

# **Infraestrutura de Hardware**

**Juliana Regueira Basto Diniz  
Abner Corrêa Barros**

**Volume 2**

**Recife, 2009**

## **Universidade Federal Rural de Pernambuco**



Reitor: Prof. Valmar Corrêa de Andrade

Vice-Reitor: Prof. Reginaldo Barros

Pró-Reitor de Administração: Prof. Francisco Fernando Ramos Carvalho

Pró-Reitor de Extensão: Prof. Paulo Donizeti Siepierski

Pró-Reitor de Pesquisa e Pós-Graduação: Prof. Fernando José Freire

Pró-Reitor de Planejamento: Prof. Rinaldo Luiz Caraciolo Ferreira

Pró-Reitora de Ensino de Graduação: Profª. Maria José de Sena

Coordenação Geral de Ensino a Distância: Profª Marizete Silva Santos

### **Produção Gráfica e Editorial**

Capa e Editoração: Allyson Vila Nova, Rafael Lira, Italo Amorim, Gláucia Micaele

Revisão Ortográfica: Marcelo Melo

Ilustrações: Abner Barros, Allyson Vila Nova, Diego Almeida e Moisés de Souza

Coordenação de Produção: Marizete Silva Santos

## **Sumário**

<b>Apresentação .....</b>	<b>4</b>
<b>Conhecendo o Volume 2 .....</b>	<b>5</b>
<b>Capítulo 1 – Evolução das Arquiteturas .....</b>	<b>7</b>
<b>Capítulo 2 – Unidade de Controle e Caminho de Dados .....</b>	<b>15</b>
<b>Capítulo 3 – Ciclo de Execução e Interrupções .....</b>	<b>69</b>
<b>Capítulo 4 – Estruturas de Interconexão .....</b>	<b>82</b>
<b>Considerações Finais .....</b>	<b>102</b>
<b>Conheça os Autores .....</b>	<b>104</b>

# Apresentação

Caro(a) Cursista,

Estamos, neste momento, iniciando o segundo volume do livro da disciplina de Infraestrutura de Hardware. Neste volume, iremos discutir um pouco sobre a evolução das arquiteturas dos processadores, bem como aspectos referentes ao processamento de dados e de controle. Também discutiremos sobre o ciclo de execução com e sem o uso de interrupções.

Por fim, apresentaremos os barramentos ou as estruturas de interconexão que servem para interligar todos os dispositivos periféricos às partes básicas do computador, tais como memória e CPU. Estudaremos os princípios básicos de funcionamento, os métodos de arbitragem, as hierarquias e alguns principais padrões comerciais de barramentos.

Bons estudos!

Juliana Regueira Basto Diniz

Abner Barros

*Professores Autores*

# Conhecendo o Volume 2

Neste segundo volume, você irá encontrar os conteúdos referentes ao módulo 02 da disciplina **Infraestrutura de Hardware**. Para facilitar seus estudos, veja os conteúdos encontrados neste volume.

## **Módulo 2 – Subsistema de Processamento Central**

**Carga horária:** 15h

### **Objetivo Principal do Módulo 2:**

- » Apresentar a evolução das arquiteturas dos processadores, bem como aspectos referentes ao processamento de dados e de controle. Em seguida, iremos discutir sobre o ciclo de execução com e sem o uso de interrupções e apresentar os barramentos ou as estruturas de interconexão que servem para interligar todos os dispositivos periféricos às partes básicas do computador, tais como memória e CPU.

### **Conteúdo Programático do Módulo 02:**

- » Evolução das Arquiteturas.
- » Processador: Controle e Dados.
- » Interrupções
- » Estruturas de Interconexão: Barramentos



## Capítulo 1

### O que vamos estudar?

Neste capítulo, vamos estudar o seguinte tema:

- » Evolução das arquiteturas dos processadores

### Metas

Após o estudo deste capítulo, esperamos que você consiga:

- » Compreender as diferenças entre arquiteturas RISC e CISC;
- » Compreender a evolução das arquiteturas sob o ponto de vista tecnológico;
- » Entender conceitos fundamentais associados às arquiteturas dos processadores.

# Capítulo 1 – Evolução das Arquiteturas



## Vamos conversar sobre o assunto?

Você sabe o que é multiprogramação? Mesmo que você nunca tenha ouvido falar em multiprogramação, você certamente perceberia se o seu sistema operacional não empregasse essa técnica, pois você nunca poderia abrir várias janelas de aplicativos numa mesma sessão. Multiprogramação é uma tentativa de manter o processador sempre ocupado com a execução de um ou mais programas por vez. Com a multiprogramação, diversos programas são carregados para a memória e o processador comuta rapidamente de um para o outro. Essa técnica permite que o processador não passe por períodos de ociosidade, à espera de respostas oriundas de programas em execução, permitindo otimizações e melhorias de desempenho de processamento. O multiprocessamento, por sua vez, caracteriza-se pelo uso de vários processadores que atuam em paralelo, cada um executando as suas tarefas

Para garantir melhor desempenho, ainda é possível que os processadores atuais utilizem multiprocessamento com multiprogramação, que é a junção dos conceitos, onde mais de um processador é utilizado e cada um deles implementa a multiprogramação, permitindo efetivamente que programas sejam executados paralelamente. Os computadores pessoais evoluíram com processadores que implementam a multiprogramação e comportam o multiprocessamento.

Neste capítulo, vamos conversar sobre a evolução das arquiteturas de processadores e para tal, falaremos de alguns conceitos que favorecem melhoria de desempenho aos processadores. A multiprogramação, como citamos, é um exemplo disso.

## 1.1 RISC versus CISC

As primeiras máquinas eram extremamente simples com poucas instruções, até que surgiu a microprogramação, favorecendo o uso de

um conjunto complexo de instruções, conhecidas como instruções de máquina.

Daí surgiram as primeiras máquinas que seguiam a arquitetura CISC (*Complex Instruction Set Computer*), que traduzindo para o português seria Computador com Conjunto Complexo de Instruções.

Mas você sabe o que significa microprogramação? É um recurso que permite que um conjunto de códigos de instruções sejam gravados no processador, possibilitando-o a receber as instruções dos programas e executá-las utilizando as instruções gravadas na sua microprogramação.

No início da década de 80, havia uma tendência em construir processadores com conjuntos de instruções cada vez mais complexos, entretanto nem todos os fabricantes seguiram essa tendência, tomando o caminho inverso no sentido de produção de processadores de alta performance. A filosofia RISC (*Reduced Instruction Set Computer*) vai de encontro à maneira de pensar sobre arquiteturas de computadores. Segundo seus defensores, os processadores CISC estavam ficando complicados. Assim, surgiu a arquitetura RISC, que traduzindo para o português, significa Computador com Conjunto Reduzido de Instruções.

A arquitetura RISC surgiu com o objetivo de otimizar a performance dos computadores e foi construída com base em alguns princípios fundamentais que permitiram otimizações de desempenho. São elas: execução de uma instrução por ciclo, conjunto reduzido de instruções, grande número de registradores, instruções com formato fixo, poucos modos de endereçamentos e implementação similar a uma linha de montagem (*pipeline*).

Mas o que vem a ser um *Pipeline*? Um pipeline é composto por uma série de estágios (operando possivelmente em paralelo) onde uma parte do trabalho é feito em cada estágio, com funcionamento similar a uma linha de montagem. Sendo assim, o trabalho não está concluído até que tenha passado por todos os estágios.

A Figura 1 apresenta uma operação simples e rotineira como lavar roupas para exemplificar o funcionamento de um pipeline. Existem 4 estágios na lavagem de roupas representados por 4 diferentes ícones na figura.

O primeiro estágio refere-se à lavagem propriamente dita. O



segundo estágio é a centrifugação, enquanto que o terceiro é a secagem. O último estágio refere-se às ações de engomar e guardar a roupa.

Na parte superior da figura, observa-se 4 tarefas (A, B, C e D). Cada tarefa envolve a operação de lavar roupas e acontecem de forma sequencial, sem a execução no *pipeline*. A parte inferior da figura se observa a execução das mesmas tarefas, se utilizando um *pipeline*.

Na escala temporal (eixo horizontal superior), observa-se que as mesmas tarefas quando executadas em um *pipeline*, terminam muito antes do que elas terminariam caso não fosse utilizado o *pipeline*. Isso acontece porque enquanto uma tarefa está na fase da lavagem, outra está na fase da centrifugação, outra na fase da secagem e assim sucessivamente. A cada instante, cada estágio do *pipeline* executa uma determinada tarefa dando a ideia de paralelismo das ações e garantindo que o tempo de execução seja reduzido, permitindo ganhos em termos de desempenho.

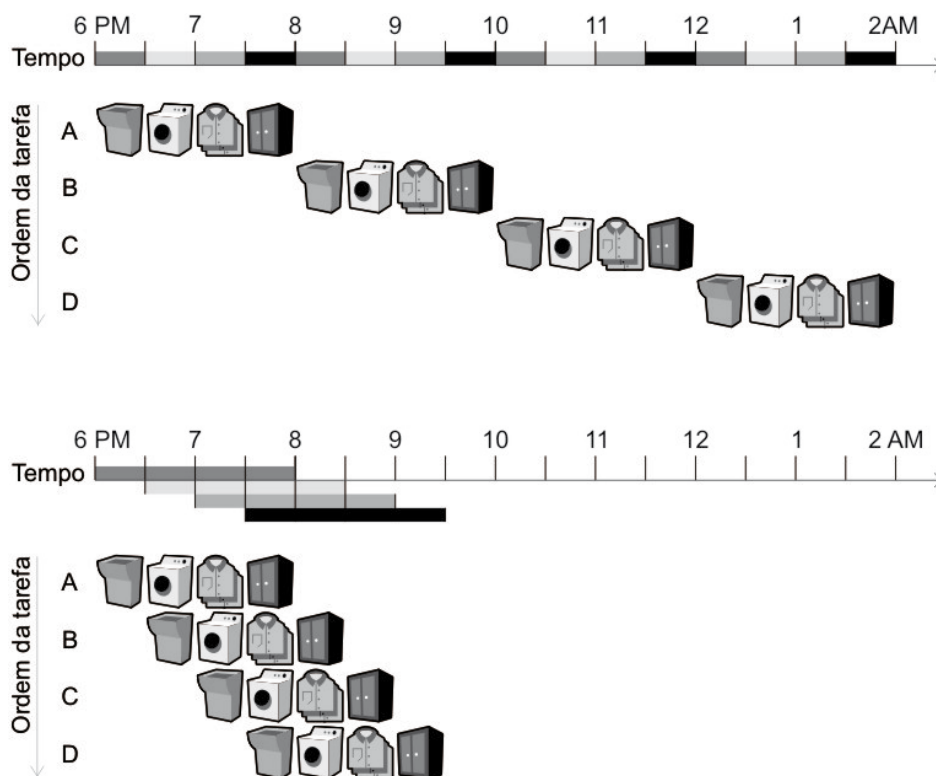


Figura 1 – Exemplo de um *Pipeline* para Lavar Roupas

Agora que você já tem ideia do que é um *pipeline* sobre o contexto simplificado de uma atividade rotineira, como lavar roupas, vamos trazê-lo para o contexto computacional. Suponha que o *pipeline* possua 5 estágios: busca instrução, execução, decodifica instrução,

acesso a dados externos e escrita de resultados. O fluxo de dados nesse *pipeline* está representado na Figura 2.

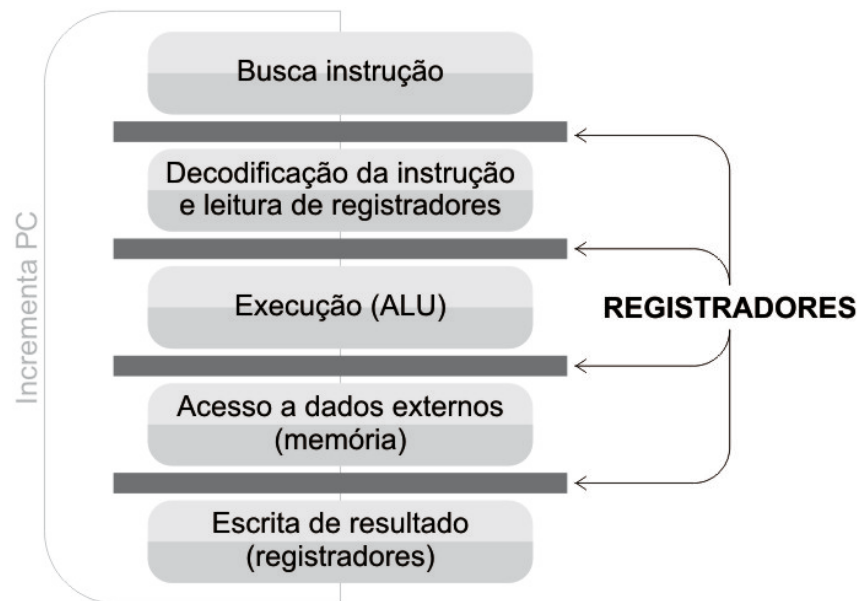


Figura 2 – Estágios do *Pipeline*

Um programa contendo 10 instruções, por exemplo, será executado por esta CPU e entrará nesse *pipeline*. Vamos numerar as instruções do programa de 1 a 10 e considerar que apenas as instruções de número 3 e 7 fazem referência à memória. As demais instruções passam apenas pelos 2 primeiros estágios do *pipeline*. Vamos considerar que a primeira linha da tabela 1 refere-se ao número de ciclos. Um ciclo é um período de tempo que uma instrução leva em um determinado estágio do *pipeline*.

Observando a Tabela 1, podemos dizer que no ciclo 1, a Instrução 1 é buscada. No ciclo 2, a Instrução 2 é buscada e a Instrução 1 é executada. No ciclo 3, a Instrução 3 (representada pela letra L) é buscada, a Instrução 2 é executada e a Instrução 1 finalizou. No ciclo 4, a Instrução 3 (L) é executada, mas não pode ser completada em um ciclo, pois faz referência à memória. No ciclo 5, a Instrução 4 é executada, embora o LOAD iniciado no ciclo anterior não tenha terminado. Isso pode acontecer, desde que a instrução 4 não tente utilizar o registrador no processo de ser carregado.

É de responsabilidade do compilador garantir que a instrução depois de um LOAD não utilize o item que está sendo buscado na memória. Se o compilador não puder encontrar nenhuma instrução para colocar depois do LOAD, ele sempre pode inserir uma instrução

NO-OP e desperdiçar um ciclo. Vale a pena ressaltar que LOAD/STORE são as únicas instruções que podem ser atrasadas.

Tabela 1 – Execução de um Programa com 10 instruções

CICLO	1	2	3	4	5	6	7	8	9	10
Busca de Instrução	1	2	L	4	5	6	S	8	9	10
Execução de Instrução		1	2	L	4	5	6	S	8	9
Referência à Memória					L				S	

Os processadores RISC, ao contrário dos CISC (possuem muitos modos de endereçamento), apresentam apenas o endereçamento por registrador. A consequência imediata é que as instruções não fazem acessos à memória, sendo, portanto, executadas em um único ciclo. Assim, apenas duas instruções, LOAD e STORE, acessam a memória. Tais instruções, em geral, não podem ser completadas em um ciclo conforme observamos com a inserção de recursos como *pipeline*.

Outro aspecto que diferencia os processadores CISC dos RISC é o número de registradores. Os registradores são memórias internas à CPU e por esse motivo possuem acesso mais rápido. Aumentando-se o número de registradores disponíveis no processador, aumenta-se o desempenho, já que mais dados estarão disponibilizados em memórias internas à CPU.

A arquitetura RISC foi amplamente difundida em estações de trabalho SPARC lançadas pela SUN Microsystems no final da década de 80. A arquitetura CISC foi empregada nos primeiros chips x86. Com o passar do tempo e com a evolução dos processadores a distinção entre CISC e RISC foi se tornando menos notável. Os processadores da Intel a partir do Pentium Pro passaram a executar um conjunto de instruções simples, tal como a filosofia RISC e novas instruções foram surgindo tornando esse conjunto de instrução não tão reduzido, aproximando alguns processadores RISC da filosofia CISC.

Essa tendência de aumento na complexidade também levou os processadores Power PC, que foram concebidos na arquitetura RISC, a agregar um número maior de instruções, aproximando-se, portanto, da arquitetura CISC.



### Atividades e Orientações de Estudo

Dedique, pelo menos, 1 hora de estudo para este primeiro capítulo. Você deve organizar uma metodologia de estudo que envolva a leitura dos conceitos que serão ditos apresentados neste volume e pesquisas sobre o tema, usando a Internet e livros de referência.

Utilize os fóruns temáticos e os chats desta disciplina para troca de informações sobre o conteúdo no ambiente virtual, pois a interação com colegas, tutores e o professor da disciplina irá ajudá-lo a refletir sobre aspectos fundamentais tratados aqui.

Também é importante que você leia atentamente o guia de estudo da disciplina, pois nele você encontrará a divisão de conteúdo semanal, ajudando-o a dividir e administrar o seu tempo de estudo semanal. Procure responder as atividades propostas como atividades somativas para este capítulo, dentro dos prazos estabelecidos pelo seu professor, pois você não será avaliado apenas pelas atividades presenciais, mas também pelas virtuais.



### Vamos Revisar?

Observamos neste capítulo a evolução das arquiteturas de computadores. Você pôde notar que inicialmente os processadores eram bastante simples com poucas instruções e modos de endereçamento. O acesso à memória principal era bastante demorado, o que levava os programas a gastarem mais tempo para a sua execução. A arquitetura RISC minimizou este problema diminuindo o número de acessos à memória principal através do uso de apenas duas instruções que fazem acesso à memória (LOAD e STORE) e do aumento na quantidade de registradores. Na arquitetura RISC, o programa é compilado diretamente pelo hardware, o que também possibilita uma melhoria de desempenho.

Com o passar dos anos, recursos que outrora foram introduzidos pelo padrão RISC passaram a ser introduzidos nas máquinas CISC e vice-versa, de modo que ambas as tecnologias passaram a se aproximar adotando estratégias de sucesso da sua antiga rival.

Aprendemos os fatores que diferenciam as duas arquiteturas e falamos de recursos empregados pelas arquiteturas modernas como multiprogramação e *pipeline*.

No próximo capítulo deste volume, você estudará a unidade de controle e o caminho de dados, ambos internos à CPU. No capítulo 3, estudaremos o ciclo de execução e as interrupções e no capítulo 4 estudaremos a interconexão da CPU com os demais subsistemas do computador através da análise dos sistemas de barramentos.



