle. Deste modo, ficava garantida a flexibilidade do computador, que pode ter o seu funcionamento facilmente alterado mediante o uso de instruções e dados diferentes, de acordo com a aplicação de cada usuário.

Sendo mais formal, os componentes da máquina de Von Neumann (Figura 3.1) podem ser descritos assim:

- **Memória**: É a unidade onde as instruções, os dados de entrada, as tabelas de referência, e os resultados intermediários são armazenados para permitir a execução de um programa.
- **Controle**: É a unidade responsável pelo sequenciamento das operações e pelo controle das demais unidades do computador.
- **Aritmética e Lógica**: É a unidade que irá executar as operações aritméticas e lógicas tais como: soma, subtração, multiplicação, divisão, raiz quadrada, movimentação entre a unidade aritmética e a memória, verificação do sinal do resultado, conversão de decimal para binário e vice-versa. Um total de 10 operações fundamentais foi definido por Von Neumann. É chamada abreviadamente de UAL.

- **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.

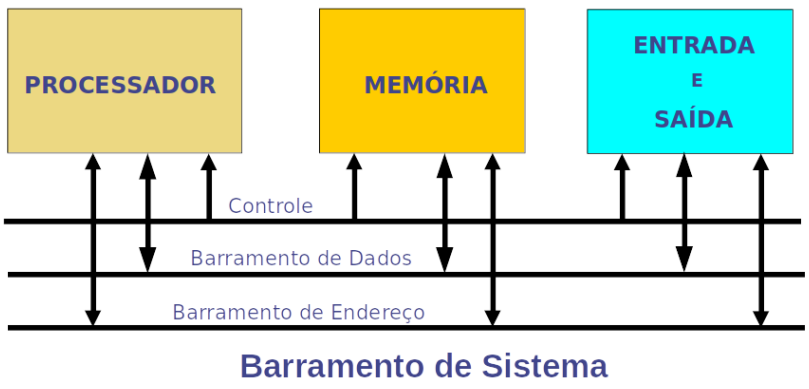
- **Saída**: É a unidade que transfere a informação (numérica ou não) para o meio externo. Todas as transferências devem ser feitas da memória para o meio externo, e nunca diretamente da unidade de controle.

Von Neumann, em uma analogia com o comportamento dos neurônios, sugere o uso da numeração binária para a representação interna dos números, ao invés da numeração decimal, pela evidente economia que isto proporciona no tempo gasto nos cálculos e na complexidade dos circuitos. As válvulas foram escolhidas como elementos básicos por serem dispositivos com o menor tempo de chaveamento (mudança do valor lógico 0 para o valor lógico 1) existentes com a tecnologia disponível naquela época. O uso de um sinal elétrico periódico para cadenciar todas as operações do computador foi também proposto, dando origem ao que chamamos de relógio do computador.

Este modelo de arquitetura proposto por Von Neumann continua sendo utilizado no projeto dos computadores comerciais nos dias de hoje. O estudo de suas características permite uma compreensão adequada do funcionamento dos computadores utilizados hoje em dia.

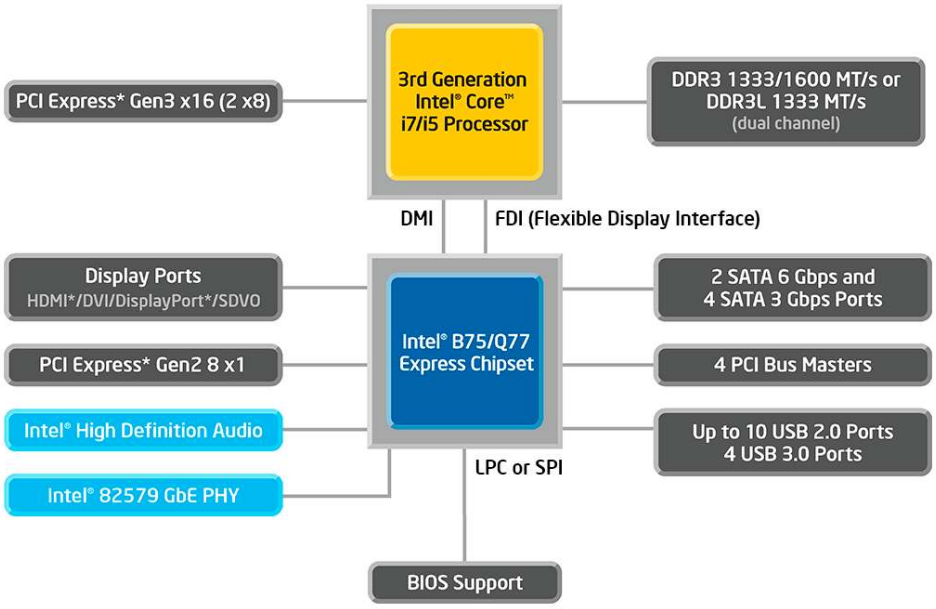
3.2 MODELO DE BARRAMENTO DE SISTEMA

Figura 3.2 – Modelo de Barramento de Sistema



O modelo de Von Neumann passou por um refinamento que recebeu o nome de modelo de barramento de sistema (Figura 3.2). Nesse modelo, a unidade de controle e a unidade aritmética são agregadas, junto com os registradores, em um único elemento que recebe o nome de processador. As unidades de entrada e saída são vistas agora também como uma única unidade, chamada de unidade de entrada/saída. A memória continua sendo vista com

Figura 3.3 – Placa de Processador Moderno



uma unidade independente, com as mesmas funções da arquitetura de Von Neumann, ou seja, armazenamento de dados e instruções dos programas em execução.

Um elemento novo que surge é o próprio barramento de sistema, que faz a interligação entre o processador, a memória e a unidade de entrada/saída. 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 até a memória. Sinais estes que vão, principalmente, determinar qual a posição de memória que irá ser lida ou escrita. Os endereços podem ser fornecidos tanto pelo processador como pela unidade de entrada/saída. A informação dessa posição de memória, que está sendo lida ou escrita na memória, transita pelo barramento de dados, que é bidirecional. Apesar do nome, tanto instruções como os dados propriamente ditos circulam por esse barramento. O barramento de controle indica qual a natureza da operação que vai ser realizada: leitura ou escrita, na maior parte dos casos, e possui também sinais para a arbitragem do barramento, para determinar quem vai utilizar o barramento naquele momento, que pode ser tanto o processador como a unidade de entrada/saída.

Eventualmente, nos modernos computadores, existe também um barramento dedicado para ligar os periféricos à unidade de entrada/saída. Isso permite que o acesso do processador à memória se faça com maior eficiência, pela diminuição do tráfego de dados no barramento de sistema.

As placas-mãe dos computadores pessoais modernos não possuem um barramento de sistema explícito, mas um conjunto de controladores que faz a intermediação entre a memória, o processador e os dispositivos de entrada e saída, como pode ser visto na Figura 3.3. Neste tipo de arquitetura o controlador de memória está embutido no próprio processador, além da

interface para acesso à placa de vídeo. Um terceiro barramento faz a interface do processador com um *chip* controlador que realiza a interface com diversos periféricos e barramentos de E/S. Nesse *chip* também encontramos embutidos o controlador de DMA, o controlador de interrupção, temporizadores e relógio de tempo real.