## Capítulo 2

### O que vamos estudar?

Neste capítulo, vamos estudar os seguintes temas:

- » A arquitetura interna de um processador
- » Principais componentes e funcionalidades do processador
- » Caminho de Dados
- » Estrutura da linguagem de máquina
- » Hardware interno do processador MIPS.

### Metas

Após o estudo deste capítulo, esperamos que você consiga:

- » Compreender como um processador funciona, e como ele é capaz de decodificar e executar as instruções de um programa.
- » Entender também como se dá a conversão de um programa de uma linguagem de alto-nível para a linguagem de montagem e desta para a linguagem de máquina do processador.
- » Descrever como o processador MIPS executa algumas de suas principais instruções utilizando os componentes de hardware presentes na sua arquitetura interna.

## Capítulo 2 – Unidade de Controle e Caminho de Dados



### Vamos conversar sobre o assunto?

No início da década de 70 do século passado, um filme de ficção científica fez história ao prever que em 2001 existiriam computadores capazes de pensar, agir e interagir com o ser humano por vontade própria. O filme a que me refiro no Brasil teve o título de **2001: Uma Odisseia no Espaço**, e o computador em questão tinha o nome de HAL 9000.

Pois bem, passados aproximadamente 40 anos do lançamento do referido filme, e mais de 70 anos após a construção do primeiro computador digital, mesmo com todo os avanços que estamos vivenciando, com os computadores cada vez mais presentes no dia a dia de cada um de nós, os computadores ainda não são capazes de agir e pensar por si mesmo e nem mesmo de se comunicar diretamente com qualquer um de nós. Ainda dependemos das linguagens de programação para instruí-los sobre qualquer coisa que queremos que eles façam. E mais que isto, em essência, em todos estes anos muito pouca coisa mudou na forma como os computadores “pensam”, em como as informações são representadas e em como os computadores processam estas informações. Mesmo assim, para a grande maioria das pessoas, e por que não dizer para muitos de nós, o funcionamento de um computador ainda permanece como um dos grandes mistérios da vida moderna.

Estamos agora começando mais um capítulo desta nossa viagem pelo interior dos computadores. Neste Capítulo vamos fazer um passeio pelo que poderíamos chamar de cérebro dos computadores, pelo interior do seu processador. Vamos buscar entender como um processador decodifica e executa as instruções contidas em um programa e como funciona a famosa “linguagem de máquina”, a língua dos computadores. Você verá que não existe nenhum mistério por trás de tudo isto, que tudo não passa de uma grande obra de engenharia de algumas mentes brilhantes do nosso século.

Você já está preparado?



### Dica de Filme

Saiba mais sobre este filme em [http://pt.wikipedia.org/wiki/2001\\_-\\_Uma\\_odiss%C3%A9ia\\_no\\_Espa%C3%A7o](http://pt.wikipedia.org/wiki/2001_-_Uma_odiss%C3%A9ia_no_Espa%C3%A7o)

Então, se acomode bem na cadeira, e nos acompanhe em mais esta viagem.

## 2.1 O processador, este ilustre desconhecido

Quando pensamos em tudo quanto um computador é capaz de fazer, em sua capacidade de processar imagens, sons, textos, jogos, planilhas eletrônicas, navegar na internet e coisas assim, ou ainda na sua capacidade quase inesgotável de efetuar cálculos de engenharia e modelar sistemas complexos através da conhecida realidade virtual, as vezes perdemos de vista que tudo isto não passa de programas sendo executados pelo processador deste computador.

Por este motivo, o processador é muitas vezes considerado o cérebro do computador, pois é ele quem decodifica e executa as instruções presentes nos programas.

De nada adianta termos um computador com uma grande quantidade de memória, ou com muito espaço de disco, ou ainda com uma placa mãe de última geração se não tivermos um processador de qualidade para processar e controlar tudo isto.

Mas como podemos identificar um processador de boa qualidade? O que diferencia um processador do outro?

Para que possamos responder esta pergunta, precisamos primeiramente relembrar os principais componentes da arquitetura interna de um processador. A Figura 1, a seguir, nos traz um modelo da estrutura interna dos processadores em geral.



Figura 1 – Estrutura interna de um processador



Vejamos para que serve cada uma das partes indicadas na figura.

- » **Unidade de Controle:** É responsável por buscar e decodificar cada instrução a ser executada. É esta unidade que determina quando e como as demais unidades funcionais do processador serão ativadas. É a unidade de controle que, a partir da decodificação da instrução a ser executada, programa a operação a ser efetuada pela Unidade Lógica e Aritmética e acessa os registradores onde estão armazenados os operandos a serem utilizados.
- » **Unidade Lógica e Aritmética:** A Unidade Lógica e Aritmética, como o próprio nome sugere, é a unidade responsável por executar todas as operações lógicas e aritméticas existentes no programa em execução.
- » **Banco de Registradores:** O Banco de Registradores serve como uma memória auxiliar de acesso rápido onde devem ficar armazenadas entre outras coisas as variáveis locais e os ponteiros do programa em execução.

Observe que não é a Unidade de Controle quem executa as operações, ela é responsável apenas por buscar e decodificar as instruções e, a partir daí, gerar os sinais de controle que ativarão as demais partes do processador. Ou seja, o processamento de dados propriamente dito é feito exclusivamente pela Unidade Lógica e Aritmética (ULA) a qual processa os dados normalmente armazenados nos registradores.

Esta associação entre o banco de registradores e a Unidade Lógica e Aritmética, por onde passam todos os dados para serem processados, é tão importante para o desempenho de um processador que recebe o nome de **Caminho de Dados**. E, o tempo necessário para que os dados sejam acessados nos registradores, aplicados à ULA, processados por esta e retornados para o banco de registradores recebe o nome de **Ciclo do Caminho de Dados**.

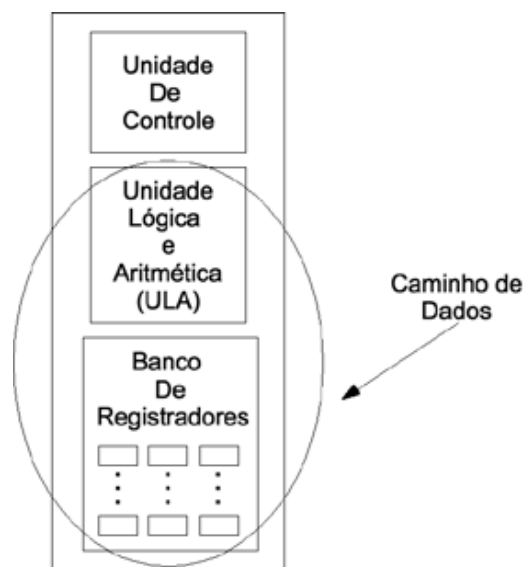


Figura 2 – Caminho de Dados (*Data Path*)

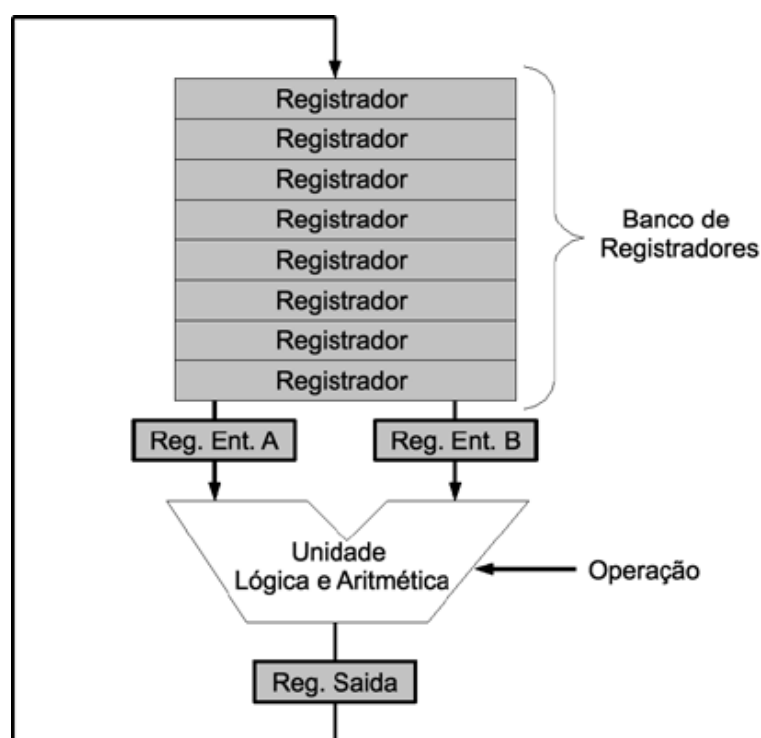


Figura 3 – Visão mais detalhada dos elementos que compõem o caminho de dados

Observe que o tempo do ciclo do caminho de dados é uma excelente medida para avaliar a eficiência do processador do ponto de vista da sua capacidade de processar informações. Mas como podemos avaliar a eficiência do processador quando inserido em uma determinada placa mãe ou em conjunto com um determinado chip de memória? Para este fim foi definido o **Ciclo de Buscar Decodificar e Executar**.

O Ciclo de Buscar Decodificar e Executar mede, portanto, o tempo total necessário para buscar uma instrução na memória principal, decodificá-la e executá-la através do ciclo do caminho de dados.

Existem diversas técnicas desenvolvidas especificamente com o objetivo de reduzir o tempo do ciclo de Buscar Decodificar e Executar, as quais estão presentes na maioria dos processadores atuais e são estas técnicas que em grande parte determinam a eficiência e o preço dos processadores. Quanto mais apurada a técnica empregada, quando mais eficiente o hardware utilizado, tanto mais caro e eficiente será o processador.

Um outro ponto que exerce grande impacto no uso e desempenho de um processador é o seu conjunto de instruções.

Em nosso contexto, o conjunto de instruções de um processador pode ser visto com um dicionário contendo todos os comandos existentes na sua linguagem de máquina os quais estão diretamente associados às operações que este processador consegue executar.

Mas por falar em linguagem de máquina, você sabe dizer o que vem a ser isto e qual a sua importância para o nosso estudo?

É isto o que vamos descobrir na próxima seção.

## **2.2 Comunicando com um computador: A Linguagem de Máquina**

Todo processo de comunicação, de troca de informações, seja entre dois seres humanos, entre um ser humano e uma máquina, ou ainda entre duas máquinas, é sempre baseado na troca de símbolos e regido por regras sintáticas e semânticas que determinam o significado que cada símbolo assume no contexto em que é utilizado.

Algum dia você já precisou se comunicar com alguém que não falava a sua língua? Como foi a sua experiência? Foi fácil ou difícil? De qualquer forma você deve ter percebido que para que haja comunicação é necessário que todas as partes envolvidas conheçam os símbolos, neste caso as palavras, e as regras sintáticas e semânticas que regem o seu significado, ou seja, que além de conhecer as palavras, saiba-se também como utilizá-las corretamente a fim de poder formar frases que expressem realmente o que se deseja comunicar.

No mundo virtual não é diferente. Para que possamos nos comunicar com um computador a fim de instruí-lo a respeito das tarefas a serem executadas, ou seja, para que possamos programá-lo, devemos conhecer algum tipo de linguagem de programação, seu vocabulário e suas regras sintáticas e semânticas.

No início da história dos computadores não existia o que nós conhecemos como linguagem de programação de alto nível, ou seja, aquele tipo de linguagem que nos permite descrever as tarefas a serem executadas através de expressões próprias da nossa linguagem coloquial. Desta forma, os programadores eram obrigados a conhecer e utilizar exclusivamente a linguagem de máquina. Isto equivale a dizer que os programas eram inteiramente escritos como sequências de zeros e uns. Sim, isto mesmo, você não leu errado, os programas eram inteiramente escritos como sequências de zeros e uns, isto porque, conforme já dissemos, os computadores são máquinas inteiramente baseadas em lógica digital, ou seja, toda e qualquer informação no interior de um computador deve ser expressa apenas como sequências de zeros e uns.




Com a evolução natural dos computadores e, por conseguinte dos programas de computador, logo percebeu-se que não era viável continuar a programar diretamente em linguagem de máquina. Assim surgiu a primeira linguagem de programação. Que na verdade não era bem uma linguagem de programação, nos moldes como conhecemos hoje, utilizando expressões da linguagem natural dos seres humanos. Era antes uma tradução direta das instruções existentes na linguagem de máquina para palavras de comando que podiam ser mais facilmente compreendidos e manipulados pelos seres humanos. Esta forma

de linguagem ficou conhecida como linguagem de montagem, ou simplesmente linguagem *assembly*, e os comandos desta linguagem ficaram conhecidos como mnemônicos, por serem abreviações e/ou siglas que representavam as operações que se queria executar. A Figura 4 a seguir nos permite ter uma pequena amostra da diferença entre programar em linguagem de montagem e diretamente em linguagem de máquina.

```

lw$t0, 0($2)
lw$t1, 4($2)
sw    $t1, 0($2)
sw    $t0, 4($2)

```



```

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

```

Figura 4 – Exemplo de trecho de programa em linguagem de montagem e o seu equivalente em linguagem de máquina

Observe que nada havia mudado no processador em si, este continuava compreendendo exclusivamente a linguagem de máquina. Apenas havia sido desenvolvida uma forma de tornar a atividade de programar os computadores um pouco mais natural e eficiente. Com esta nova abordagem, os programas eram primeiramente escritos em linguagem de montagem e em seguida passavam por um processo de tradução automática, normalmente efetuado por um outro programa de computador, denominado processo de montagem, daí o nome da linguagem ser linguagem de montagem. Neste processo todas as instruções mnemônicas utilizadas eram convertidas em comandos da linguagem de máquina, ou seja, em sequências de zeros e uns, para que se tornassem inteligíveis ao computador.

Com o passar do tempo, percebeu-se que seria possível construir programas melhores e em menor tempo se, em vez de se programar diretamente em linguagem de montagem, fosse possível programar em uma linguagem com comandos e tipos de dados mais expressivos, que estivessem mais perto do raciocínio humano, ou seja, com um maior grau de abstração. Surgia assim a primeira linguagem de programação de alto nível. Com ela, um programa podia ser escrito

utilizando expressões idiomáticas e estruturas textuais mais próximas as que nós utilizaríamos para descrever um algoritmo qualquer, o que tornava o processo de criação de um programa algo bem mais cômodo e natural. Esta facilidade entretanto tinha um custo, pois o processo de conversão de um programa escrito nesta linguagem para a linguagem de máquina tinha agora que ser feito em duas etapas. Primeiro o programa era traduzido da linguagem de alto nível para a linguagem de montagem, e deste para a linguagem de máquina. Este processo de tradução da linguagem de alto nível para a linguagem de montagem ficou conhecido como compilação e o programa que executa esta tarefa ficou conhecido como Compilador. A Figura 5 a seguir nos traz um exemplo do processo de conversão de um trecho de código escrito em linguagem de alto nível para linguagem de máquina, passando pela linguagem de montagem.

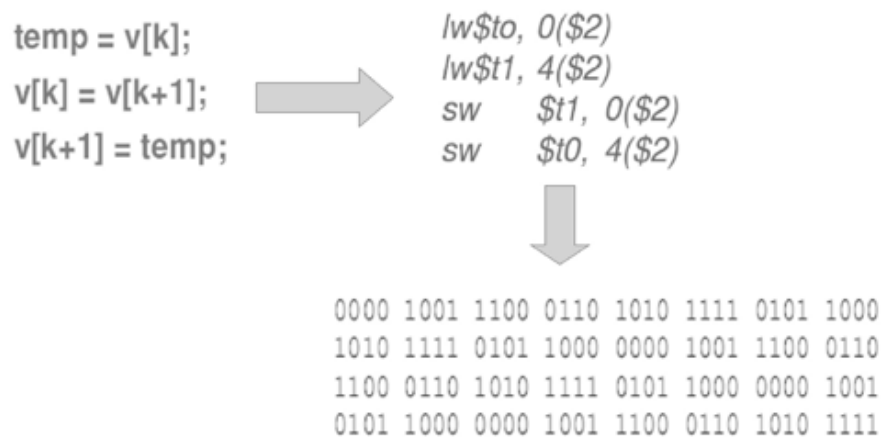


Figura 5 – Processo de conversão de um programa escrito em linguagem de alto nível para linguagem de máquina

Com o passar dos anos foram surgindo muitas linguagens e compiladores novos, entretanto, o processo conversão entre um programa escrito em alguma linguagem de alto nível e a linguagem de máquina continua praticamente o mesmo. A Figura 6, a seguir, nos traz o fluxo completo de Compilação e execução de um programa escrito em uma linguagem de alto nível para a linguagem de máquina.

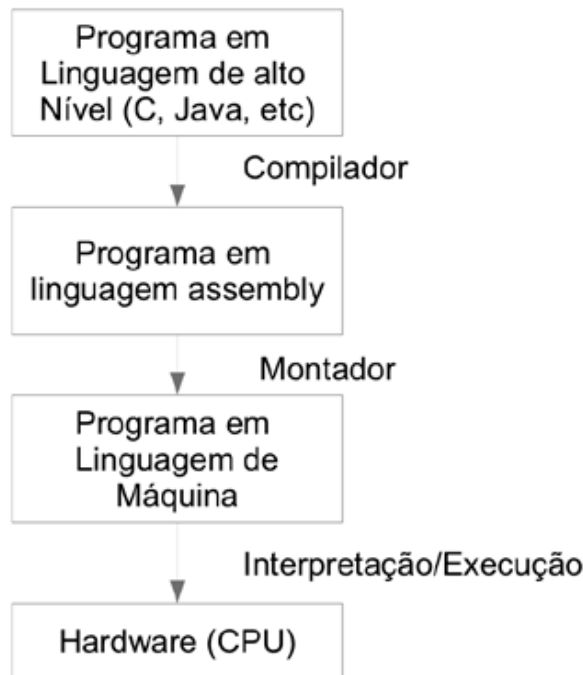


Figura 6 – Fluxo completo de Compilação de um programa qualquer

Isto tudo é muito interessante, não é mesmo?

Bem, agora que já temos uma visão de como se dá de fato a nossa comunicação com os computadores, ou seja, de como nossos programas passam da linguagem na qual foram escritos para a linguagem de máquina, que tal nos aprofundarmos um pouco mais e estudarmos como o processador decodifica e executa estas instruções? Que tal conhecer por dentro um dos processadores mais estudados e comercializados em todos os tempos?

Na próxima seção, vamos conhecer o processador MIPS, sua arquitetura interna, algumas instruções da sua linguagem de máquina e ver em detalhes como este processador decodifica e executa as instruções da sua linguagem de máquina.

## Arquitetura básica de um processador

Existem atualmente diversos tipos de processadores, indo desde os processadores mais simples, dedicados a executar pequenas funções em aparelhos eletro/eletrônicos, até aos processadores mais complexos, projetados para aplicações em que se precise de um alto poder de processamento, como em supercomputadores por exemplo. Todos estes processadores podem ser agrupados na forma de classes ou famílias conforme podemos ver na Tabela 1 a seguir:

Tabela 1 - Tipos de Processadores

Família	Aplicação
Processadores de propósito geral	Computadores pessoais (Desktop, Notebooks, etc)
Microcontroladores	Sistemas Embarcados (TVs, DVDs, GPS, Celular, Micro-ondas, Relógios, Console de Jogos, etc)
Processadores Vetoriais	Processamento de Alto desempenho, super computadores
Processadores Super-Escalar	Computadores de médio e grande porte
Processadores Gráficos	Placas gráficas, processamento de realidade virtual
Processadores Digitais de Sinal (DSP)	Equipamentos médicos, processamento de sinal de satélite, telecomunicações, etc.

Entretanto, de um modo geral, podemos dizer que todos os processadores compartilham de uma mesma arquitetura básica, ainda que diferindo no modo como esta arquitetura é implementada.

Ou seja, ainda que cada uma destas famílias de processadores possua características bem específicas, o que as torna especialmente aplicáveis a um ou outro tipo de aplicação, ainda assim podemos traçar um paralelo entre as suas unidades internas, de tal forma que possamos tratá-las apenas como processadores e nada mais.

Como vimos na Figura 1, todos os processadores possuem internamente no mínimo uma unidade de controle, um banco de registradores e uma unidade lógica e aritmética. Este seria um modelo mínimo razoável se o processador não precisasse acessar a memória principal e os dispositivos de entrada e saída durante o processamento normal das informações. A Figura 7, a seguir, nos traz o que poderíamos considerar um modelo mais completo de um processador.



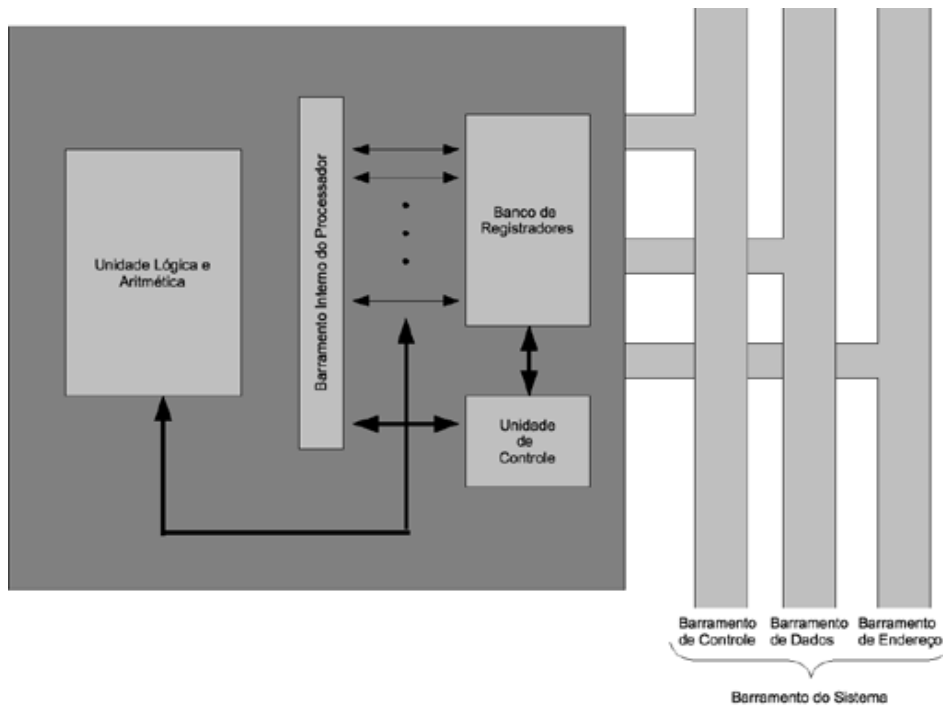


Figura 7 - Arquitetura interna de um processador incluindo o barramento do sistema

Existe ainda uma outra visão da arquitetura interna do processador que é muito útil quando se deseja ter uma visão mais funcional do mesmo. A Figura 8 nos traz esta visão. Nela em vez de destacar-se as unidades funcionais, destacam-se as funções associadas a alguns registradores de controle do processador.

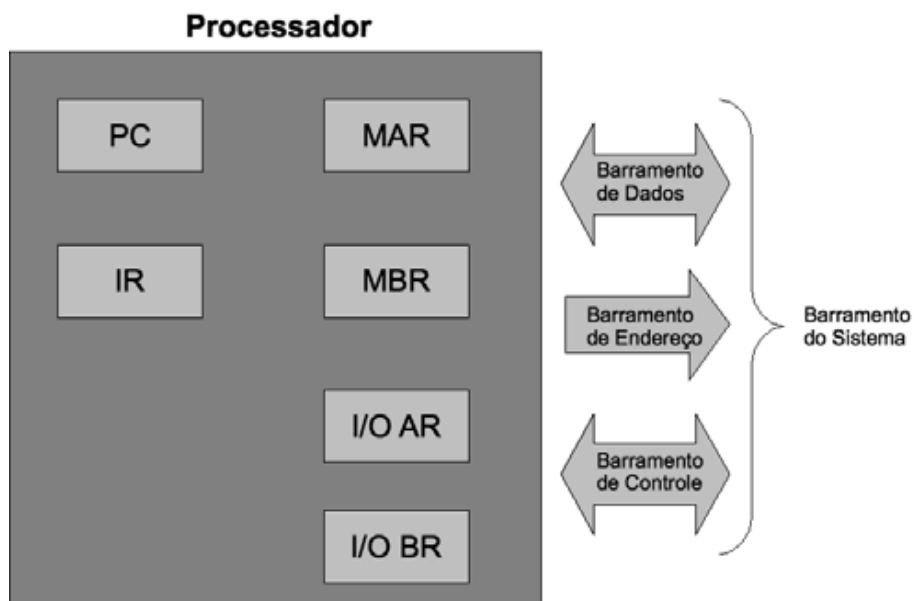


Figura 8 – Visão interna do processador destacando os registradores de controle

Os registradores de controle apresentados na Figura 8 normalmente

não estão acessíveis ao programador, sendo o seu conteúdo controlado pela unidade de controle do processador. Entretanto, ainda assim é muito importante conhecer as suas funcionalidades para se ter uma visão de como o processador funciona.

- » O registrador **PC**, do inglês *Program Counter*, que em português significa **Contador de Programa**, armazena sempre o endereço de memória da próxima instrução a ser executada. Seu conteúdo é atualizado sempre que uma nova instrução é trazida da memória para ser executada.
- » O registrador **IR**, do inglês *Instruction Register*, que em português significa **Registrador de Instrução**, armazena a instrução que foi trazida da memória para ser executada.
- » O registrador **MAR**, do inglês *Memory Address Register*, que em português significa **Registrador de Endereço de Memória**. Este registrador armazena o endereço de memória a ser acessado quando algum operando da operação em execução se encontra armazenado na memória.
- » O registrador **MBR**, do inglês *Memory Buffer Register*, que em português significa **Registrador do Buffer de Memória**. Este registrador é utilizado como buffer de dados entre o processador e a memória principal. Sempre que um operando é lido da memória este é primeiramente armazenado neste registrador antes que possa ser utilizado internamente pelo processador. E, sempre que o processador precisa gravar algum dado na memória principal, este é primeiramente armazenado neste registrador, permanecendo nele até que o processo de gravação na memória seja concluído.
- » O registrador **I/O AR**, do inglês *Input/Output Address Register*, que em português significa **Registrador de Endereço de Entrada e Saída**. Este registrador armazena o endereço do dispositivo de entrada e saída a ser acessado quando o processador vai executar alguma operação de escrita ou leitura nos dispositivos de entrada e saída.
- » O registrador **I/O BR**, do inglês *Input/Output Buffer Register*, que em português significa **Registrador do Buffer de Entrada e Saída**. Este registrador é utilizado como buffer de dados entre o processador e os dispositivos de entrada e saída. Sempre que um dado é lido de algum dispositivo de entrada e saída, este é

primeiramente armazenado neste registrador antes que possa ser utilizado internamente pelo processador. E, sempre que o processador precisa gravar algum dado em algum dispositivo de entrada e saída, este é primeiramente armazenado neste registrador, permanecendo nele até que o processo de gravação seja concluído.

Bem, agora que já conhecemos as principais famílias de processadores existentes no mercado, e que pudemos ter uma visão geral dos elementos presentes na arquitetura interna de todos os processadores, podemos dar uma olhadinha em como esta arquitetura interna funciona, ou seja, como é o fluxo de execução de uma instrução qualquer no interior de um processador.

Do ponto de vista de execução das instruções contidas em um programa, podemos dizer que todos os processadores seguem o fluxo apresentado na Figura 9 a seguir.

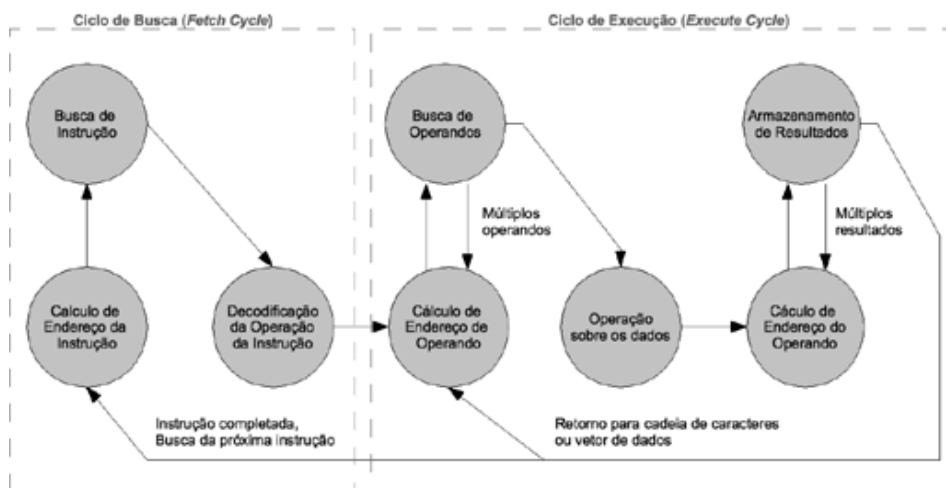


Figura 9 – Fluxo de execução de uma instrução de computador

Este fluxo pode ser dividido em:

► Ciclo de Busca:

- » O registrador Contador de Programa (PC) é carregado com o endereço da instrução a ser executada.
- » O endereço da instrução é inserido no barramento de endereços do sistema juntamente com a ativação dos sinais de controle que indicam que é uma operação de leitura de memória. Em resposta a memória fornece *OP-CODE* da instrução (palavra binária que representa a instrução a ser

executada).

- » O *OP-CODE* da instrução é carregado no Registrador de Instruções para ser decodificado pela Unidade de Controle do Processador.
  - » A Unidade de Controle decodifica a instrução e desencadeia a sequência de operações necessárias à ativação dos módulos que serão utilizados na execução da referida instrução.
- Ciclo de Execução - o ciclo de execução dependerá do tipo de instrução a ser executada, a qual pode ser:
- » Instrução de transferência de dados envolvendo a memória principal (Leitura ou escrita na memória):
    - Neste caso a primeira coisa a ser feita é calcular as posições de memória a serem acessadas, que podem estar direta ou indiretamente indicados no corpo da instrução. Em ambos os casos é a unidade de controle que calcula e/ou define o endereço da posição de memória a ser acessado.
    - O endereço calculado é então armazenado no registrador MAR, e deste é inserido no barramento de endereços do sistema juntamente com os sinais de controle que indicam que é uma operação de leitura ou escrita de memória.
    - Sendo uma operação de leitura, em resposta aos sinais de controle inseridos no barramento do sistema, a memória fornece no barramento de dados do sistema o conteúdo da posição de memória acessada. Ao perceber que o dado está disponível no barramento, a unidade de controle providencia que o mesmo seja armazenado no registrador MBR para que fique disponível para ser processado.
    - Sendo uma operação de escrita, a unidade de controle providencia que o dado a ser escrito seja primeiramente armazenado no registrador MBR, o qual fica então disponível no barramento de dados do sistema para ser gravado pela memória principal.

- » Instrução de transferência de dados envolvendo dispositivos de entrada e saída (Instrução de Escrita ou Leitura em algum dispositivo de Entrada e Saída, também conhecida como operação de I/O)
  - Da mesma forma que na operação de transferência de dados envolvendo a memória principal, neste caso a primeira coisa a ser feita é calcular o endereço do dispositivo a ser acessado, o qual pode estar direta ou indiretamente indicado no corpo da instrução. Em ambos os casos a unidade de controle calcula e/ou define o endereço do dispositivo a ser acessado.
  - O endereço calculado é então armazenado no registrador I/O AR, e deste é inserido no barramento de endereços do sistema juntamente com os sinais de controle que indicam que é uma operação de leitura ou escrita de I/O.
  - Sendo uma operação de leitura, em resposta aos sinais de controle inseridos no barramento do sistema, o dispositivo acessado devolve o dado requerido diretamente no barramento de dados do sistema. A unidade de controle ao perceber que o dado está disponível se encarrega de armazená-lo no registrador I/O BR ficando então o dado disponível para ser processado.
  - Sendo uma operação de escrita, a unidade de controle se encarrega de armazenar o dado a ser escrito no registrador I/O BR após o que este fica disponível no barramento de dados do sistema para que fique acessível ao dispositivo endereçado.
- » Instruções Lógicas e Aritméticas:
  - A unidade de controle programa a Unidade Lógica e Aritmética com a operação a ser efetuada.
  - A unidade de controle identifica o tipo e a localização dos operandos.
  - Se estes estiverem armazenados na memória, a unidade de controle calcula o endereço das posições de memória a serem acessados, colocando um a um

os endereços no registrador MAR a partir do que estes são conectados ao barramento de endereço do sistema juntamente com os sinais de controle que indicam que é uma operação de leitura de memória. Em resposta aos sinais de controle inseridos no barramento do sistema, a memória insere no barramento de dados do sistema, um a um, todos os dados armazenados nas posições indicadas os quais são carregados no registrador MBR e só então podem ser lidos pelo processador para carga nos registradores de trabalho da Unidade Lógica e Aritmética.

- Se estiverem armazenados nos registradores, a unidade de controle gera os sinais de controle necessários para acessá-los, conectando-os diretamente à entrada da Unidade Lógica e Aritmética.
- A Unidade Lógica e Aritmética efetua a operação programada, deixando o resultado disponível à saída do seu registrador de trabalho.
- A unidade de controle identifica o tipo e a localização dos operandos que receberão os resultados da operação. Observe que a quantidade de operandos envolvidos depende da operação em curso.
- Se estes forem ser armazenados na memória principal, primeiramente a unidade de controle calcula e/ou define os endereços das posições de memória a serem acessadas, colocando um a um os dados a serem escritos no registrador MBR e o endereço da posição de memória onde os mesmos deverão ser escritos no registrador MAR, ficando desta forma os mesmos disponíveis no barramento de dados e endereço do sistema juntamente com os sinais de controle que indicam que é uma operação de escrita de memória. Em resposta a memória armazena o conteúdo fornecido nas posições de memória que forem indicadas.
- Se estes forem ser armazenados em registradores, a unidade de controle gera os sinais de controle

necessários para acessá-los, conectando-os diretamente à saída da Unidade Lógica e Aritmética, juntamente com os sinais que indicam que é uma operação de escrita em registradores.

» Instruções de Desvio

- Caso seja uma instrução de chamada de desvio com retorno programado, ou seja, uma chamada de sub-rotina, a unidade de controle primeiramente armazena o conteúdo do registrador Contador de Programa (PC) em algum lugar para ser utilizado posteriormente como endereço de retorno, calcula o endereço para onde o programa deverá ser desviado e carrega o registrador Contador de Programa (PC) com o novo endereço.
- Caso seja uma instrução de chamada de desvio sem retorno, a unidade de controle simplesmente calcula o endereço para onde o programa deverá ser desviado e carrega o registrador Contador de Programa (PC) com o novo endereço.
- Caso seja uma instrução de retorno programado, ou seja, um retorno de uma sub-rotina, a unidade de controle carrega o registrador Contador de Programa (PC) com o endereço anteriormente armazenado.

Como podemos observar, ainda que muito superficialmente nestas pequenas descrições, o processo de buscar, decodificar e executar uma instrução não é uma tarefa simples sendo em parte o grande diferencial entre um processador de alto desempenho e um processador mais simples.

Bem, agora que conhecemos um pouco como é e como funciona a arquitetura interna de um processador em geral, podemos continuar com o nosso estudo sobre o MIPS.

## O processador MIPS

Fruto de um projeto de pesquisa arrojado de um grupo de pesquisadores liderados pelos cientistas David Patterson e Carlo Séquin, o processador MIPS dominou por muitos anos o mercado de equipamentos eletrônicos em geral. Para se ter uma ideia, só em 1997

as vendas deste processador superaram a casa dos 19 milhões de unidades. Em 2007, com mais de 250 milhões de clientes ao redor do mundo, os processadores MIPS dominavam absolutos o mercado de equipamento de entretenimento digital. Atualmente, os processadores MIPS ainda detêm uma boa fatia do mercado de equipamentos eletrônicos, estando presentes em equipamentos como:

- » Wireless Access Point
- » Máquina Fotográfica Digital
- » Console de Jogos Eletrônicos
  - Nintendo 64
  - Sony Playstation e Sony Playstation II
  - Sony PSP
- » Computadores de grande porte

## Arquitetura Interna do MIPS

Diferentemente dos demais processadores existentes à época do seu lançamento, o processador MIPS foi projetado de forma a ter uma arquitetura simples e regular, com um pequeno número de instruções que pudessem ser decodificadas e executadas em apenas um ciclo de relógio.

Isto causou uma certa estranheza e descrédito por parte dos demais fabricantes de processadores, os quais construíam processadores extremamente complexos, que ao contrário do MIPS possuíam centenas de instruções, algumas das quais tidas como extremamente flexíveis e poderosas e que levavam entre dezenas e centenas de ciclos de relógio para serem decodificadas e executadas.