A seguir vamos examinar cada um dos componentes do modelo de barramento de sistema com mais detalhes.

3.2.1 Processador

No processador, além da unidade aritmética e lógica e da unidade de controle, encontramos também os registradores. Os registradores são elementos de memória, de pequena capacidade, mas de alta velocidade, colocados junto da UAL para armazenar os valores que vão ser utilizados como operandos e receber os resultados gerados pela UAL. Se os operandos e resultados tivessem fossem armazenados diretamente na memória, o tempo para a realização das operações da UAL aumentaria tremendamente. O conjunto desses registradores é denominado banco de registradores.

Os registradores são referenciados explicitamente pelas instruções lógicas, aritméticas e de transferência de dados. Existe um registrador invisível ao programador, chamado de registrador de instrução (RI), que armazena a instrução que está sendo executada. Existe um registrador especial denominado apontador de instruções (PC), que contém o endereço da próxima instrução que vai ser executada.

A função do processador é executar os programas que estão armazenados na memória principal. Isso é feito buscando suas instruções, examinando-as, e então executando-as uma após a outra. O processador é responsável pela realização de uma série de funções, dentre as quais destacamos:

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

Na Figura 3.4 podemos ver a fotografia de um processador de 64 bits utilizado nos modernos computadores.

O processador tem seu funcionamento também sincronizado pelo relógio, que pode operar em frequência diferente dos demais componentes do computador, para cadenciar a execução

Figura 3.4 – Processador Intel Core I7

das instruções em suas diversas fases. Quanto mais rápido (maior a frequência) for o sinal de relógio, mais rápido as instruções, e por consequência os programas, serão executados. Os componentes básicos do processador (portas lógicas, flip-flops, etc.) e a potência máxima que consegue ser dissipada pelo encapsulamento, limitam a frequência máxima que o relógio pode ter.

Maiores detalhes sobre o funcionamento do processador podem ser vistos na Capítulo 3.3.

3.2.2 Memória

A memória principal é utilizada para armazenar os programas e dados que vão ser processados durante a operação normal do computador. As informações armazenadas na memória principal podem ser alteradas durante a execução de um programa.

A menor unidade de informação que pode ser manipulada na memória é o byte, que é um conjunto de 8 bits. Cada byte na memória possui um endereço distinto para que possa ser referenciado, ou seja, o seu conteúdo possa ser lido ou escrito. Para que a informação possa ser lida ou escrita na memória, deve ser acompanhada de um endereço, que é fornecido pelo processador ou pela unidade de entrada/saída.

Os dados ou instruções armazenados na memória são transferidos de/para o processador pelo barramento de dados. O número de bits do barramento de dados varia de acordo com o tipo de pastilha ou cartão de memória utilizado para compor a memória principal do processador.

A unidade de memória do computador é formada uma memória volátil, chamada de memória principal, e por outra não volátil. A memória não volátil, que mantém o seu conteúdo quando o computador é desligado, é utilizada para iniciar o funcionamento do computador, realizando os testes iniciais e cópia do sistema operacional para a memória principal. Nos primeiros computadores compatíveis com o IBM/PC esta memória recebia o nome do programa que era nela armazenado: BIOS (Basic Input/Output System). Embora seja não volátil, nos modernos computadores a memória que contém a BIOS pode ser atualizada diretamente no

computador, mediante reprogramação de seu conteúdo, mas isto deve ser necessário apenas eventualmente.

Maiores detalhes sobre a memória serão apresentados no Capítulo 4.

3.2.3 Entrada e Saída

A unidade de entrada e saída contém os circuitos de interface necessários para permitir a comunicação entre os dispositivos de entrada/saída (também chamados de periféricos) com as demais partes do computador. As unidades de entrada/saída (E/S) fazem a conversão da informação eletrônica no formato binário para os meios externos e vice-versa. Exemplos de dispositivos de entrada/saída são o teclado, terminal de vídeo, mouse, impressora, disco rígido, cd-rom, entre outros.

Depois que os resultados finais são calculados por um programa, eles são transferidos da memória para os dispositivos de entrada e saída, quando serão traduzidos para o ser humano na forma de caracteres, imagens ou sons; armazenados em dispositivos como discos ópticos ou magnéticos; ou transmitidos para outros computadores e dispositivos pela internet. A interação do processador com os dispositivos de entrada e saída se dá de duas formas básicas:

- por *loop de status*
- por interrupção

Nos processadores mais simples a única forma de comunicação disponível é por *loop de status*, onde o processador verifica periodicamente a situação dos dispositivos de entrada e saída, normalmente através de instruções especiais de E/S acessando registradores em um espaço de endereçamento reservado para os dispositivos de E/S. Esses registradores indicam quando o processador pode transferir os dados da memória para esses dispositivos de E/S e vice-versa, e eles são atualizados adequadamente pelos dispositivos de E/S conforme o desenrolar dessas operações de E/S.

Nos processadores mais modernos, linhas de interrupção conectam o processador aos dispositivos de E/S. Essas linhas são ativadas para informar ao processador o término das operações de E/S que foram solicitadas previamente. A sua ativação faz com que o processador interrompa a execução do programa atual, desviando automaticamente para rotinas especiais que vão realizar o tratamento adequado desta interrupção.

Sem os periféricos a comunicação entre o homem e o computador não seria possível. Os dispositivos de entrada/saída que são responsáveis pelo armazenamento de informação de uma forma não volátil (meio magnético ou ótico, por exemplo) são chamados de memória secundária ou memória de massa. Toda informação que precisa ser mantida depois que o computador for desligado é guardada nesses dispositivos.

A memória secundária é onde os programas e dados, incluindo aqueles do sistema operaci-

onal, são armazenados de uma forma persistente no computador. Hoje em dia é constituída, principalmente, pelo conjunto de discos magnéticos do computador e também, cada vez mais, pelos discos de estado sólido. Há vários outros tipos de dispositivos, removíveis ou não, que podem ser considerados parte da memória secundária, tais como fitas magnéticas, discos óticos, entre outros. 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.

Maiores detalhes sobre a memória e a hierarquia de memória podem ser encontrados no Capítulo 5.

3.3 FUNCIONAMENTO DO 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 pelo usuário.

Embora existam diversos tipos de processadores, com diferentes conjuntos de instruções em linguagem de máquina e com variados níveis de sofisticação, podemos afirmar com segurança que o funcionamento básico de todos eles é o mesmo como definido por Von Neumann (Fig. 3.1) há mais de cinquenta anos.

Para executar um programa que está na memória, o processador realiza permanentemente a busca de instruções, no endereço de memória indicado pelo apontador de instruções (PC). Depois de lida da memória, a instrução é transferida para o registrador de instrução (RI) e o valor do apontador de instruções é automaticamente incrementado para o endereço da instrução seguinte. Caso a instrução executada seja uma instrução de desvio, o valor do PC é alterado para o endereço especificado por essa instrução de desvio.