De um modo geral, o objetivo destes fabricantes era prover a linguagem de máquina dos seus processadores com instruções cada vez mais próximas das utilizadas nas linguagens de alto-nível, a fim de poder reduzir o tamanho dos programas e assim tornar sua execução cada vez mais rápida e eficiente.

Entretanto, devido às limitações de tecnologia existentes à época, os chips ainda eram relativamente pequenos se comparados com os produzidos hoje em dia, possuindo apenas algumas centenas de milhares de transistores, muitas destas instruções não podiam



ser implementadas diretamente em hardware, tendo assim que ser implementadas através de um artifício que ficou conhecido como **microcódigo**.

Um **microcódigo** nada mais era que uma referência a um pequeno trecho de código que ficava armazenado em uma memória especial dentro do próprio processador. Como as instruções deste trecho de código ficavam armazenadas dentro do próprio processador, elas podiam ser acessadas muito mais rapidamente que o restante das instruções que ficavam armazenadas na memória principal. Esta diferença na velocidade de acesso permitia que um programa que utilizasse instruções baseadas em microcódigo executasse muito mais rapidamente que um programa que não as utilizasse.

Ou seja, sempre que um fabricante de processador percebia que uma determinada operação mais complexa estava sendo utilizada com grande frequência pelos programadores em geral, a qual normalmente era implementada através de uma sub-rotina, utilizando várias instruções básicas da linguagem de máquina do processador, estes criavam um novo microcódigo com este conjunto de instruções e definiam uma nova instrução na linguagem de máquina do processador para lhe fazer referência.

Desta forma, devido a sua praticidade e pelos resultados alcançados, o uso dos microcódigos tinha se tornado a melhor solução para aumentar o desempenho e a complexidade dos processadores.

Os processadores MIPS vinham portanto quebrar estes paradigmas, eliminando por completo o uso dos microcódigos e reduzindo drasticamente o número de instruções da linguagem de máquina do processador, dando assim início a uma nova geração de processadores que ficou conhecida como processadores RISC, acrônimo da língua inglesa que traduzido ao pé da letra significa Processador com Conjunto Reduzido de Instruções, conforme abordamos no capítulo 1 deste volume. Os demais processadores, os que seguiam a abordagem padrão, em contrapartida, ficaram conhecidos como processadores CISC, também um acrônimo da língua inglesa que traduzido significa Processador com Conjunto de Instruções Complexo.

O outro diferencial da arquitetura do MIPS, como já citamos, era a presença de um grande número de registradores de propósito geral, os quais poderiam ser utilizados como espaço de memória de acesso

ultrarrápido era um outro ponto. Vale lembrar que um registrador nada mais é que um espaço de memória com o tamanho exato da palavra do processador construído dentro do próprio processador e ao qual este tem acesso quase que instantâneo, diferentemente da memória principal a qual é construída fora do processador e portanto tem um acesso bem mais lento.

E, para completar o cenário, um outro ponto que diferenciava o processador MIPS dos demais era a exigência de que os operandos para as operações lógicas e aritméticas estivessem armazenados exclusivamente em registradores ou diretamente no corpo da instrução. Com isto os projetistas do MIPS objetivavam reduzir ao máximo o acesso à memória principal e assim permitir que seus programas pudessem ser executados ainda mais rapidamente. Ou seja, o grande número de registradores tinha um objetivo claro e direto: otimizar uso da unidade lógica e aritmética do processador.

Os demais processadores, por outro lado, permitiam que os operandos estivessem armazenados tanto nos registradores quanto na memória principal, o que era claramente um dos motivos do seu baixo desempenho.

Apenas a título de comparação, as Figuras 10 e 11 a seguir nos trazem o ciclo completo de buscar, decodificar e executar das instruções lógicas e aritméticas sendo executadas num processador clássico e no MIPS. Observe que no MIPS como os operandos já estão armazenados nos registradores, o acesso a memória principal pode ser reduzido drasticamente, bem como o processo de acesso aos operandos fica bem simplificado.



Figura 10 – Execução de uma instrução lógica ou aritmética nos processadores em geral

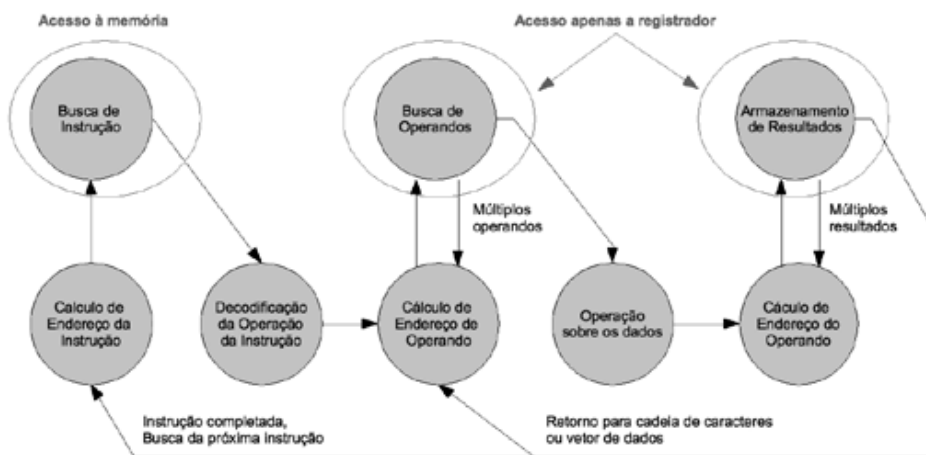


Figura 11 – Execução de uma instrução lógica ou aritmética no MIPS

Como você deve ter percebido, tanto o MIPS quanto os demais processadores executam basicamente as mesmas ações durante o processo de buscar, decodificar e executar uma instrução. Ou seja, o que muda é como estas ações são implementadas pela arquitetura interna do processador.

Bem, olhando agora para a arquitetura do MIPS em comparação com uma arquitetura clássica da época, pode até fazer algum sentido imaginar que este pudesse ter até um bom desempenho, mas para os fabricantes de processadores de época, isto não fazia nenhum sentido. Como diziam eles, um processador com tão poucas instruções e ainda por cima com instruções tão simples, que não faz uso de microcódigos e que só permite operações lógicas e aritméticas com operandos já armazenados em registradores poderia ter um bom desempenho? Um programa escrito para este processador deve ficar enorme... Como então este processador pode ter um desempenho superior aos demais?

A resposta a esta pergunta é ao mesmo tempo simples e complexa.

O grande desempenho do MIPS advinha de algumas decisões de projeto muito bem arquitetadas, que resultaram em um conjunto de instruções simples e regular, o que permitiu implementar um esquema de *pipeline* extremamente eficiente, de tal forma que toda e qualquer instrução podia ser decodificada e executada em um único ciclo de clock (conforme estudamos no capítulo 1), e um grande número de registradores de propósito gerais totalmente conectados à Unidade

Lógica e Aritmética, de tal modo que qualquer destes registradores pudesse ser utilizado tanto como operando como destino das operações lógicas e aritméticas.

Isto se comparado à estratégia dos processadores CISC, que levavam dezenas e até centenas de ciclos de clock para executar uma única instrução e que permitiam que os operandos das operações lógicas e aritméticas estivessem armazenados diretamente na memória principal, fazia com que, mesmo seus programas sendo duas ou até três vezes maiores que os demais, executassem muito mais rapidamente que seus equivalentes em um processador CISC.

Muito interessante, não é mesmo? Mas que tal agora darmos uma olhada mais de perto nas instruções da linguagem de máquina do MIPS para em seguida vermos como estas instruções eram decodificadas e executadas em hardware?

## Linguagem de Máquina do MIPS

Conforme dissemos, devido aos princípios de simplicidade e regularidade, todas as instruções do MIPS podem ser divididas em apenas três tipos básicos, descritos a seguir:

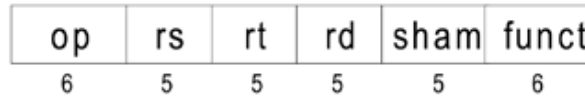
- » Instruções **Tipo R**: Instruções em que todos os operandos se encontram armazenados em registradores.
- » Instruções **Tipo I**: Instruções onde um dos operandos é fornecido na forma de um valor imediato dentro da própria instrução e os demais encontram-se armazenados em registradores.
- » Instruções **Tipo J**: Este formato de Instrução é utilizado exclusivamente para representar instruções de desvio, conhecidas como instruções Jump.

As figuras a seguir (12, 13 e 14) demonstram como estão divididos os 32 bits da palavra do processador para cada uma destas classes de instruções. Mais tarde veremos como esta organização permitiu uma decodificação rápida e direta destas instruções.



### Saiba Mais

**JUMP** é uma palavra inglesa que pode ser traduzida como salto ou pulo.



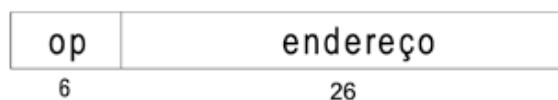
- op: operação básica a ser realizada (opcode)
- rs: primeiro operando
- rt: segundo operando
- rd: registrador de destino (resultado)
- shamt: quantidade de bits a ser deslocado
- funct: função específica a ser executada

Figura 12 – Formato interno das instruções tipo R



- op: operação básica a ser realizada (opcode)
- rs: registrador base a ser operado com o valor imediato ou operando
- rt: registrador de destino ou operando
- imediato: valor constante a ser operado

Figura 13 – Formato interno das instruções tipo I



- op: operação básica a ser realizada (opcode)
- endereço: endereço da instrução para onde o programa deve ser desviado

Figura 14 – Formato interno das instruções tipo J

Para que possamos entender como os princípios de simplicidade e regularidade estão presentes no formato interno de cada uma destas classes de instruções, vamos primeiramente analisar o significado de cada um dos campos assinalados.

Você observou que todas as classes de instruções possuem um campo **op** com 6 bits. É neste campo que é definida a classe e o

tipo de instrução representado. Se, por um lado o pequeno número bits deste campo limita em 64 o número máximo de classes ou tipos de instruções disponíveis nos processadores MIPS, por outro lado a sua decodificação pode ser feita quase que instantaneamente com o auxílio de uma tabela ou através de um circuito lógico combinacional extremamente simples.

Como você deve ter notado, o conteúdo dos outros campos representam os operandos e outras informações necessárias à execução da operação representada. Veja inclusive que os nomes e funções dos campos são praticamente os mesmos em todas as classes de instruções.

Continuando nossa análise, vamos fazer uma análise mais detalhada no formato das instruções tipo R.

## Instruções Tipo R

Como dissemos, as instruções tipo R são aquelas em que todos os operandos se encontram armazenados em registradores. Como existem diversas instruções deste tipo nos processadores MIPS, existe um segundo campo nestas instruções, o campo **func**, o qual determina qual destas instruções deve ser executada. Ou seja, quando a unidade de controle recebe uma instrução para ser decodificada e, pelo valor armazenado no campo **op** verifica que é uma instrução do tipo R, este utiliza o valor armazenado no campo **func** para determinar qual a instrução deverá ser executada.

Determinada a instrução a ser executada, a unidade de controle utiliza os campos **rs**, **rt** e **rd** para acessar o conteúdo dos dois operandos, e para endereçar o registrador de destino onde será armazenado o resultado obtido.

Observe que os campos **rs**, **rt** e **rd** possuem 5 bits cada, o que é suficiente para armazenar o endereço de todos os 32 registradores presentes no MIPS.

Vejamos agora o que muda nas instruções tipo I.

## Instruções Tipo I

A decodificação das instruções tipo I é bem parecida com o que acabamos de ver. Após verificar pelo conteúdo do campo **op** que trata-

se de um instrução tipo I, uma vez que esta não possui o campo **func**, a unidade de controle pode passar a acessar os operandos, dos quais um se encontra no registrador apontado pelo conteúdo do campo **rs** e o outro no campo imediato. O resultado obtido, quando necessário, é armazenado no registrador apontado pelo campo **rt**.

## Instruções Tipo J

Por fim, a decodificação das instruções tipo J são ainda mais simples que as duas primeiras apresentadas. Uma vez verificado que trata-se de uma instrução desta classe, a unidade de controle simplesmente obtém o operando do campo endereço o qual é aplicado a ULA para calcular o endereço para o qual a execução do programa será desviada.

Muito simples, não é mesmo?

Observe que, tanto para as operações tipo I quanto para as do tipo J, o processo de decodificação das instruções se resumiu à verificação do conteúdo do campo op. E, mesmo no caso das instruções tipo R, as quais possuem o campo func para determinar qual a instrução dentro da classe será executada, ambos os campos podem ser decodificados simultaneamente, o que simplifica e agiliza o processo de decodificação.

Bem, agora que já sabemos, em linhas gerais, como as três classes de instruções do MIPS estão internamente organizadas e como são decodificadas, vamos conhecer algumas destas instruções da linguagem de máquina do MIPS.

## Instruções do MIPS

Do ponto de vista de funcionalidade, as instruções da linguagem de montagem do MIPS podem ser divididas em:

- » Instruções Lógicas e Aritméticas
- » Instruções de transferência de dados
- » Instruções de tomada de decisão
- » Instruções de desvio

Infelizmente, devido o pouco tempo que dispomos, veremos apenas algumas instruções de cada uma das classes apresentadas.

Veremos apenas o suficiente para que possamos entender na prática o funcionamento interno do processador. Aos que quiserem se aprofundar no assunto, recomendamos a leitura do livro Organização e Projeto de Computadores constante em nossa bibliografia.

Vamos começar pelas operações lógicas e aritméticas.

## Instruções Lógicas e Aritméticas

Como dissemos a princípio, tão importante quanto conhecer as instruções de uma determinada linguagem é saber como utilizá-las corretamente. Em nosso caso isto significa conhecer os mnemônicos, sua funcionalidade e a sua sintaxe, ou seja, a maneira correta de escrever um comando utilizando uma determinada instrução.

Para nossa felicidade, todas as instruções Lógicas e Aritméticas possuem uma mesma sintaxe. Todas devem ser escritas da seguinte maneira:

***Instrução RegistradorDeDestino, PrimeiroOperando, SegundoOperando***

Vamos analisar o formato destas instruções:

» A primeira coisa que temos que ter em mente é que, conforme dissemos, o processador MIPS só permite que operações lógicas e aritméticas sejam operadas sobre operandos já armazenados em registradores ou entre um valor armazenado em um registrador em um valor que já esteja presente no corpo da instrução, chamado de valor imediato. Temos que lembrar também que o resultado de toda e qualquer operação lógicas ou aritmética do MIPS deve ser automaticamente armazenado em um registrador, veja o detalhe na saída da ULA na Figura 3. Por este motivo, a sintaxe das instruções lógicas e aritméticas exige a referência a três operandos na seguinte ordem:

- O registrador de destino, ou seja, aquele registrador que receberá o resultado da operação,
- O primeiro operando, que obrigatoriamente será um registrador,
- O segundo operando, que tanto poderá ser um registrador ou um valor numérico qualquer. Observe que se o segundo operando for um registrador teremos uma instrução tipo R



e se este for um valor numérico teremos uma instrução tipo I.

A Tabela a seguir nos traz alguns exemplos de instruções Lógicas e Aritméticas. Nestes exemplos, os parâmetros rd, rs e rt indicam os campos de bits da instrução em linguagem de máquina. No uso da instrução, em linguagem de montagem os mesmos serão substituídos por registradores do banco de registradores do MIPS, conforme podemos ver na coluna Exemplo.

**Tabela 2 – Exemplos de Instruções Lógicas e Aritméticas**

Instrução	Formato	Tipo	Operação	Exemplo → Execução
Add	Add rd, rs, rt	R	$R[rd] = R[rs] + R[rt]$	Add R3, R4, R5 → $R3 = R4 + R5$
Addi	Addi rt, rs, #im	I	$R[rt] = R[rs] + \#im$	Addi R3, R3, 23 → $R3 = R3 + 23$
Sub	Sub rd, rs, rt	R	$R[rd] = R[rs] - R[rt]$	Sub R7, R2, R9 → $R7 = R2 - R9$
And	And rd, rs, rt	R	$R[rd] = R[rs] \& R[rt]$	Add R9, R6, R3 → $R9 = R6 \& R3$
Andi	Andi rt, rs, #im	I	$R[rd] = R[rs] \& \#im$	Andi R3, R4, 25 → $R3 = R4 \& 25$
Or	Or rd, rs, rt	R	$R[rd] = R[rs]   R[rt]$	Or R2, R5, R7 → $R2 = R5   R7$
Ori	Ori rt, rs, #im	I	$R[rd] = R[rs]   \#im$	Ori R4, R2, 33 → $R4 = R2   33$

Muito simples, não é mesmo?

Vamos conhecer agora as instruções de transferência de dados.

## Instruções de Transferência de Dados

Esta classe de instruções é destinada exclusivamente à transferência dos dados da memória principal para o banco de registradores e do banco de registradores para a memória principal.

No MIPS todo o acesso aos dados armazenados na memória principal é feito de maneira indireta, ou seja, o endereço de memória a ser acessado deve ser calculado somando-se o conteúdo de um registrador com um valor presente no campo imediato da instrução.

Esta estratégia se fez necessária para manter todas as instruções com 32 bits. Se fosse definida uma instrução que permitisse o acesso direto aos dados, com o endereço de acesso presente no corpo da instrução, fatalmente esta instrução iria ter mais de 32 bits e sairia fora do padrão estabelecido.

A seguir temos alguns exemplos desta classe de instruções.

Tabela 3 – Exemplos de Instruções de Transferência de Dados

Instrução	Formato	Tipo	Operação	Exemplo → Execução
Lw	Lw rt, #im(rs)	I	$R[rt] = M[R[rs] + Im]$	Lw R3, 3(R5) → $R3 = M[R5 + 3]$
Sw	Sw rt, #im(rs)	I	$M[R[rs] + Im] = R[rt]$	Sw R3, 3(R5) → $M[R5 + 3] = R3$

Vamos analisar primeiramente a instrução Lw.

A instrução Lw carrega o registrador indicado no campo **rt** com o conteúdo da posição de memória indicada pelo conteúdo do registrador indicado no campo **rs** somado ao valor do campo **#im**. Parece um pouco confuso, não é mesmo? Mas vejamos os exemplos a seguir, e eu creio que tudo ficará mais claro.

Na Figura 15 a seguir podemos ver um exemplo do que ocorre durante a execução da operação Sw. Como podemos ver, neste exemplo o campo **rt** foi substituído pelo registrador R3, o campo **rs** pelo registrador R5 e o campo **#im** tem o valor 8. Ao somarmos o conteúdo do registrador R5 com o valor do campo imediato obtemos  $10 + 8 = 18$ , que é o endereço de memória onde será gravado o conteúdo do registrador R3.