As instruções podem ler ou escrever dados (chamados de operandos), que podem ser acessados de diversas maneiras (chamados modos de endereçamento) em posições distintas na memória computador.

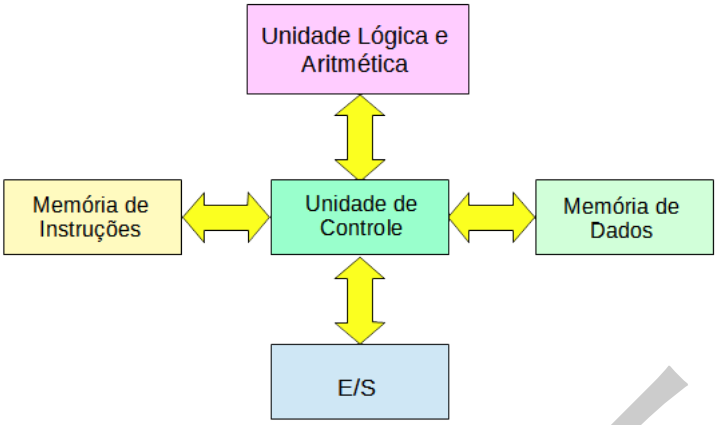
Repare que fica tudo armazenado na memória: em princípio dados e instruções podem ficar na mesma memória, sem conflito, desde que estejam em endereços diferentes. Há algumas situações em que é conveniente colocar dados e instruções em memórias diferentes, esse tipo especial de organização recebe o nome especial de arquitetura *Harvard*(Fig. 3.5).

Para o processamento das instruções o processador conta com um circuito especial, chamado de unidade aritmética e lógica (UAL), capaz de realizar operações aritméticas como soma e subtração com os operandos das instruções, sempre no formato binário. A fim de agilizar essas operações o processador possui internamente dispositivos especiais de armazenamento, chamados de registradores ou acumuladores, para guardar os resultados temporários dessas operações.

A unidade aritmética e lógica pode realizar diversas operações, entre elas:

- Adição

Figura 3.5 – Arquitetura Harvard



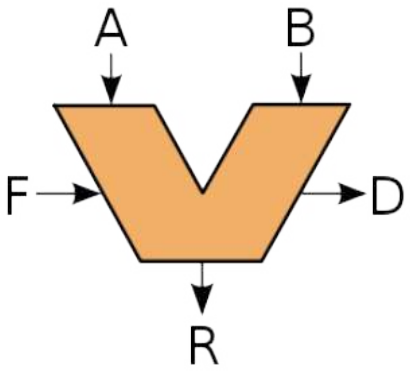
- Subtração
- Multiplicação
- Divisão
- Operações lógicas (E, OU, OU EXCLUSIVO, INVERSÃO, etc.).
- Deslocamento e Rotação (à esquerda e à direita)
- Comparação

As operações aritméticas e lógicas são realizadas pela leitura dos operandos de leitura dos registradores ou memória e com a escrita do resultado no registrador de destino ou memória. Os registradores são referenciados explicitamente pelas instruções e os endereços de memória são constantes embutidas na própria instrução ou valores contidos em registradores.

A largura da arquitetura de um processador (8, 16, 32 ou 64 bits) é definida pela largura em bits do maior operando inteiro que pode ser utilizado em uma única operação pela UAL. Note que a largura de uma arquitetura:

- **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.

Figura 3.6 – Unidade Aritmética e Lógica



Como consequência direta, a largura em bits do maior operando admitido pela UAL irá determinar, normalmente, a largura em bits do acumulador e dos registradores de uso geral do processador. Não há sentido para que sejam maiores ou menores do que isso.

Uma representação esquemática da unidade aritmética e lógica pode ser vista na Figura 3.6.

A unidade de controle é responsável pela coordenação da atividade de todos os componentes do processador, realizando também busca da instrução na memória e transferindo-da para o registrador de instruções (RI). A unidade de controle 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.

Há duas formas de se implementar a unidade de controle:

- Através de microprogramação
- Controle direto pelo hardware (PLA, ROM)

As unidades de controle microprogramadas são características das arquiteturas do tipo CISC. O controle diretamente pelo hardware é encontrado normalmente nas arquiteturas do tipo RISC (veja a seção 3.7).

O processador executa uma instrução em uma série de etapas:

Figura 3.7 – Unidade de Controle Microprogramada

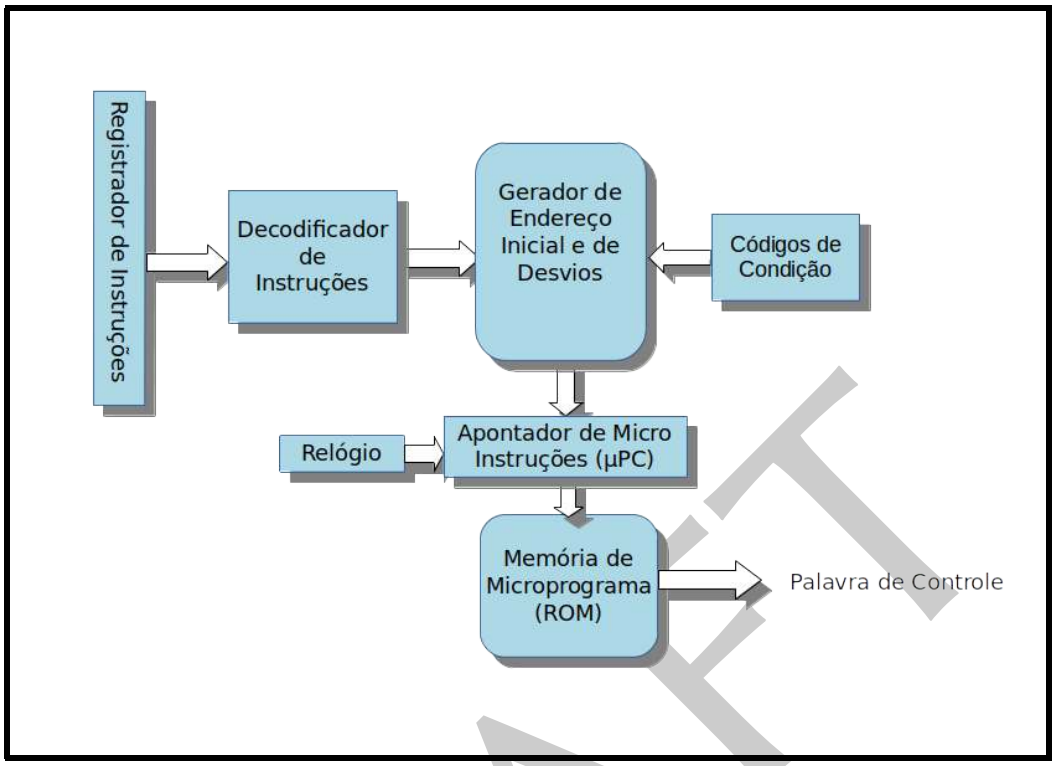


Figura 3.8 – Unidade de Controle por Hardware

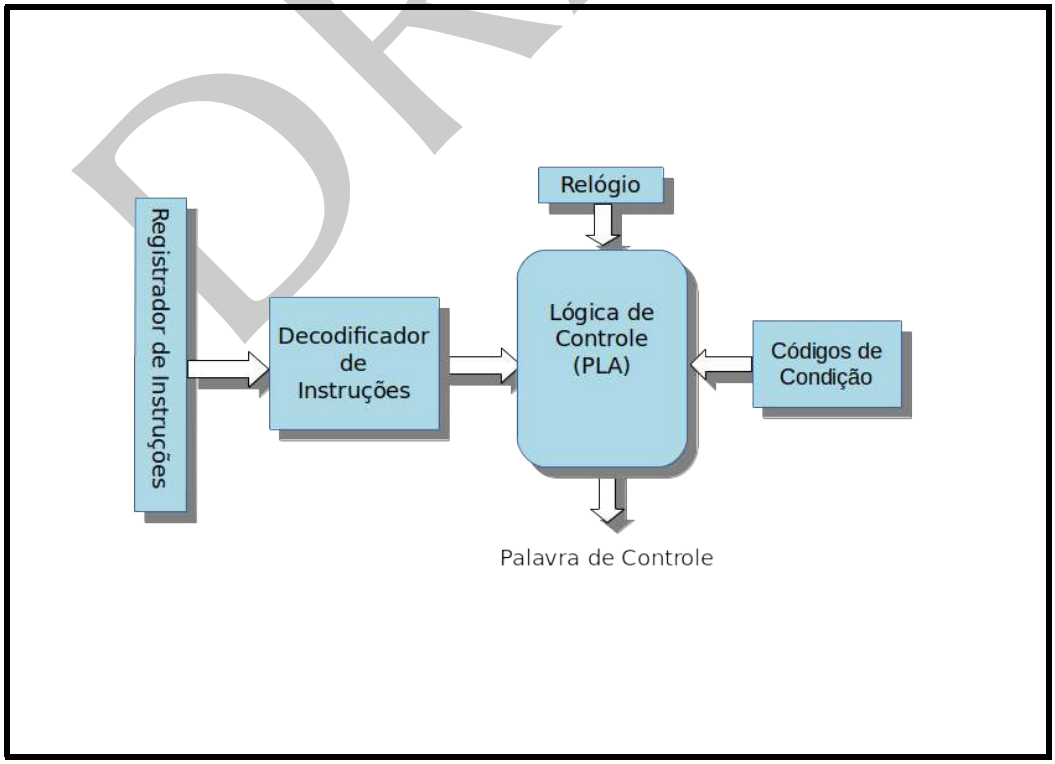
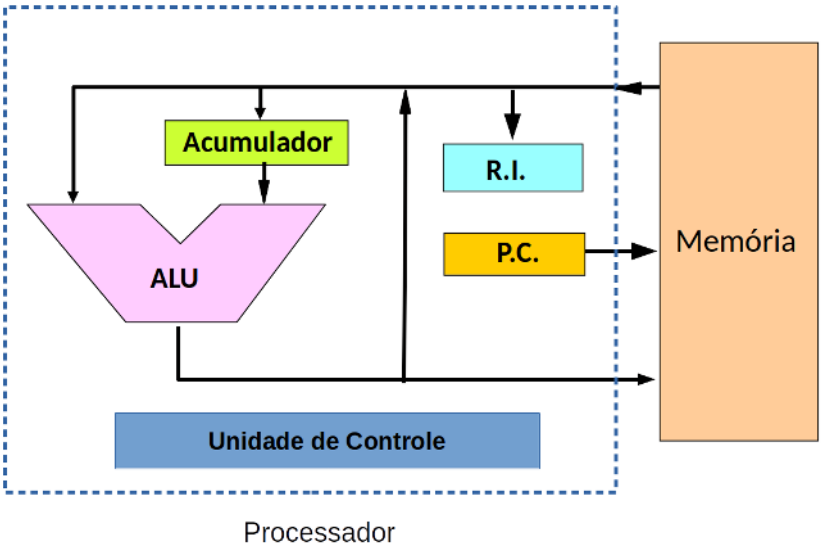


Figura 3.9 – Processador com Acumulador



- Busca a próxima instrução que está localizada na memória para o registrador de instrução;
- Atualiza o apontador de instruções (PC) para o endereço da próxima instrução a ser executada;
- Determina o tipo de instrução e o número de operandos;
- Busca os operandos, se houver, para os registradores do processador;
- Executa a instrução;
- Armazena os resultados;
- Volta ao passo 1 para executar a próxima instrução.

Os processadores mais simples possuem pelo menos um acumulador (AC), enquanto que os mais sofisticados possuem dezenas ou até mesmo centenas de registradores para armazenar esses resultados temporários das operações. Em qualquer caso, em algum momento, esses valores armazenados nos registradores serão transferidos para a memória do computador, nas posições correspondentes às variáveis do programa. Um modelo bem simples de processador pode ser visto na Figura 3.9.

Uma facilidade que os processadores modernos oferecem é para a chamada de rotinas ou procedimentos, ou seja, trechos de programa que podem ser acessados a partir de vários pontos do programa principal. A chamada de procedimentos, ao contrário de um desvio normal, requer que um endereço de retorno seja armazenado em algum local, para que o processador possa retornar para o ponto do programa principal de onde a rotina foi chamada. Assim, esse

endereço armazenado é utilizado para atualizar o valor do PC para retornar de uma rotina, voltando para o endereço da instrução imediatamente depois daquela que originalmente chamou a rotina.

Diversas formas podem ser utilizadas para armazenar este endereço de retorno, sendo que uma delas guarda este endereço em uma estrutura dados na memória, chamada de pilha, onde os elementos são retirados sempre na ordem inversa com que são colocados, ou seja, os últimos elementos colocados são os primeiros a serem retirados e vice-versa.

Existe então um registrador especial, chamado de um apontador de pilha (SP), que aponta sempre para o último elemento que está no topo da pilha e que é automaticamente atualizado quando os elementos são inseridos ou retirados da pilha. Uma característica especial da pilha é que ela é ”cresce” no sentido inverso dos endereços de memória, do final para o começo da memória. Quando um valor é inserido na pilha, o apontador de pilha é decrementado, antes da movimentação dos dados, de um valor igual ao tamanho do dado a ser colocado na pilha. Quando um valor é retirado da pilha, o apontador é incrementado, depois da movimentação dos dados, de um valor igual ao tamanho do dado retirado na pilha.