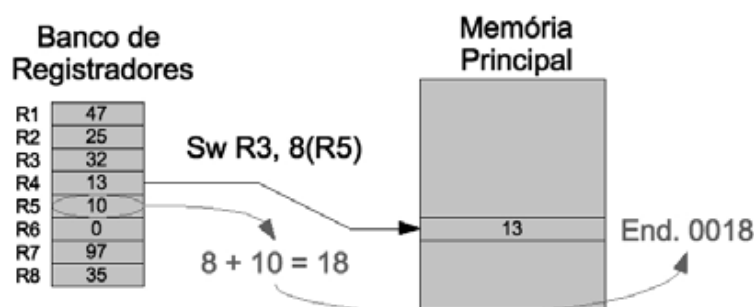


Figura 15 - Exemplo de operação Sw

Na Figura 16 a seguir temos um outro exemplo de execução da

operação Sw. Desta vez o campo rt foi substituído pelo registrador R7, o campo rs pelo registrador R2 e o campo #im pelo valor 2. Ao somarmos o conteúdo do registrador R3 com 2 obtemos:  $25+3 = 28$ , que é o endereço de memória onde o conteúdo do registrador R7 será armazenado.

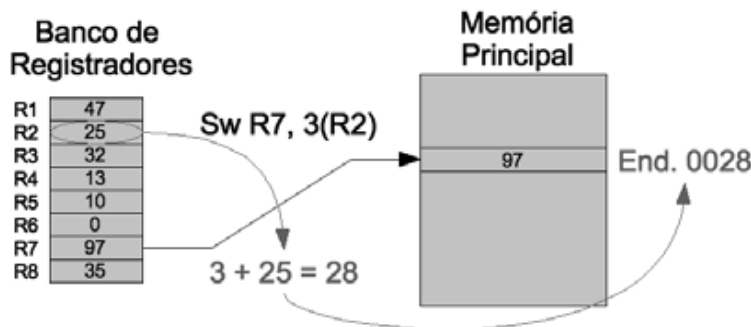


Figura 16 - Exemplo de operação Sw

Muito simples, não é mesmo?

Para concluir, vejamos agora alguns exemplos de uso e execução da operação Lw.

A Figura 17, a seguir, nos traz um exemplo de utilização da instrução Lw onde rt foi substituído por R1, rs por R8 e #im por 3. Observe que a operação Lw funciona de maneira inversa à operação Sw, em vez de salvar o conteúdo do registrador na memória ela carrega o registrador com o conteúdo armazenado na posição de memória que for acessada. Nesta caso, como R8 tem o valor 35 e o campo #im tem o valor 3 estarmos acessando o endereço 38, o qual tem o valor 97 armazenado. Com a execução da operação uma cópia deste valor é então carregado no registrador R1. O exemplo da Figura 18 é análogo a esse, tente entender por você mesmo.

Agora, para terminar, só uma perguntinha. Do ponto de vista de formato interno, o que você acha, a que classe de instruções pertencem as instruções Lw e Sw? A classe R, I ou J?

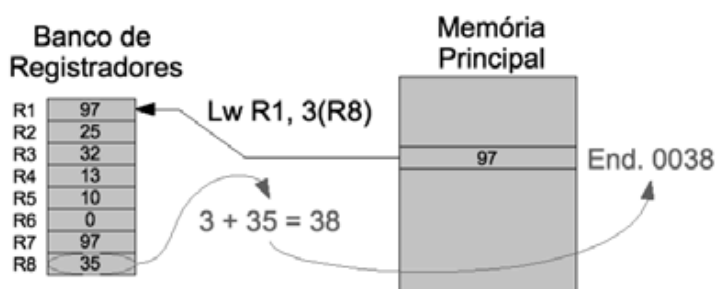


Figura 17 - Exemplo de execução da operação Lw

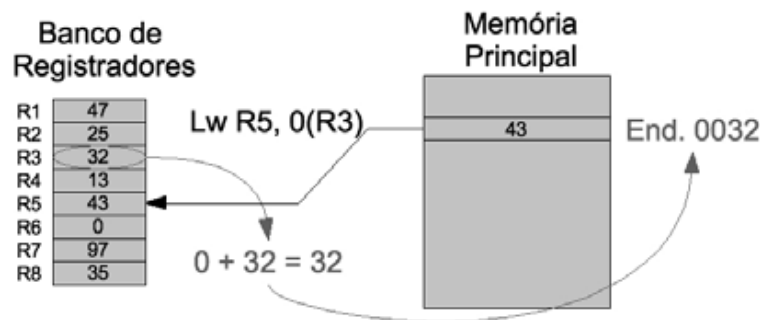


Figura 18 - Exemplo de execução da operação Lw

## Instruções de tomada de decisão

O MIPS possui dois tipos de instruções de tomada de decisão, as que são conhecidas como instruções de desvio condicional, que são aquelas que podem desviar a execução do programa para um trecho do código ou outro a partir da avaliação de alguma inferência lógica, e as instruções que carregam ou não um registrador com um valor pré-determinado também a partir da avaliação de alguma inferência lógica. A tabela Tabela 4, a seguir, nos traz alguns exemplos desta classe de instruções.

Tabela 4 – Exemplos de instruções de tomada de decisão

Instrução	Formato	Tipo	Operação	Exemplo → Execução
Beq	Beq rs, rt, #im	I	Se $rs == rt$ , desvia para a instrução em #im	Beq R3, R5, 25 → Se $R3 == R5$ desvia para a instrução 25
Bneq	Bneq rs, rt, #im	I	Se $rs != rt$ , desvia para a instrução em #im	Bneq R3, R5, 29 → Se $R3 != R5$ desvia para a instrução 29
Slt	Slt rd, rs, rt	R	Se $rs < rt$ , carrega rd com 1, caso contrário carrega rd com 0	Slt R3,R4,R5 → Se o conteúdo de R4 for menor que o conteúdo de R5, carrega R3 com 1, caso contrário carrega R3 com 0
Slti	Slti rt, rs, #im	I	Se $rs < rt$ , carrega rd com 1, caso contrário carrega rd com 0	Slt R3,R4,34 → Se o conteúdo de R4 for menor que 34, carrega R3 com 1, caso contrário carrega R3 com 0

Normalmente estas instruções são utilizadas para controlar o fluxo de execução de um programa, exatamente como ocorre nas

linguagens de alto nível com as instruções **if**, **while** e **for**. Veja na Figura 19 a seguir um exemplo de como ficaria a tradução de um trecho de código utilizando uma instrução **if**, escrito na linguagem C, quando traduzido para a linguagem de montagem do MIPS.

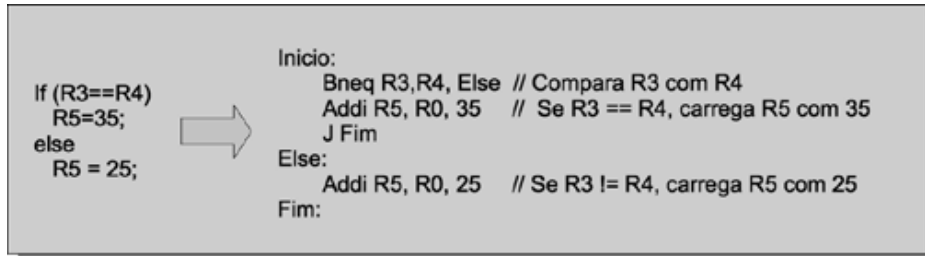


Figura 19 - Exemplo de trecho de código utilizando a instrução if convertido para a linguagem de montagem do MIPS

Eu sei que o trecho de código em linguagem de montagem pode não ter ficado tão claro para alguns, mas o ponto mais importante a destacar é a utilização da instrução **Bneq**. Observe que a depender do seu resultado o programa tomará um fluxo ou outro, ou seja, se o conteúdo de R3 for igual ao conteúdo de R4 o programa continuará o seu fluxo normal, caso contrário o programa será desviado para o trecho indicado com a *Label* Else.

A Figura 20 a seguir nos traz mais um exemplo. Neste exemplo podemos observar o uso das instruções **Beq** e **Slt** juntas para descrever a funcionalidade de uma instrução **if** um pouco mais elaborada. Observe que primeiro foi utilizada a instrução **Slt** para verificar se o conteúdo de R3 era menor que o conteúdo de R4, sendo o resultado desta comparação armazenado em R1. Em seguida utilizou-se a instrução **Beq** para verificar o resultado da comparação anterior e, a depender do resultado obtido, desviar ou não o fluxo de execução do programa.

Muito engenhoso, não é mesmo? Ainda bem que os compiladores se encarregam de fazer estas conversões entre as linguagens de alto nível e a linguagem de montagem para nós.

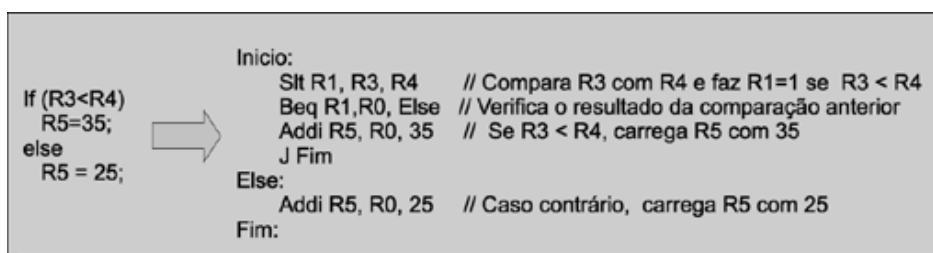


Figura 20 - Exemplo de trecho de código utilizando a instrução Slt



#### Saiba Mais

*Uma Label é um nome que se insere no início de uma linha de código de um programa escrito em linguagem de máquina para poder referenciá-la mais tarde. Durante o processo de montagem todas as Labels são substituídas pelos endereços de memória onde o referido trecho de código ficou armazenado.*

## Instruções de desvio

Em todos os tipos de processadores existem pelo menos dois tipos de instruções de desvio, as instruções de desvio condicional, as quais nós já fomos apresentados quando estudamos as instruções de tomada de decisão, e as instruções de desvio incondicional, as quais iremos estudar agora.

No Mips existem três tipos de instruções de desvio incondicional, conforme podemos ver na Tabela 5 a seguir.

Tabela 5 – Instruções de desvio incondicional

Instrução	Formato	Tipo	Operação	Exemplo → Execução
Jr	J rs	I	PC = rs	J R31 → PC = R31
J	J endereço	J	PC = endereço * 4	J 25 → PC = 100
Jal	Jal endereço	J	R31=PC+4, PC = endereço *4	Jal 25 → R31=PC, PC=100

A primeira instrução é a instrução Jr, que significa Jumper para registrador. Esta instrução permite desviar a execução do programa do endereço que está programado no PC para um outro endereço qualquer que já esteja armazenado em algum outro registrador. Por simples que possa parecer, esta instrução é muito útil e é normalmente utilizada para prover o retorno de sub-rotinas.

A instrução seguinte é a instrução J, que significa simplesmente Jumper. Esta instrução permite desviar a execução do programa do endereço que está programado no PC para um outro que esteja indicado no campo #im.

Aqui precisamos fazer um parêntese em nosso estudo para uma observação muito importante. Como você deve ter observado pelo formato das instruções do tipo J, conforme apresentado na Figura 14, a fim de manter o padrão adotado em todas as instruções do MIPS de reservar 6 bits para o campo **op**, o campo endereço possui apenas 26 bits, o que não seria suficiente para preencher o registrador PC, o qual possui 32 bits. Desta forma não seria possível acessar todas as posições de memória do processador com as instruções da classe J.

A solução a este impasse se deu de uma maneira simples mas não

menos engenhosa que as demais soluções de engenharia adotadas no MIPS.

Como todos sabemos a palavra do processador MIPS possui 32 bits, ou seja 4 bytes, e a memória principal vem com um barramento de endereços que permite que esta seja acessada byte a byte. Observe as Figuras 21, 22 e 23 a seguir. Verifique que se quisermos referenciar as posições de memória palavra por palavra em vez de byte a byte poderíamos reduzir 2 bits em nossa referência, uma vez que os dois bits menos significativo seriam sempre iguais a zero.

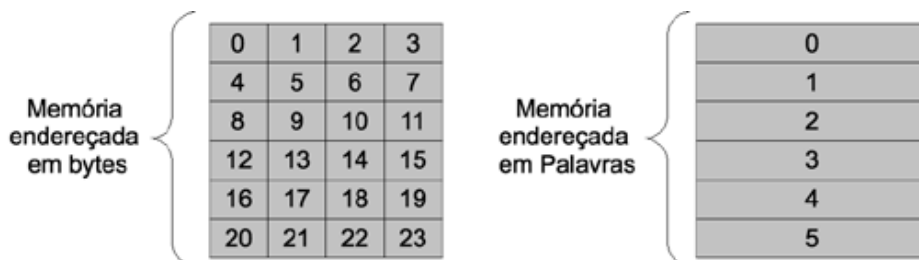


Figura 21 - Diferença entre o endereçamento por bytes e o endereçamento por palavras

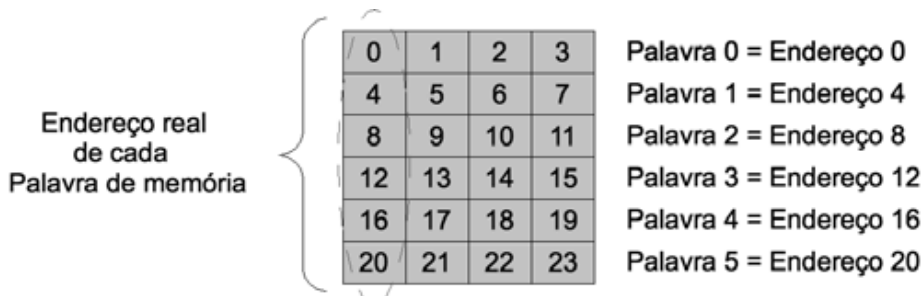


Figura 22 - Endereço real de cada palavra de 32 bits quando armazenada na memória

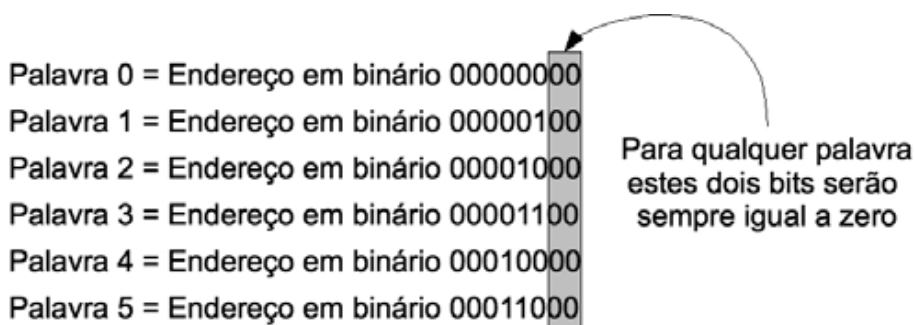


Figura 23 - Endereço em binário para as palavra de 32 bits quando armazenadas na memória

Desta forma, a fim de aumentar o alcance das instruções J e Jal, ficou definido que o campo endereço não conteria o endereço da instrução para onde o programa seria desviado, mas sim uma referência à palavra de 32 bits onde a instrução estivesse armazenada. Com isto era como se o campo endereço tivesse dois bits a mais, passando de 26 para 28 bits. Observe a Figura 24 a seguir, nela temos

um exemplo prático de como se dá a conversão entre a referência, a palavra de memória e o seu endereço real.

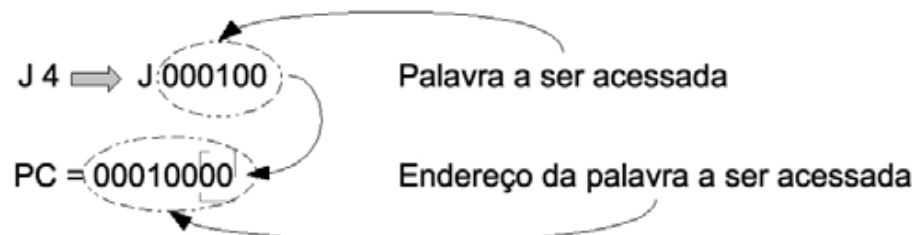


Figura 24 - Conversão entre a referência a palavra de memória a ser acessada e o endereço de memória onde ela se encontra, em binário

Ainda que muito criativa, esta estratégia resolve só em parte o nosso problema, uma vez que ainda ficariam faltando quatro bits para completar os trinta e dois necessários para carregar o PC numa operação de desvio. Observe que se não preencheremos os trinta e dois bits do PC na hora em que formos executar um desvio podemos cair em situações inusitadas como a que pode ser vista na Figura 25 a seguir. Nela temos uma possível configuração do uso da memória de um computador onde quatro programas estão em execução. Observe que cada programa ocupa uma área restrita da memória a qual não é conhecida no momento da compilação do programa. Veja que se não for possível indicar claramente o endereço da posição de memória no momento de executar um desvio, um programa pode acabar sendo desviado para dentro da área de memória do outro.

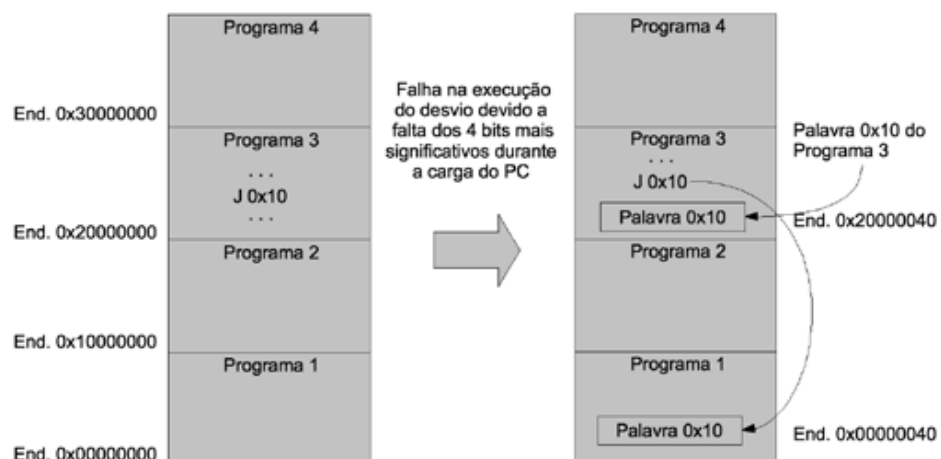


Figura 25 - Falha na execução do desvio devido a falta dos 4 bits mais significativos durante a carga do PC

A solução para situações como esta foi também extremamente simples e funcional, e consiste em manter inalterados os 4 bits mais significativos do PC durante as operações de desvio e sobreescrever os demais com o endereço da palavra de memória que se deseja



acessar. Com isto todos os desvios foram transformados em desvios relativos, ou seja, os desvios são sempre efetuados tomando em conta o endereço atual do programa em execução. Veja como ficaria a execução de uma instrução de desvio com o artifício do desvio relativo ao valor do PC na Figura 26 a seguir. Observe que calcula-se o endereço de memória da palavra a ser acessada, o que como sabemos resulta em uma referência de 28 bits, e simplesmente sobrepõe-se estes 28 bits aos 28 bits menos significativos do PC, deixando os 4 bits mais significativos inalterados.