3.4 TIPOS DE ARQUITETURA DE PROCESSADORES

Os processadores podem ter diversos tipos de arquitetura que se diferenciam basicamente em relação à forma de acesso e de armazenamento interno dos operandos para a execução das instruções. Faremos nesta seção uma breve introdução aos seguintes tipos de arquitetura:

- **Pilha**
- **Acumulador**
- **Memória-Memória**
- **Registrador-Memória**
- **Registrador-Registrador**

Para cada uma dessas arquiteturas mostraremos na Tabela 3.1 como realizar a operação $\mathbf{C} := \mathbf{A} + \mathbf{B}$ em detalhes, utilizando pseudo instruções em linguagem de montagem.

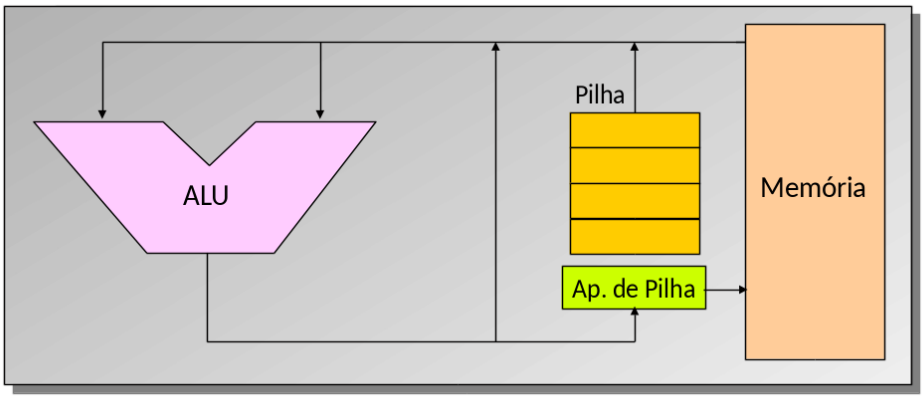
Pilha	Acumulador	Memória-Memória	Registrador-Memória	Registrador-Registrador
PUSH A	LOAD A	ADD C, B, A	LOAD R1, A	LOAD R1, A
PUSH B	ADD B		ADD R1, B	LOAD R2, B
ADD	STORE C		STORE C, R1	ADD R3, R1, R2
POP C				STORE C, R3

Tabela 3.1 – Tipos de Arquitetura

3.4.1 Arquitetura de Pilha

Nos processadores com arquitetura de pilha todos os operandos implicitamente estão no topo da pilha. Com isso, as instruções gastam menos bits, quando comparados com as arquiteturas com registrador, para codificar a origem e destino das operações. Isso era uma propriedade importante nos computadores mais antigos, pois resultava em um tamanho menor para a codificação das instruções, permitindo uma significativa economia de memória.

Figura 3.10 – Arquitetura de Pilha



As únicas operações de acesso à memória são ”POP” e ”PUSH”, para retirar e colocar um operando no topo da pilha. Normalmente os primeiros elementos no topo da pilha se encontram junto ao processador e o restante da pilha é distribuído na memória a partir do endereço dado pelo apontador de pilha. A Figura 3.10 representa um modelo desta arquitetura. A máquina virtual Java pode ser considerado um exemplo moderno deste tipo de arquitetura.

```
PUSH A
PUSH B
ADD
POP  C
```

No exemplo mostrado acima, podemos verificar que a operação $\mathbf{C := A + B}$ é feita colocando-se primeiro o valor da variável A no topo da pilha, com a a instrução **PUSH A**. Logo a seguir a variável B é colocado no topo da pilha com a instrução **PUSH B**. Agora, com os valores de A e B já colocados nas duas primeiras posições da pilha, a instrução **ADD** é executada, somando esses dois valores e o resultado é colocado por sua vez no topo da pilha. A seguir o resultado é retirado do topo da pilha e escrito na variável C em memória com a instrução **POP C**.

3.4.2 Arquitetura de Acumulador

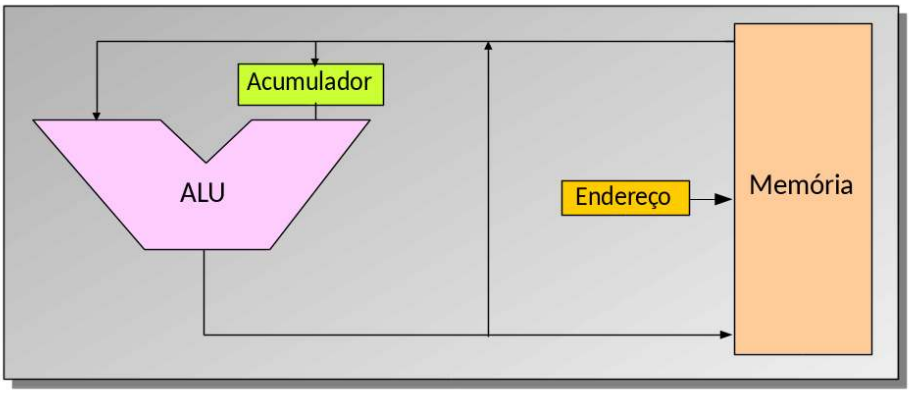
A arquitetura de acumulador possui o acumulador com um dos operandos implícitos na maioria instruções. O acumulador é um registrador especial colocado junto à unidade aritmética e lógica (UAL) com o intuito de agilizar as operações que são realizadas pelo processador. Se todas as operações fossem realizadas diretamente com todos os operandos em memória provavelmente haveria um grande impacto negativo no desempenho desses processadores, já que tempo gasto para acessar os dados na memória é bem maior que o tempo para acessar o dado armazenado no acumulador.

É um modelo de arquitetura muito utilizado em processadores mais simples, tantos nos processadores mais antigos como nos modernos processadores embarcados, por ser de fácil implementação e ser uma solução de compromisso entre custo e desempenho. Exemplos modernos de arquitetura de acumulador são os processadores PIC e o 8051. A Figura 3.11 representa um modelo desta arquitetura.

```
LOAD  A
ADD   B
STORE C
```

No exemplo mostrado acima, podemos verificar que a operação $\mathbf{C} := \mathbf{A} + \mathbf{B}$ é feita colocando-e primeiro o valor da variável A no acumulador, com a instrução **LOAD A**. Logo a seguir o acumulador é somado com a variável B com a instrução **ADD B**, sendo que o resultado é colocado de volta no acumulador. A seguir o resultado é retirado do acumulador e escrito na variável C em memória com a instrução **STORE C**.

Figura 3.11 – Arquitetura de Acumulador



3.4.3 Arquitetura Memória-Memória

Os processadores com arquitetura memória-memória possuem instruções aritméticas e lógicas com todos os operandos colocados de forma explícita, ou seja, possuem um total de 3 operandos, podendo estar todos em memória. Embora em sua maioria não dispensem o uso também de registradores, as instruções aritméticas e lógicas podem referenciar todas as suas variáveis diretamente na memória. Normalmente o código gerado para essas máquinas possui um número menor de instruções mas que, por sua vez, resultam em arquiteturas com implementação bastante complexa. Um exemplo de arquitetura deste tipo é o VAX, um processador bastante comercializado nas décadas de 70 e 80 no século passado.

```
ADD C, B, A
```

No exemplo mostrado acima, podemos verificar que a operação $\mathbf{C := A + B}$ é feita com apenas uma única instrução **ADD C, B, A** que lê as variáveis A e B da memória, realiza a soma e escreve o resultado de volta na memória na variável C.