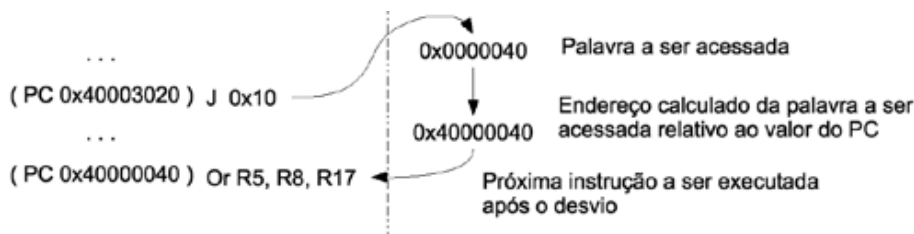


Figura 26 - Exemplo de execução da instrução J com o artifício de calcular o desvio relativo ao valor do PC

Esta estratégia é seguida tanto pelas instruções de desvio condicional como pelas de desvio incondicional, com exceção da instrução Jr que como já foi explicado não necessita de tais artifícios.

Por fim, chegamos à última instrução de desvio incondicional, a instrução Jal, cujo mnemônico é um acrônimo da expressão inglesa **Jump and Link**. Esta instrução foi especialmente projetada para executar desvios com retorno programado, ou seja, para ser utilizada na chamadas a sub-rotinas. A sua execução é em tudo semelhante a instrução J que acabamos de estudar, a única diferença é que antes de se executar a carga do PC com o endereço de desvio o seu conteúdo é copiado para o registrador R31 para ser utilizado posteriormente como endereço de retorno da sub-rotina. Veja um exemplo na Figura 27 a seguir.

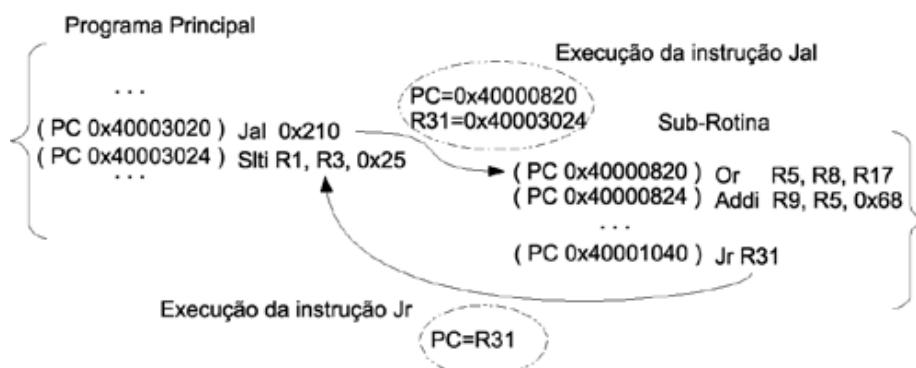


Figura 27 - Exemplo de chamada de sub-rotina com a instrução Jal e retorno com a instrução Jr

Muito interessante, não é mesmo?

Bem, agora para terminar nosso estudo sobre a arquitetura interna do MIPS que tal vermos como algumas destas instruções são implementadas em linguagem de máquina e como elas são executadas diretamente em hardware?

A Tabela 6, a seguir, nos traz algumas instruções da linguagem de montagem do MIPS com os seus respectivos formato interno, descrição simbólica da sua operação e os valores para o campo **op** e para o campo **func**, quando aplicável.

**Tabela 6 – Exemplo de algumas instruções e os seus respectivos valores para os campos op e func**

Instrução	Formato	Operação	Op./Func.
Add	R	$R[rd] = R[rs] + R[rt]$	0/20
Addi	I	$R[rt] = R[rs] + Im$	8
And	R	$R[rd] = R[rs] \& R[rt]$	0/24
Andi	I	$R[rt] = R[rs] \& Im$	C
Beq	I	Desvia p/ a instrução indicada se $R[rs] == R[rt]$	4
Bneq	I	Desvia p/ instrução indicada se $R[rs] != R[rt]$	5
J	J	Desvia para a instrução indicada	2
Jal	J	Desvia para a instrução indicada e salva retorno em R31	3
Lw	I	$R[rt] = M[R[rs] + Im]$	23
Sw	I	$M[R[rs] + Im] = R[rt]$	2b
Sub	R	$R[rd] = R[rs] - R[rt]$	0/22

Vamos começar por um exemplo bem simples. Vamos começar com a instruções Add. Observe a Figura 28 a seguir. Verifique que para converter uma instrução da linguagem de montagem para a

linguagem de máquina primeiramente precisamos identificar a classe da instrução para poder fazer o preenchimento correto dos campos op e func e verificar o seu formato interno, conforme vimos nas Figuras 12, 13 e 14. Neste caso, tomando como referência os valores da Tabela 6 encontramos que op=0x00 e func = 0x20. Em seguida, baseado no formato da instrução, podemos fazer o preenchimento dos demais campos. O campo shant não é utilizado por nenhuma das instruções que estudamos, devendo ser preenchido sempre com 0. Observe que primeiramente preenchemos os campos com os valores em hexadecimal para em seguida convertê-los para binário e só então formarmos a palavra binária de 32 bits que representa a instrução na linguagem de máquina.

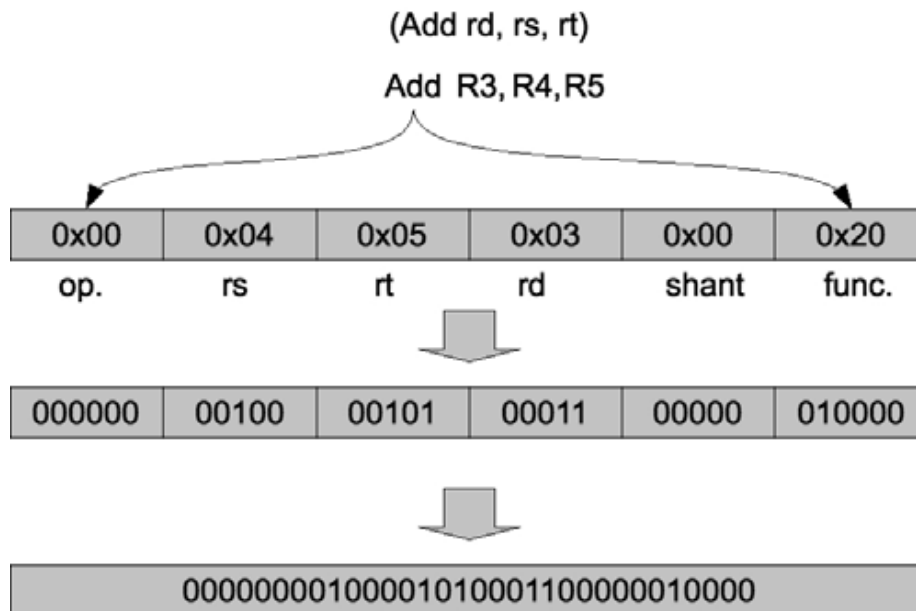


Figura 28 - Exemplo de conversão da instrução Add para a linguagem de máquina do MIPS

Muito simples, não é mesmo?

Vamos agora pegar mais dois exemplos, só que desta vez de instruções da classe I. Vamos pegar as instruções Addi e Beq. A Figura 29 nos mostra como ficaria a conversão destas duas instruções para a linguagem de máquina do MIPS. Muito simples, não é mesmo?

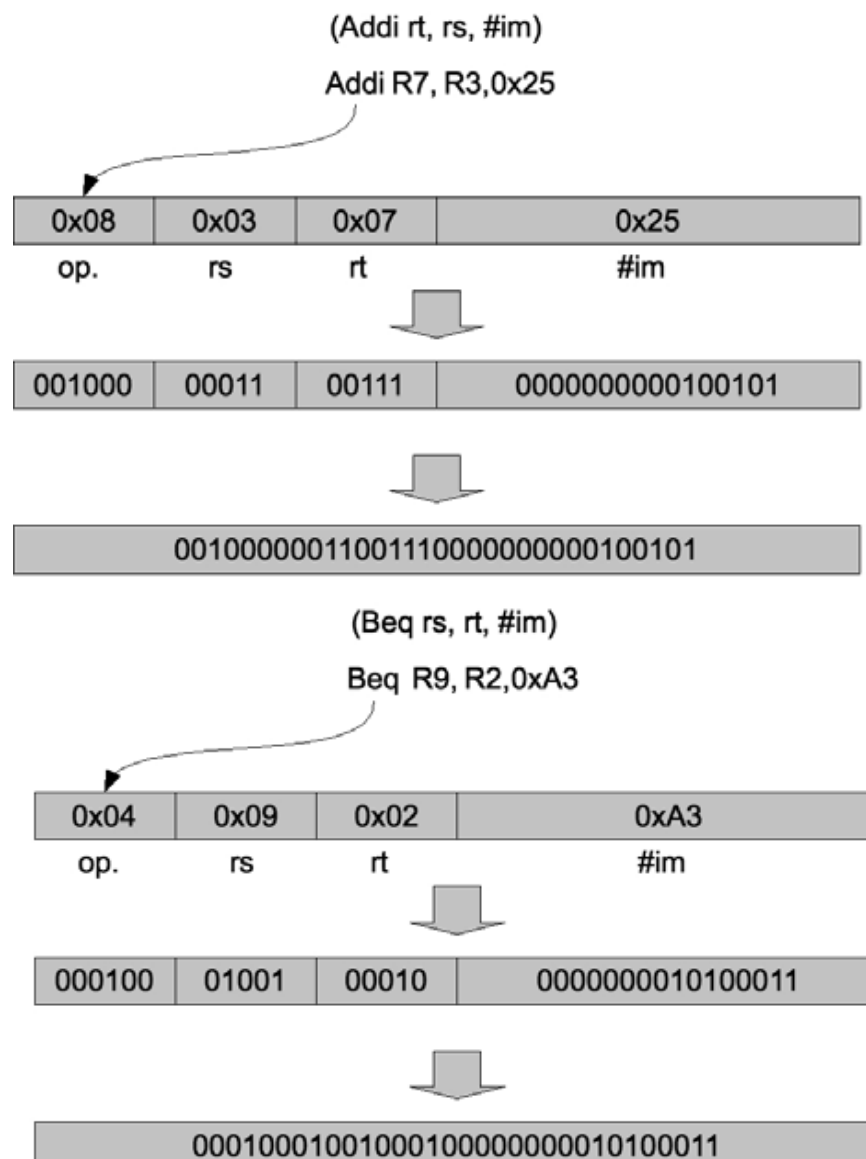


Figura 29 - Exemplo de conversão das instruções Addi e Beq para a linguagem de máquina do MIPS

Bem, eu creio que estes exemplos são suficientes para que possamos ver que o processo de conversão de uma instrução da linguagem de montagem para a linguagem de máquina é extremamente simples e direto.

Agora, para finalizar, vem a melhor parte, vamos dar uma olhadinha em como o processador decodifica e executa estas instruções diretamente em hardware.

## Decodificação e execução das instruções

Antes que uma instrução possa ser executada ela precisa

primeiramente ser decodificada pela unidade de controle do processador.

O processo de decodificação das instruções da linguagem de máquina do MIPS é extremamente simples e pode ser feito da seguinte forma:

- » Separe os 6 bits mais significativos da palavra binária para identificar o valor do campo op.
- » Baseado no valor do campo op, identifique a classe da instrução, se esta é da classe R, I ou J.
- » Sendo da classe R, separe os 6 bits menos significativo para identificar o valor do campo func.
- » A partir dos valores dos campos op e func e, baseado na tabela que associa estes valores às instruções do processador, identifique por fim a instrução a ser executada.
- » Identificada a instrução e o seu formato interno, carregue os seus operandos e programe a Unidade Lógica e Aritmética com a operação a ser efetuada.

Vejamos um exemplo na Figura 30, a seguir. Todas as demais instruções são decodificadas exatamente da mesma forma. O que você achou? Muito fácil, não é mesmo?

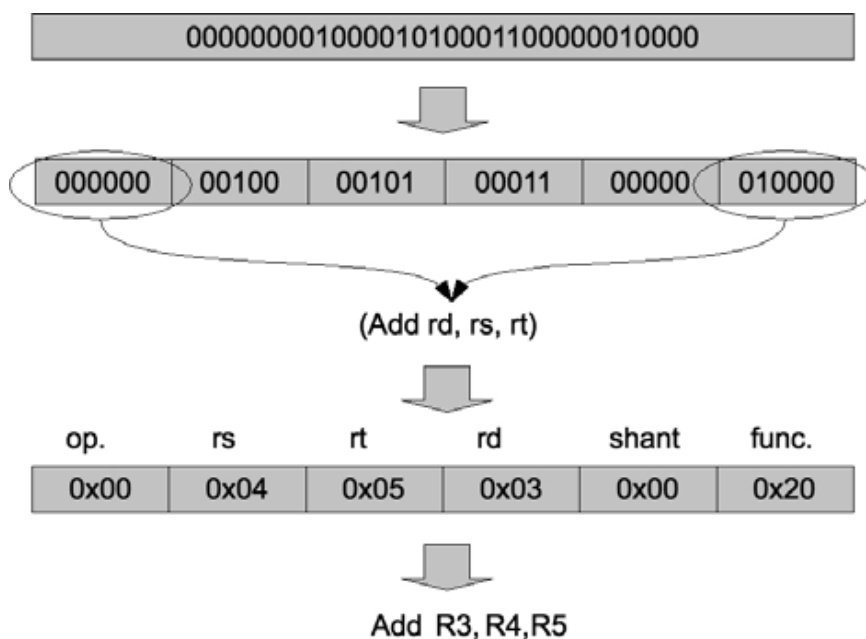


Figura 30 - Exemplo de decodificação da instrução Add

Bem, agora que já sabemos como as instruções são codificadas da linguagem de montagem para a linguagem de máquina e como, uma vez codificadas, estas podem ser decodificadas para que sejam executadas. Estamos prontos para ver como estas instruções são executadas em hardware.

Para que possamos entender como se dá a execução das instruções da linguagem de máquina no hardware interno do MIPS, precisamos primeiramente conhecer como este hardware está constituído. Isto é feito analisando o seu diagrama elétrico. Para tanto vamos começar conhecendo os símbolos utilizados na representação deste diagrama.

O primeiro componente que vamos conhecer é o Mux.

### MUX

O nome Mux é uma abreviação da palavra Multiplexador. Um Multiplexador é um componente eletrônico que funciona como uma chave seletora, que permite selecionar entre duas entradas qual será conectada à saída.

Observe o diagrama de um Multiplexador na Figura 31, a seguir. Temos uma entrada seletora, indicada com a palavra **Select**. Duas entradas, a entrada 0 e a entrada 1, e uma saída indicada com a letra Y. Quando **Select** é igual a 1, a entrada 1 é conectada a saída Y. Quando **Select** é igual a 0, a entrada 0 é conectada a saída Y. Quando dizemos que uma entrada é conectada a saída queremos dizer que a saída Y assumirá o mesmo nível lógico que estiver aplicado a entrada selecionada por todo tempo em que esta estiver selecionada. Se durante este tempo a entrada selecionada mudar o seu nível lógico, digamos de 0 para 1, a saída Y acompanhará esta mudança.

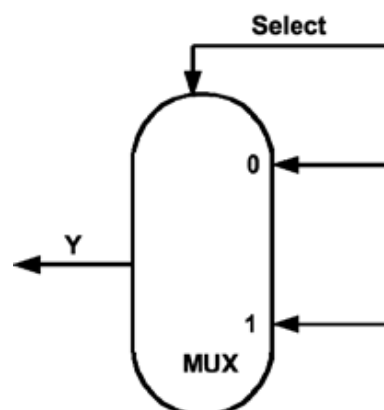


Figura 31 - Diagrama elétrico do Mux

Em seguida temos o Somador.

### Somador

O somador, como o próprio nome define, efetua uma operação de adição entre os valores aplicados às suas entradas. Esta operação segue o que estudamos para números da base Binária utilizando a representação de complemento a dois. A Figura 32, a seguir, nos traz o diagrama elétrico do somador. Nele temos apenas as entradas A e B e a saída indicada como A+B. Tanto as entradas como a saída estão preparadas para trabalhar com uma palavra binária do tamanho da palavra do processador, no caso do MIPS 32 bits.

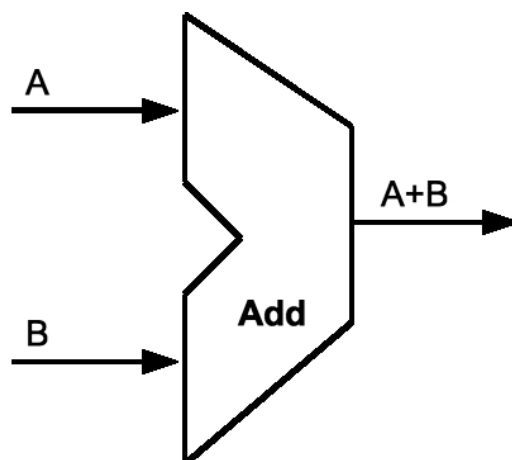


Figura 32 - Diagrama elétrico do somador

Nosso próximo componente é a Unidade Lógica e Aritmética (ULA).

### Unidade Lógica e Aritmética

A Unidade Lógica e Aritmética é responsável por efetuar todas as operações lógicas e aritméticas existentes em qualquer programa. A Figura 33, a seguir, nos traz o diagrama elétrico deste componente.