3.4.4 Arquitetura Registrador-Memória

Os processadores com arquitetura registrador-memória normalmente possuem instruções aritméticas e lógicas com dois operandos, sendo que um deles está em memória e o outro em registrador. O resultado da operação é escrito automaticamente no mesmo registrador. Os exemplos clássicos deste tipo arquitetura são o IBM 360 e os processadores da linha Intel x86.

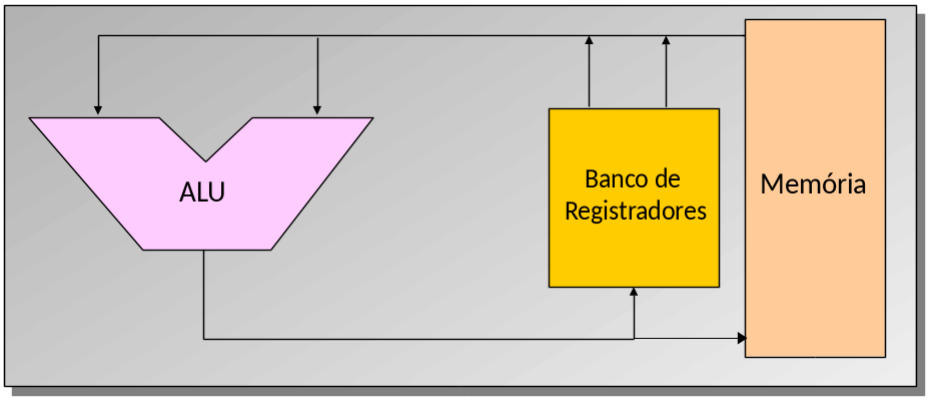
```
LOAD  R1, A
ADD   R1, B
STORE C, R1
```

No exemplo mostrado acima podemos verificar que a operação $\mathbf{C := A + B}$ é realizada colocando-se primeiro o valor da variável A no registrador R1, com a instrução **LOAD R1, A**. Logo a seguir o valor armazenado no registrador é somado com a variável B com a instrução **ADD R1, B**, sendo que o resultado é colocado de volta no registrador R1. A seguir o resultado é copiado do registrador R1 e escrito na variável C em memória com a instrução **STORE C, R1**.

3.4.5 Arquitetura Registrador-Registrador

Os processadores com arquitetura registrador-registrador possuem características particulares. A primeira delas é que as instruções aritméticas e lógicas possuem os três operandos em registrador, sendo dois de origem e um destino. As únicas instruções que fazem acesso à memória são ”LOAD”, para carregar os dados da memória para os registradores e ”STORE”, para fazer o caminho inverso.

Figura 3.12 – Arquitetura de Registrador



A compilação dos programas em alto nível resultam um código com um maior número de instruções, porém mais simples. Essas características permitem o desenvolvimento de projetos de processadores mais simples e, em consequência, mais rápidos e com menor consumo de energia. Por isso é o modelo de arquitetura, característico das arquiteturas RISC (veja a Seção 3.7), é o mais disseminado entre os processadores comerciais atualmente. A Figura 3.12 apresenta um modelo simplificado desta arquitetura.

```
LOAD  R1, A
LOAD  R2, B
ADD   R3, R1, R2
STORE C, R3
```

No exemplo mostrado acima, podemos verificar que a operação $\mathbf{C} := \mathbf{A} + \mathbf{B}$ é feita colocando-se primeiro o valor da variável A no registrador R1 com a instrução **LOAD R1, A**. Logo a seguir o registrador R2 recebe o valor da variável B com a instrução **LOAD R1, B**. Depois conteúdo de R1 é somado com R2 e o resultado é colocado no registrador R3. A seguir o valor armazenado em R3 é escrito na variável C em memória com a instrução **STORE C, R3**.

3.5 MODOS DE ENDEREÇAMENTO

Em uma arquitetura de processador as instruções podem ter diversos modos de acessar os seus operandos. Os operandos podem ser constantes ou variáveis em registrador ou memória. Alguns modos de endereçamento existentes podem ser vistos a seguir:

- **Registrador ou Acumulador**

> ADD R4, R3 ⇒ Regs[4] ← Regs[4]+Regs[3]
> Operando no registrador ou acumulador.

- **Imediato**

> ADD R4, #8 ⇒ Regs[4] ← Regs[4]+8
> Operando é uma constante na instrução.

- **Direto ou Absoluto**

> ADD R1, (1001) ⇒ Regs[1] ← Regs[1]+Mem[1001]
> Operando em memória com endereço na instrução.

- **Indireto (via Registrador)**

> ADD R4, (R1) ⇒ Regs[4] ← Regs[4]+Mem[Reg[1]]
> Operando em memória com endereço no registrador.

- **Indireto via Memória**

> ADD R2, @(1003) ⇒ Regs[2] ← Regs[2]+Mem[Mem[1003]]
> Operando em memória com endereço na memória.

- **Deslocamento**

> ADD R2, 100(R1) ⇒ Regs[2] ← Regs[2]+Mem[R1+100]
> Operando em memória com endereço obtido pela soma do conteúdo do registrador com uma constante.

- **Indexado**

> ADD R2, (R3+R4) ⇒ Regs[2] ← Regs[2]+Mem[R3+R4]
> Operando em memória com endereço obtido pela soma do conteúdo dos registradores.

- **Pilha**

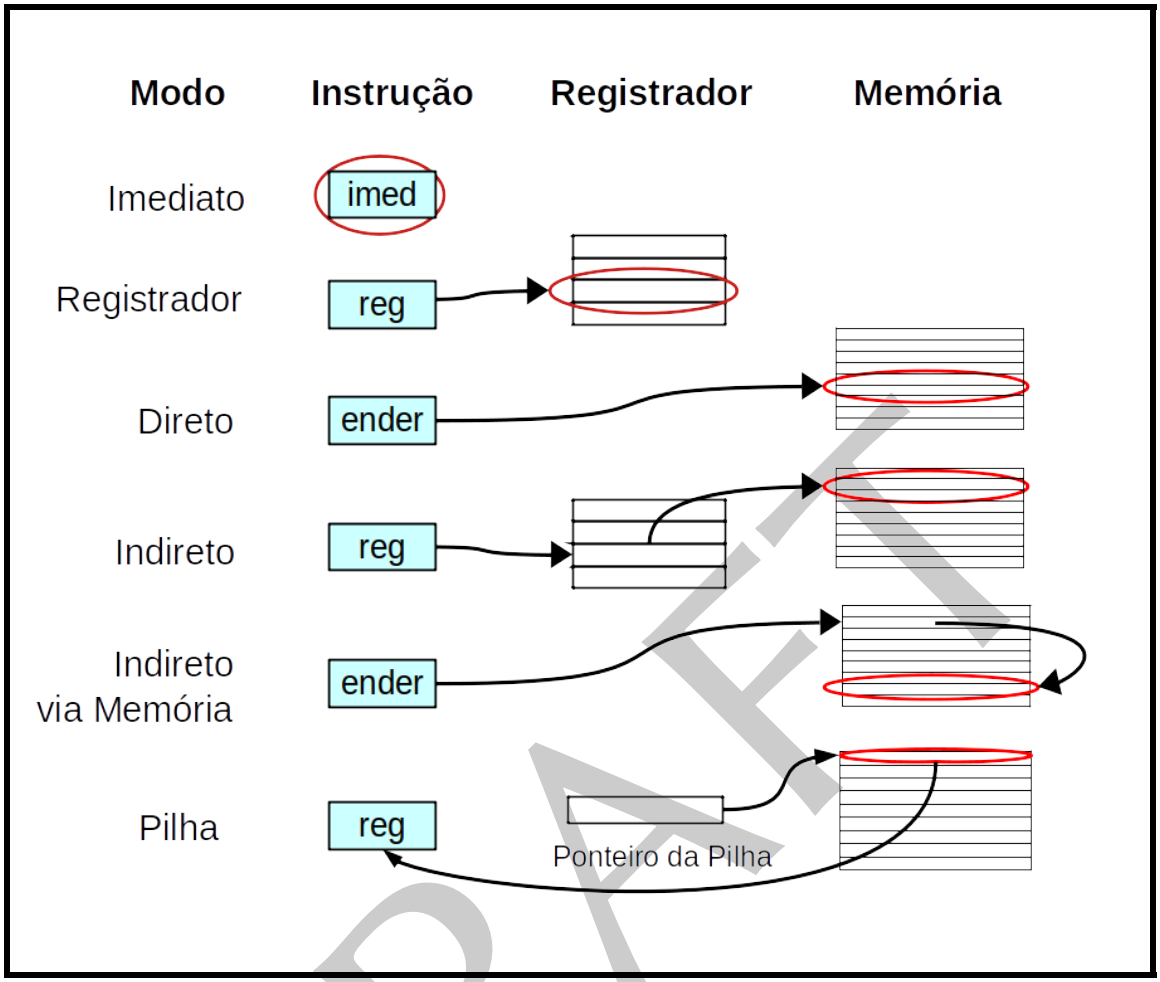
PUSH $\Rightarrow$ SP = SP +1 ; MEM[SP] $\leftarrow$ ACC A pilha é um operando implícito da instrução

Note que uma instrução pode fazer uso de modos diferentes para buscar cada um dos seus operandos. A Figura 3.13 ilustra alguns modos de endereçamento.

- No modo registrador ou acumulador, o operando da instrução está armazenado em um registrador ou no acumulador. Note que no caso do acumulador, ele será um operando implícito da instrução, e não aparecerá no código em linguagem de montagem.
- No modo imediato o valor do operando está codificado na própria instrução ou em *bytes* subsequentes a ela.
- No modo direto, o valor codificado na instrução (ou nos *bytes* subsequentes) não é operando em si, mas o endereço do operando na memória. Um novo acesso precisa ser realizado para acessar o dado propriamente dito.
- No modo indireto, alguma vezes referenciado como indireto via registrador, o registrador não possui o operando, mas o endereço do operando na memória. Aqui também é necessário um acesso à memória para buscar o operando.
- No modo indireto via memória, a instrução (ou os *bytes* subsequentes) contém o endereço de memória da posição de memória que tem o endereço do operando. São necessários dois acessos à memória para buscar o operando.
- No modo deslocamento, a instrução (ou os *bytes* subsequentes) contém uma constante que é somada ao conteúdo de um registrador, também especificado na instrução, para obter o endereço do operando na memória. É necessário mais um acesso à memoria para buscar o operando da instrução.
- No modo indexado, a instrução (ou os *bytes* subsequentes) indica quais registradores devem ter seus conteúdos somados para obter-se o endereço do operando na memória. É necessário mais um acesso à memoria para buscar o operando da instrução.
- No modo pilha, o apontador de pilha (SP) é utilizado para indexar a memória e buscar o operando, de uma forma implícita .

Existem ainda outros modos de endereçamento mais complexos utilizado por diversos processadores, mas cujo estudo foge ao escopo deste livro. Para maiores informações recomendamos a consulta ao livro de Hennessy e Patterson (HENNESSY J. L.; PATTERSON, 2003).

Figura 3.13 – Modos de Endereçamento



3.6 ORDENAÇÃO DOS BYTES NA MEMÓRIA

Uma variável com mais de um byte de comprimento pode ser armazenada de duas formas diferentes, de acordo com a ordenação que o processador dá para a sequência de bytes dessa variável na memória. Estes modos de ordenação são conhecidos com *big-endian* e *little-endian*.

Um sistema *big-endian* armazena o byte mais significativo de uma palavra no menor endereço de memória e o menos significativo no maior. Um sistema *little-endian*, por outro lado, armazena o byte menos significativo no menor endereço.

Os computadores armazenam as varáveis em vários grupos de tamanhos binários. Na maioria dos computadores modernos, o menor grupo de dados com um endereço tem oito bits e é chamado de byte. As variáveis com dois ou mais bytes, por exemplo, uma variável de 32 bits contém quatro bytes, possui duas maneiras distintas para um computador ordenar os seus bytes individuais na memória e, ao longo da história da computação, os dois métodos foram usados, levando a alguns problemas ocasionais.

Internamente, qualquer computador funciona igualmente bem, independentemente de qual ordenação ele usa, já que seu *hardware* usa consistentemente a mesma ordenação para

armazenar e carregar seus dados. Por esse motivo, programadores e usuários de computador normalmente ignoram a ordenação do computador com o qual estão trabalhando.

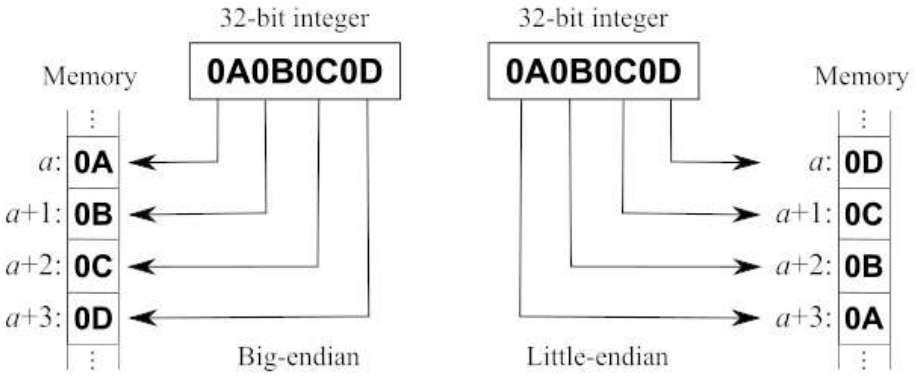
No entanto, essa ordenação pode se tornar um problema ao mover dados externamente ao computador - como ao transmitir dados entre computadores diferentes pela internet, ou um programador investigando bytes de dados internos do computador a partir de um *dump* de memória - e a ordenação usada difere do esperado. Nesses casos, a ordenação dos dados deve ser entendida e levada em consideração.