Como podemos observar, a ULA possui uma entrada para indicar a operação a ser realizada, indicada com o nome Operação, e mais duas entradas para os operandos, sinalizadas como entradas A e B. A ULA possui ainda duas saídas. Uma para fornecer o resultado da operação efetuada e outra para indicar quando o resultado da operação é zero. Tanto as entradas A e B como a saída Resultado estão preparadas para trabalhar com palavras binárias do tamanho da palavra do processador, no caso 32 bits. Já a entrada Operação, irá

possuir tantos bits quanto forem necessários para representar todas as operações que esta consegue executar. Para as versões mais simples do processador MIPS, esta entrada pode possuir apenas 3 bits codificados conforme podemos ver na Tabela 7 a seguir.

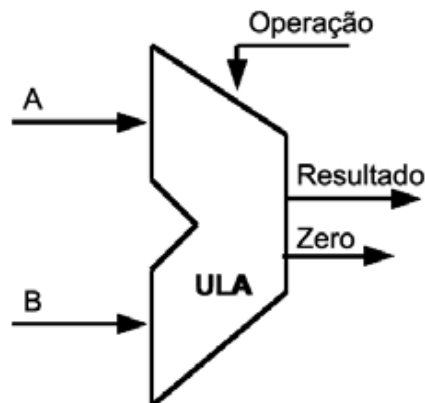


Figura 33 - Diagrama elétrico da Unidade Lógica e Aritmética (ULA)

Tabela 7 – Codificação das funções da ULA nos três bits da entrada Operação

Código	Função
000	AND
001	OR
010	Soma
110	Subtração
111	Compara se $A < B$

Nosso próximo componente interno do processador é o banco de registradores.

### Banco de Registradores

Conforme havíamos citado, um registrador nada mais é que um espaço de memória de acesso ultra-rápido, dentro do próprio processador, capaz de armazenar uma palavra do processador, no caso do MIPS uma palavra com 32 bits.

A fim de ordenar o acesso aos diversos registradores existentes no processador estes ficam agrupados na forma de um Banco de Registradores com 32 registradores, conforme podemos ver na Figura 34 a seguir.



Como podemos observar, o banco de registradores possui cinco entradas e duas saídas assim denominadas:

- » Dado: Porta de entrada para uma palavra de 32 bits a ser escrita no registrador apontado pela porta End. Reg. Dest. durante a ativação do sinal Escrever.
- » End. Reg. rs: Porta de entrada com 5 bits, utilizada para carregar o endereço do registrador associado ao registrador rs.
- » End. Reg. rt: Porta de entrada com 5 bits, utilizada para carregar o endereço do registrador associado ao registrador rt.
- » End. Reg. Dest. : Porta de entrada com 5 bits, utilizada para carregar o endereço do registrador que será escrito.
- » Escrever: Porta de entrada de um bit, associada ao sinal que ativa o processo de escrita no registrador destino.
- » rs: Porta de saída de 32 bits com o conteúdo do registrador associado ao registrador rs
- » rt: Porta de saída de 32 bits com o conteúdo do registrador associado ao registrador rt

Observe que as portas End. Reg. rs, End. Reg. rt e End. Reg. Dest. possuem exatamente 5 bits cada, o que é suficiente para endereçar qualquer um dos 32 registradores do Banco de Registradores.

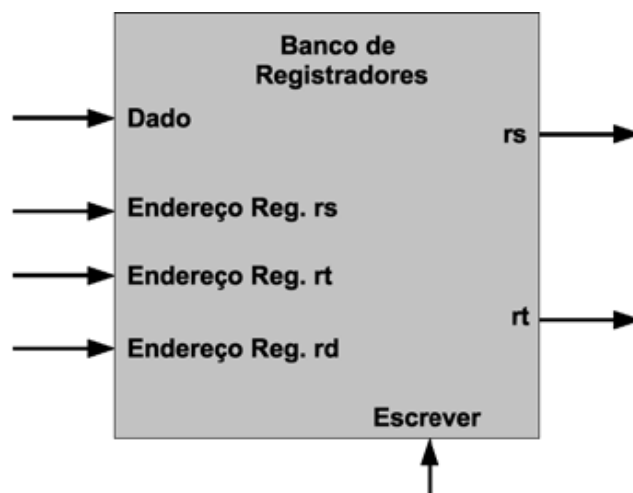


Figura 34 – Banco de Registradores

### Unidade de Controle

Conforme já dissemos, a Unidade de Controle é responsável por

buscar e decodificar as instruções e, a partir da decodificação destas, controlar todos os demais componentes internos do processador. A Figura 35, a seguir, nos traz o símbolo utilizado para representar a Unidade de Controle no diagrama interno do processador.

Como era de se esperar, esta unidade recebe como entrada os bits dos campos op e func da instrução em linguagem de máquina e, a partir do conteúdo destes campos, gera os seguintes sinais de saída:

- » RegDest: Sinal que seleciona a partir de um Mux, que conjunto de bits da instrução em linguagem de máquina será utilizado para endereçar o registrador de destino. Assumindo nível lógico 0 para as instruções tipo R e nível lógico 1 para as instruções tipo I, selecionando respectivamente os bits 11-15, registrador rd, ou 16-20, registrador rt, como endereço do registrador de destino.
- » RegWrite: Sinal que ativa a escrita no banco de registradores
- » AluSrc: Sinal que controla mux que seleciona qual será o segundo operando da Unidade Lógica e Aritmética. O primeiro operando, como veremos um pouco mais a frente, será sempre o conteúdo do registrador rs. Este sinal assume nível lógico 0 para as instruções tipo R, indicando que o segundo operando será o conteúdo do registrador rt, e o nível lógico 1 para instruções tipo I, indicando que o segundo operando será o conteúdo imediato já presente na instrução.
- » MemRead: Sinal que ativa a leitura da memória de dados.
- » MemToReg: Sinal que controla o mux que seleciona o conteúdo a ser escrito no banco de registradores. Este sinal assume nível lógico 1 quando está sendo executada a instrução lw, indicando que o dado lido da memória será escrito em algum registrador, assumindo nível lógico 0 para todas as demais instruções, deixando resultado presente à saída da ULA disponível para ser escrito no banco de registradores.
- » Operação: Palavra com 3 bits que indica a operação a ser efetuada pela ULA.
- » Branch: Sinal que indica que está em curso uma instrução de desvio condicional. Mais a frente veremos melhor o seu funcionamento.

- » **MemWrite:** Sinal que ativa a escrita na memória de dados. Este sinal é ativo durante a execução de instruções de transferência de dados do banco de registradores para a memória de dados, como em instruções instrução sw (store word) por exemplo.

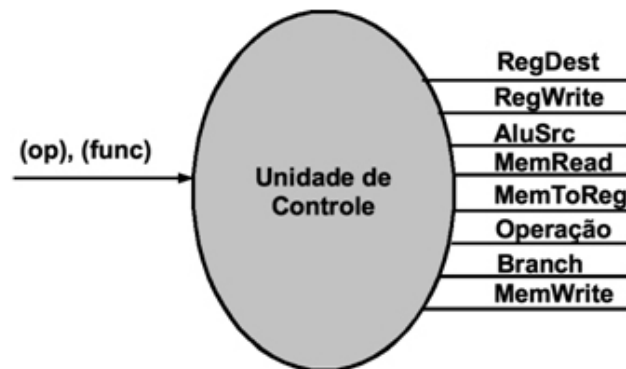


Figura 35 - Diagrama elétrico da Unidade de Controle

Por fim, chegamos aos dois últimos elementos da arquitetura interna do MIPS, a *Memória de Dados* e a *Memória de Instruções*. Na verdade, normalmente, existe apenas uma memória principal a qual armazena tanto os dados quanto as instruções a qual fica fora do processador. Entretanto, apenas como um recurso didático, para facilitar a compreensão do funcionamento interno do processador, não apenas separamos em Memória de Dados e Memória de Instruções como as incluímos fazendo parte da arquitetura do processador.

### Memória de Dados

A Memória de Dados representa o espaço de memória onde estão armazenados os dados do programa, tanto variáveis como constantes. A Figura 36, a seguir, nos traz o símbolo utilizado para representar a Memória de Dados na arquitetura interna do MIPS.

Observe que esta contém quatro portas de entradas e uma de saída assim denominadas:

- » **Endereços:** Com 32 bits, corresponde a porta por onde é fornecido o endereço da posição de memória a ser acessada, tanto em operações de leitura como de escrita.
- » **Dados:** A memória de dados possui duas portas identificadas como portas de dados. Uma de entrada e uma de saída, ambas com 32 bits. A porta de escrita corresponde à porta por onde é fornecido o dado a ser escrito na memória em operações

de escrita. A porta de leitura corresponde à porta por onde os dados presentes na memória podem ser lidos em operações de leitura.

- » Escrever: Com 1 bit, corresponde ao sinal que habilita a escrita na memória.
- » Ler: Com 1 bit, corresponde ao sinal que habilita a leitura da memória.

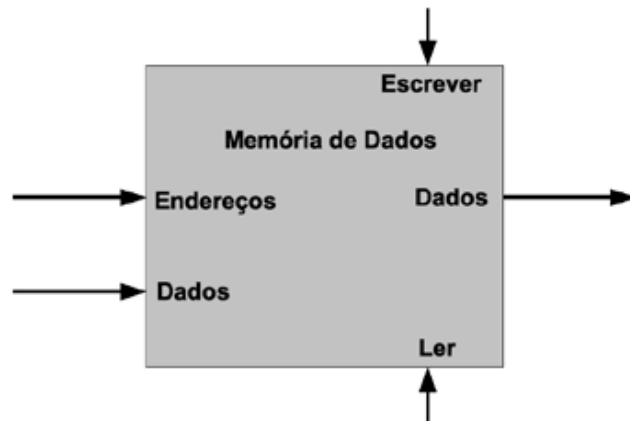


Figura 36 - Diagrama elétrico da memória de dados

### Memória de Instruções

A Memória de Instruções representa o espaço de memória onde estão armazenadas as instruções do programa a ser executado. A Figura 37, a seguir, nos traz o símbolo utilizado para representar a Memória de Instruções na arquitetura interna do MIPS.

Observe que esta contém apenas uma porta de entrada e uma porta de saída assim denominadas:

- » Endereços: Com 32 bits, corresponde a porta por onde é fornecido o endereço da posição de memória a ser acessada.
- » Instruções: Com 32 bits, corresponde a porta por onde é lida a palavra binária que representa a instrução presente na posição de memória acessada.

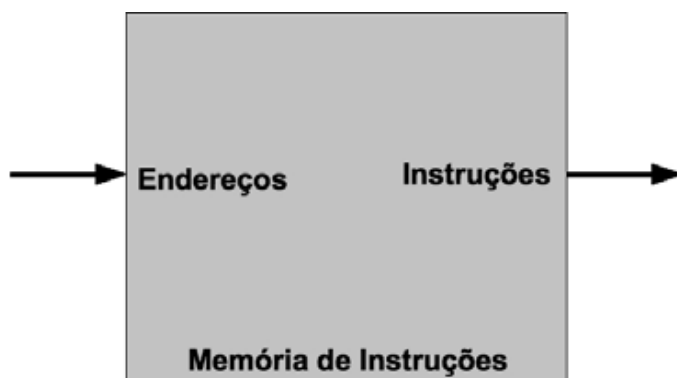


Figura 37 - Diagrama elétrico da memória de instruções

Agora que já conhecemos os símbolos dos principais componentes que formam a arquitetura interna do MIPS, podemos dar uma olhada no seu diagrama completo.

### Diagrama Interno do MIPS

Na figura Figura 38, a seguir, temos o diagrama interno do processador. Apesar de este ser um diagrama simplificado, é suficiente para que possamos ter uma ideia do funcionamento do seu hardware.

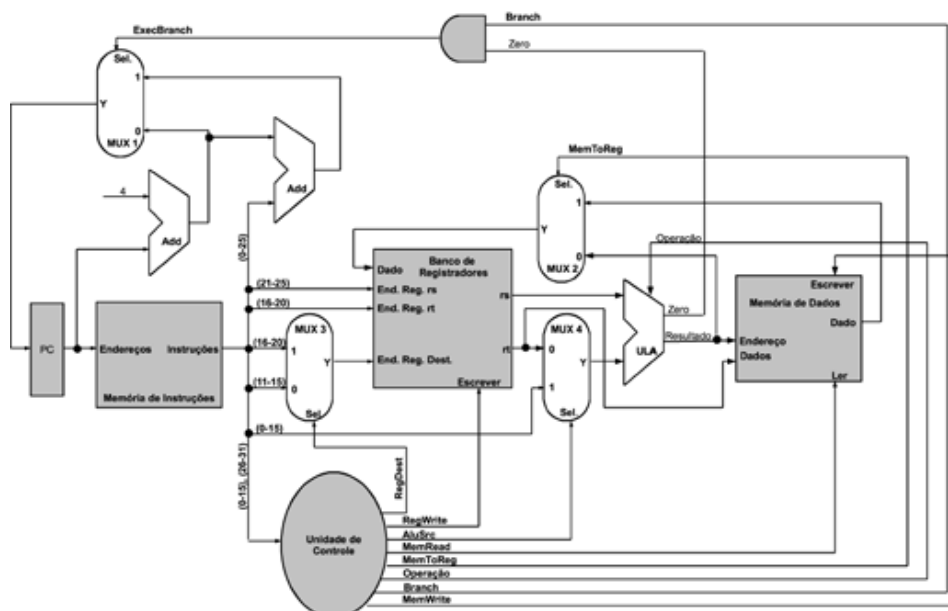


Figura 38 – Diagrama interno simplificado do processador MIPS

Para facilitar a compreensão do seu funcionamento, vamos analisar a execução de algumas instruções diretamente neste hardware, verificando os valores atribuídos a cada um dos sinais internos do processador para cada uma destas instruções, e suas consequências

no sistema como um todo.

Vamos começar analisando uma instrução aritmética do tipo R. Vamos analisar a instrução **Add R3, R4, R4** a qual já teve o seu formato interno e a sua representação na linguagem de máquina do MIPS apresentado na Figura 28. Acompanhe pela Figura 39, a seguir, a execução desta instrução.

1. O conteúdo do PC é aplicado à memória de instruções a fim de ler a instrução a ser executada.
2. A memória de instruções fornece a instrução armazenada no endereço indicado. O conteúdo da instrução é dividido em palavras binárias menores as quais são simultaneamente aplicadas à unidade de controle para decodificação da instrução e ao banco de registradores para ler os possíveis operandos. Ao mesmo tempo o conteúdo do PC que foi aplicado à memória de instruções é somado a 4 para já preparar para a carga da próxima instrução.
3. Ao mesmo tempo que o banco de registradores fornece o conteúdo dos registradores endereçados, a unidade de controle decodifica a instrução, identificando que é uma instrução Add, e programa todos multiplexadores e a ULA para a execução correta da instrução. Observe na figura como os multiplexadores ficaram configurados. Veja que por se tratar de uma instrução tipo R o sinal AluSrc foi configurado com nível lógico 0, o que permitiu que o conteúdo dos registradores rs e rt fossem conectados à ULA para serem operados.
4. A ULA executa a instrução programada pela unidade de controle.
5. O resultado obtido é escrito no registrador de destino.

Observe que, por não se tratar de uma instrução de desvio, o sinal Branch foi configurado com nível lógico 0, o que, independente do resultado obtido no sinal Zero, força o sinal ExecBranch para 0. Como consequência o Mux 1 fica configurado para carregar o PC com o resultado de PC + 4.

Bem, temos que admitir, os engenheiros que projetaram o MIPS fizeram mesmo um bom trabalho, não é mesmo?

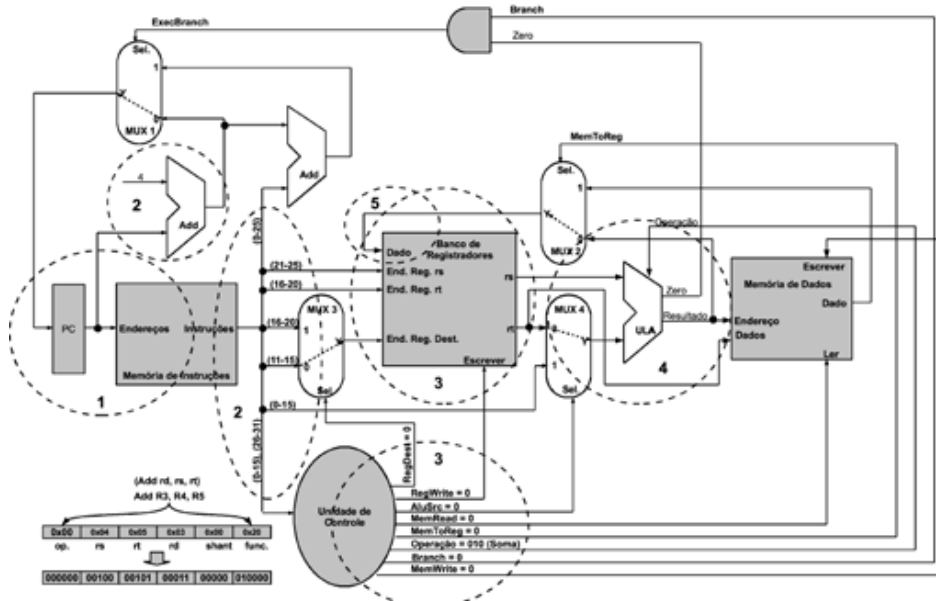


Figura 39 - Execução da instrução Add no Hardware do MIPS

Vamos agora ver outra instrução, desta vez uma instrução tipo I. Só para não ficar muito diferente do que acabamos de ver, vamos analisar a instrução `Addi R3, R4, 0x22`. Acompanhe pela Figura 40, a seguir, a execução desta instrução no hardware do MIPS.

Como você pode observar, a execução das duas instruções são praticamente idênticas, com uma única diferença, a programação do sinal `AluSrc` que passa de 0 para 1. Com isto o Mux 4 passa a conectar à entrada da ULA o conteúdo do campo imediato da instrução, bits 0 à 15, para serem utilizados como segundo operando.