Ambos os tipos de ordenação são amplamente utilizados no desenvolvimento dos processadores. A escolha inicial da ordenação de um novo projeto geralmente é arbitrária, mas revisões e atualizações de tecnologia posteriores perpetuam essa ordenação existente e muitos outros atributos de projeto para manter a compatibilidade com versões anteriores.

A ordenação utilizada nos protocolos de rede é a *big-endian*, como no conjunto de protocolos da Internet, onde é chamada de ordem de rede, transmitindo primeiro o byte mais significativo primeiro. Por outro lado, o *little-endian* é a ordem dominante para arquiteturas de processador (x86, a maioria das implementações de ARM, implementações de base do RISC-V) e a memória associada.

Algumas arquiteturas, com os processadores MIPS e ARM, podem fazer uso de um ou outro modo de ordenação indistintamente, a partir de uma configuração em um registrador de controle, e por isso são chamadas de *bi-endian*.

Figura 3.14 – Ordenação dos bytes



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

Uma curiosidade a respeito da origem dos termos *little-endian* e *big-endian*, que se deve a um artigo ”On Holy Wars and a Plea For Peace” de Danny Cohen publicado em 1 de abril de 1980 (sic!) (Danny Cohen, 1980). Uma versão editada apareceu na IEEE Computer Magazine, em outubro de 1981.

Este artigo clássico, muito influente e divertido, introduziu os termos *little-endian* e *big-endian* e instou as pessoas a parar de travar guerras sagradas em torno de qual ordenação de bytes era melhor. Ainda é uma leitura obrigatória para qualquer pessoa que tenha algum interesse relativo à ordenação de bytes.

Os termos *little-endian* e *big-endian* foram emprestados da sátira de Swift, "As Viagens de Gulliver", em que dois países entraram em guerra pela questão sobre qual extremidade de um ovo cozido, a menor ou a maior, ele deveria ser consumido primeiro.

Anteriormente, a ordenação dos bytes na memória tinha sido alvo de discussões intensas, tanto na indústria e academia, mas que se acalmaram bastante nos últimos anos.

3.7 RISC VERSUS CISC

Um dos objetivos da arquitetura de um processador é permitir a execução dos programas o mais rapidamente possível. Há várias técnicas que são empregadas para isso, mas de uma forma simplificada o tempo de execução de um programa pode ser definido pela seguinte equação:

$$T_p = C_i \times T_c \times N_i \tag{3.1}$$

Onde,

- 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

O C_i também é conhecido com CPI (Ciclos por Instrução) e define o número médio de ciclos gastos por cada instrução. É um valor que varia de acordo com o tipo de programa, por exemplo, inteiro ou de ponto flutuante, além da arquitetura do processador e é obtido dividindo-se o número total de ciclos de relógio gastos para executar um programa pelo total de instruções executadas.

O seu inverso é IPC (Instruções por Ciclo) e tem sido usado mais modernamente na arquiteturas superescalares, para indicar o número de instruções, normalmente maior do que um, que podem ser executadas a cada ciclo de relógio do processador.

O tempo do ciclo (T_c) é o inverso da frequência do relógio, segundo a fórmula:

$$T_c = \frac{1}{F} \tag{3.2}$$

Por exemplo, se a frequência do relógio for 100 MHz (100 x 10^6), o tempo de ciclo de relógio será de 10 ns (10 x 10^{-9}).

Logo, quando maior a frequência do relógio, menor será o tempo de ciclo de relógio, normalmente expresso em nanosegundos, e menor será o tempo total de execução do programa. Logo, se buscava reduzir o tempo de execução aumentando ao máximo a frequência do relógio. Mas isso tem um limite em função do atraso dos componentes do processador, assim como da potência que pode ser dissipada pelo conjunto encapsulamento e dissipador.

Uma outra forma de melhorar o desempenho é diminuir o número total de instruções (N_i) 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. Essas instruções podiam realizar de uma só vez várias operações complexas, acessando o maior número possível de operandos de uma única vez, com vários e diversos modos de endereçamento possíveis.

Mas a contrapartida é que o número de ciclos para executar cada instrução era muito maior, e um *hardware* mais complexo resultava em frequências de relógio menores, dado o longo tempo necessário para a decodificação das instruções. Essa aproximação foi utilizada pelas arquiteturas CISC (complex instruction set computers), utilizada nas primeiras arquiteturas de processadores, incluindo aquelas utilizadas nos primeiros microprocessadores, sendo absolutos até o início da década de 80.

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, procurava-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.

Para atingir esse objetivo as instruções foram simplificadas ao máximo, realizando cada uma delas o menor número operações e com a menor quantidade de operandos possível. Além disso, os modos de endereçamento dos operandos utilizados eram extremamente simples, com o objetivo de diminuir a complexidade do *hardware* do processador. As novas arquiteturas receberam o nome de RISC (Reduced Instruction Set Computers). A consequência negativa disso é que um maior número de instruções em linguagem de máquina eram necessárias para codificar um programa em linguagem de alto nível quando comparados com as arquiteturas convencionais da época, mas fato esse plenamente compensado pelos demais ganhos que a arquitetura apresentava.

Durante a década que se seguiu um intenso debate foi realizado sobre qual tipo de arquitetura seria melhor e mais rápida. Porém, no início da década de 90, os processadores RISC se estabeleceram como o padrão de arquitetura no mercado.

Exemplos de arquitetura CISC eram então os processadores x86 da Intel. Já os processadores SPARC, MIPS e ARM são exemplos de arquiteturas RISC. Atualmente os processadores da Intel são uma arquitetura híbrida, sendo externamente compatíveis com o código da arquitetura x86, mas que são traduzidas em tempo de execução para as instruções RISC da arquitetura interna que efetivamente executam as instruções.

Resumindo, as arquiteturas CISC apresentavam as seguintes características:

- 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 podiam referenciar operandos na memória principal;
- Cada fase do processamento da instrução podia ter duração variável em função da complexidade.

Que resultavam nas seguintes consequências:

- Implementação com uso de pipeline (vamos ver mais adiante) é difícil;
- A taxa média de execução das instruções por ciclo tende a ser bastante superior a 1 ciclo por instrução (CPI);
- A unidade de controle é em geral microprogramada;
- Códigos compactos podem ser gerados pelos compiladores;

Por outro lado as arquiteturas RISC ofereciam as seguintes características resumidamente:

- 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.

Resultando também nas seguintes consequências:

- São implementadas com o uso do pipeline;
- A taxa média de execução de instruções por ciclo de máquina é igual ou inferior a 1 ciclo por instrução (CPI);
- A unidade de controle é em geral *hardwired*;
- O processo de compilação é complexo e requer cuidados especiais para otimização do desempenho do código gerado.