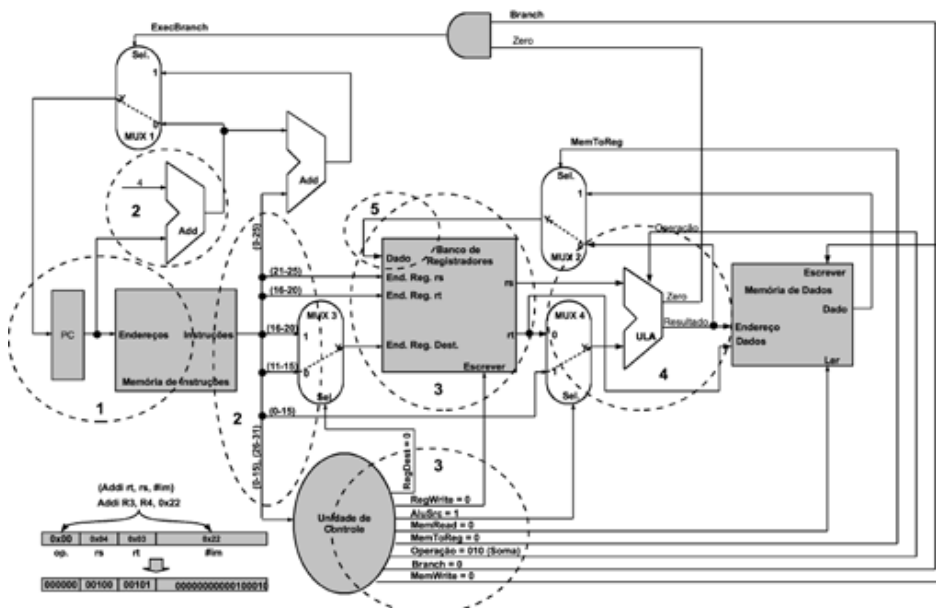


Figura 40 - Execução da instrução Addi no Hardware do MIPS

Muito interessante, não é mesmo?

Vamos agora pegar uma instrução que pode parecer um pouco mais complicada mas que, como você poderá constatar, é também extremamente simples. Vamos analisar a instrução **Lw R8, 3(R5)**. Acompanhe pela Figura 41, a seguir, o passo a passo da execução desta instrução no hardware do MIPS.

1. O conteúdo do PC é aplicado à memória de instruções a fim de ler a instrução a ser executada.
2. A memória de instruções fornece a instrução armazenada no endereço indicado. O conteúdo da instrução é dividido em palavras binárias menores as quais são simultaneamente aplicadas à unidade de controle para decodificação da instrução e ao banco de registradores para ler os possíveis operandos. Ao mesmo tempo o conteúdo do PC que foi aplicado à memória de instruções é somado a 4 para já preparar para a carga da próxima instrução.
3. Ao mesmo tempo que o banco de registradores fornece o conteúdo dos registradores endereçados, a unidade de controle decodifica a instrução, identificando que é uma instrução Lw, e programa todos multiplexadores e a ULA para a execução correta da instrução. Observe na figura como os multiplexadores ficaram configurados. Veja que por se tratar de uma instrução tipo I o sinal AluSrc foi configurado com nível lógico 1, o que ajusta para que sejam operados o registrador rs e o campo imediato da instrução para serem operandos pela ULA. Um ponto interessante e sutil a ser observado é que a ULA deve ser configurada para uma operação de soma (010), uma vez que para calcular o endereço de memória a ser acessado é necessário somar o conteúdo do registrador rs com o valor do campo imediato.
4. A ULA executa a instrução programada pela unidade de controle. Só que, diferente do que aconteceu nas instruções Add e Addi, o resultado da operação da ULA em vez de ser enviado para ser armazenado em algum registrador, é aplicado à porta de endereços da Memória de Dados.
5. Por fim, o dado que foi lido da memória é enviado para ser escrito no registrador indicado no campo rt através do MUX 2.



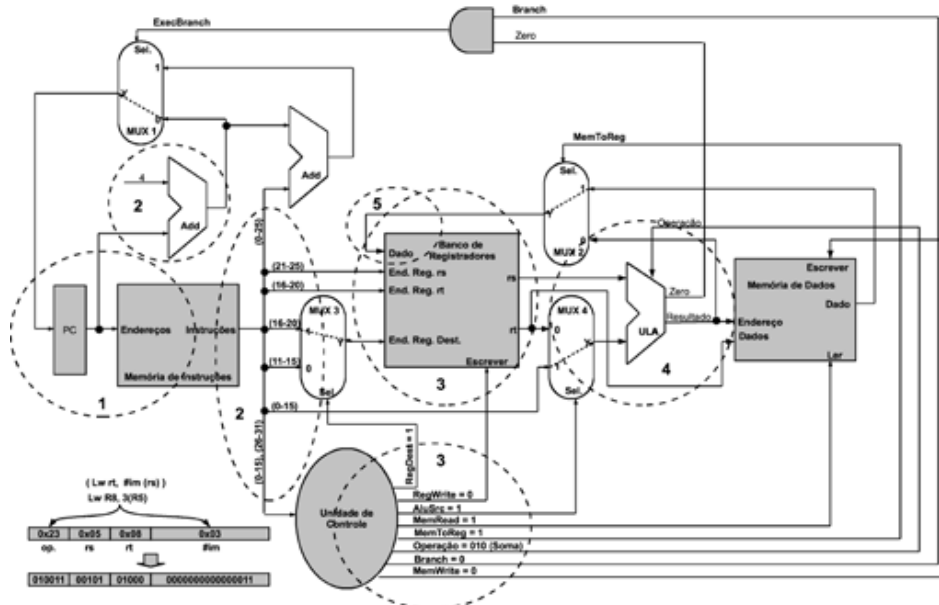


Figura 41 - Execução da instrução Lw no Hardware do Mips

Para concluir, vamos dar uma olhada em como ficaria a execução de uma instrução de desvio condicional. Vamos ver a execução da instrução **Beq R9, R2, 0xA3**. Acompanhe pela Figura 42, a seguir, como fica a execução desta instrução.

1. O conteúdo do PC é aplicado à memória de instruções a fim de ler a instrução a ser executada.
2. A memória de instruções fornece a instrução armazenada no endereço indicado. O conteúdo da instrução é dividido em palavras binárias menores as quais são simultaneamente aplicadas à unidade de controle para decodificação da instrução e ao banco de registradores para ler os possíveis operandos. Ao mesmo tempo o conteúdo do PC que foi aplicado à memória de instruções é somado a 4 para já preparar para a carga da próxima instrução.
3. Ao mesmo tempo que o banco de registradores fornece o conteúdo dos registradores endereçados, a unidade de controle decodifica a instrução, identificando que é uma instrução **Beq**, e programa todos multiplexadores e a ULA para a execução correta da instrução. Observe na figura como os multiplexadores ficaram configurados. Veja que apesar de se tratar de uma instrução tipo I o sinal **AluSrc** foi configurado com nível lógico 0, o que define que os registradores `rs` e `rt` serão operandos pela ULA. Lembre-se, a execução desta instrução exige que

sejam comparados os conteúdos dos registradores rs e rt para definir se ocorrerá ou não o desvio programado. Observe pela Tabela 7 que a ULA não possui a operação de comparação. A solução encontrada foi utilizar a operação de subtração para fazer a comparação do conteúdo dos dois registradores.

4. A ULA executa a instrução programada pela unidade de controle, no caso uma subtração. Caso os dois registradores tenham um mesmo conteúdo, a operação resultará em zero, o que é sinalizado pela ULA levando o nível lógico do sinal Zero, à sua saída, para 1. Caso contrário, caso o conteúdo dos registradores sejam diferentes entre si, o sinal Zero ficará com nível lógico 0.
5. Por fim, caso o sinal Zero seja 1, o sinal ExecBranch também terá seu nível lógico alterado para 1, uma vez que este é obtido a partir da função lógica and entre os sinais Zero e Branch. Isto configurará o MUX 1 para carregar o PC com o resultado da soma de PC+4 mais o conteúdo do campo imediato, o que configurará um desvio no programa. Caso contrário, caso o sinal Zero seja 0, sinal ExecBranch também terá o seu nível lógico igual a 0, o que configurará o MUX 1 para carregar o PC com PC+4 como de costume.

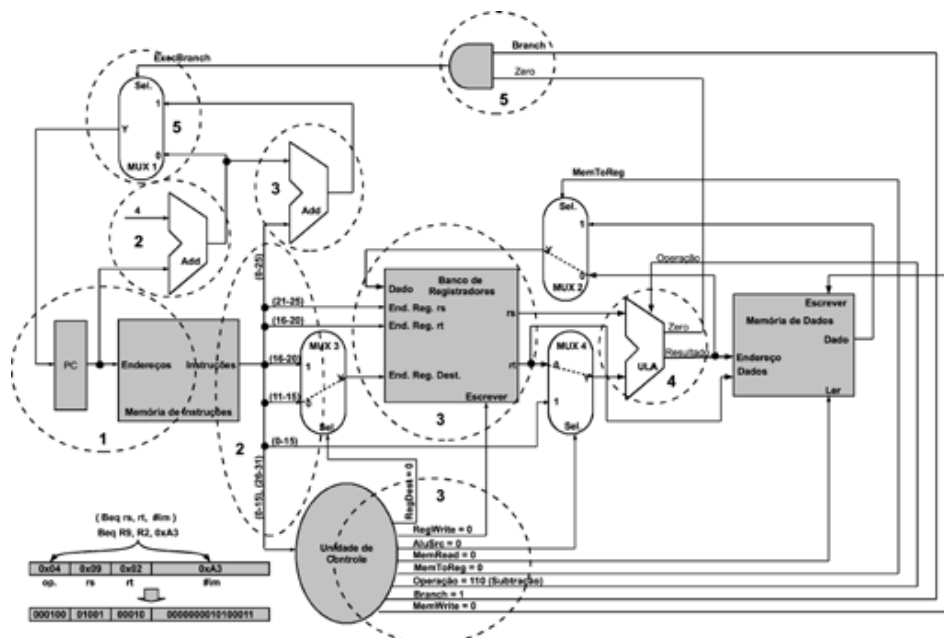


Figura 42 - Execução da instrução Lw no Hardware do Mips

Bem, com isto terminamos nosso estudo pela hardware do MIPS.

Existem muitos outros pontos a serem explorados com relação ao hardware deste processador, os quais estão disponíveis na literatura indicada em nossa bibliografia.



### **Vamos Revisar?**

Neste capítulo, nós conhecemos as principais classes de processadores e o que eles têm em comum com respeito a sua arquitetura interna.

Conhecemos também o Caminho de Dados e vimos a sua importância para o desempenho do processador.

Aprendemos o fluxo de execução de uma instrução no processador, com as várias fases em que se dividem os seus ciclos de busca e de execução.

Neste capítulo, aprendemos também sobre a linguagem de máquina e sobre a linguagem de montagem do MIPS. Vimos como as instruções da linguagem de máquina são decodificadas pela unidade de controle deste processador e como esta unidade de controle utiliza as informações contidas nestas instruções para controlar o hardware deste processador durante a execução destas instruções.



### **Atividades e Orientações de Estudo**

Como você deve ter percebido, o estudo da arquitetura interna de um processador não é algo muito simples.

Desta forma, recomendamos que você continue dedicando algum tempo para resolver os exercícios do seu caderno de exercícios. E, lembre-se você sempre pode contar com a ajuda dos tutores da disciplina.



## Capítulo 3

### O que vamos estudar?

Neste capítulo, vamos estudar os seguintes temas:

- » As interrupções;
- » O ciclo de instrução com o uso de interrupções;

### Metas

Após o estudo deste capítulo, esperamos que você consiga:

- » Aprender os princípios básicos das interrupções;
- » Compreender porque o uso de interrupções possibilita o aumento do desempenho dos sistemas computacionais;
- » Entender os impactos causados pelas interrupções no ciclo de instrução.

## Capítulo 3 – Ciclo de Execução e Interrupções



**Vamos conversar sobre o assunto?**

### 3.1 Introdução

Durante a execução dos programas, muitos computadores precisam otimizar o desempenho dando oportunidade aos programas de tratar eventos inesperados como falhas em dispositivos, tratar referência a um endereço de memória fora do espaço de endereçamento do programa, dentre outras ocorrências.

Imagine ainda que se o processador puder tratar outras ações enquanto espera uma resposta de um periférico, interrompendo assim o seu fluxo sequencial de trabalho, o seu desempenho poderia ser melhorado. É seguindo essa ideia que surgem os mecanismos de tratamentos de interrupções.

Mas o que vem a ser uma interrupção dentro do contexto computacional? Podemos dizer que uma interrupção, como o próprio nome já sugere, é uma parada na execução sequencial de um programa do usuário para tratar um evento assíncrono que pode ser causado por diversos fatores.

Imagine, por exemplo, uma situação onde um programa que executa no processador requisita leitura de dados de um arquivo que está em disco. Normalmente, as CPUs trabalham em velocidades muito superiores aos dispositivos periféricos. Mesmo com toda a evolução tecnológica, há uma tendência que esses dispositivos continuem mais lentos que a própria CPU, pois esta última é o cérebro do computador, devendo, portanto atender a vários dispositivos periféricos.

Durante a leitura do disco, o programa em execução na CPU ficará em estado de espera, aguardando o término na leitura dos dados. Caso esse tempo de espera seja aproveitado com a execução de outras atividades que não dependessem do resultado da leitura do arquivo, haveria um ganho em termos de desempenho de processamento. Se você entendeu essa situação, você entendeu o propósito do uso das interrupções como alternativa para melhorar o desempenho dos

processadores na execução dos programas.

Para exemplificar, podemos pensar em um cenário onde um programa executa numa CPU que não utiliza interrupções. Esse programa possui uma operação de leitura em disco e durante a execução do comando de leitura do periférico, o controle da operação é desviado para a controladora de disco. Se você não sabe o que é a controladora do disco, não se preocupe, pois veremos em volumes subsequentes.

O programa do nosso exemplo segue a sua execução e ao encontrar um comando de leitura de disco, desvia a operação de leitura para a controladora. Durante a operação de leitura há um tempo ocioso da CPU, pois ela continua parada, aguardando o retorno da operação de leitura por parte da controladora. Quando a controladora concluiu a operação de leitura, o fluxo do programa retorna novamente à CPU que seguirá com a execução dos demais comandos. Observe a Figura 1 abaixo representando o esquema de execução do referido programa.

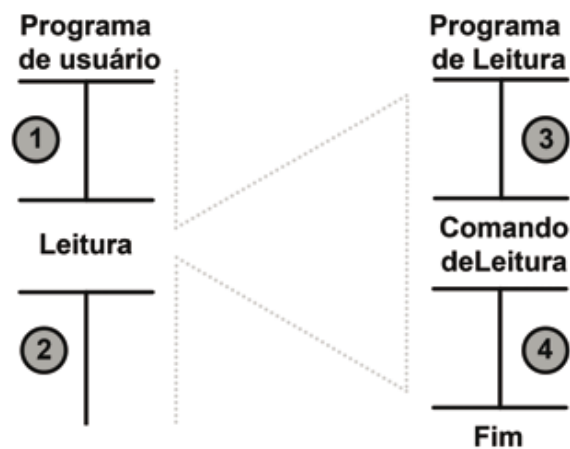


Figura 1 – Programa executando sem interrupção.

Durante a operação de leitura haverá um intervalo de tempo (T) no qual a CPU não poderá executar ações deste programa, devido o aguardo do retorno da operação de leitura. Isso não é desejável, pois o desempenho da CPU fica bastante degradado. Sendo assim, uma maneira de otimizar esse desempenho é introduzir o uso de interrupções.

Ao ser executado, o programa da Figura 1 seguirá a seguinte sequência de execução:

1, 3, T, 4, 2

Onde T = Tempo de leitura do disco

Se pensarmos que para execução dos blocos de comandos 1, 3, 4, 2 sejam decorridos 1 milissegundo para a execução de cada um, o tempo total para a execução do programa seria 4 milissegundo + T, onde T é um tempo ocioso para a CPU.

Suponha agora esse mesmo programa sendo executado numa CPU que trabalha com interrupções. Neste caso ao encontrar um comando de leitura, a CPU desvia o controle para a controladora, mas enquanto aguarda pelo retorno da operação, ela poderá executar outras instruções daquele mesmo programa quando estas não dependem diretamente da saída da operação de leitura. O mesmo esquema de execução apresentado anteriormente comporta-se conforme a Figura 2, no caso de uso de interrupções.

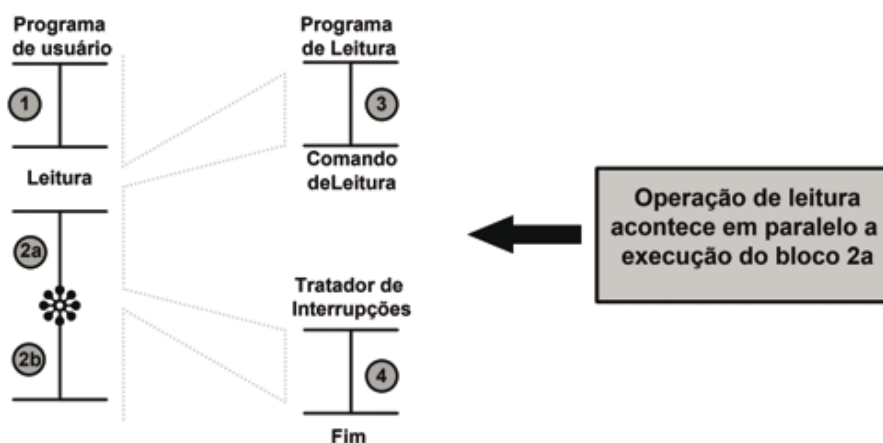


Figura 2 – Programa executando com interrupção.

Note agora que a operação de leitura acontece em paralelo ao primeiro conjunto de comandos do bloco 2. Quando a operação de leitura termina, a controladora irá colocar um sinal no barramento, avisando que a operação acabou.

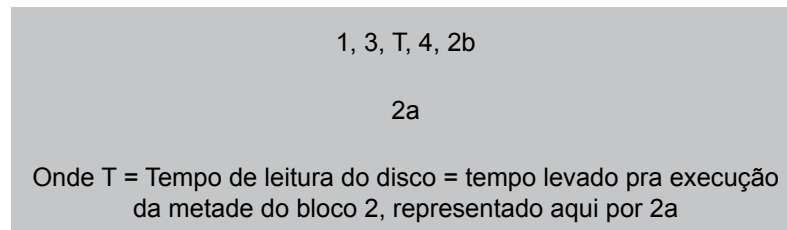
Neste ponto, a CPU salva o contexto de execução do programa (para saber onde o programa parou) e vai verificar o retorno da operação de leitura (**Tratador de Interrupções**). Após o tratamento da interrupção, o fluxo de execução continua na instrução imediatamente após aquela do momento da interrupção, executando assim, o bloco de instruções representado por 2b.



#### Saiba Mais

*Tratador de interrupções é o programa do sistema operacional que identifica qual o tipo de interrupção que aconteceu.*

Ao ser executado, o programa da Figura 2 seguirá a seguinte sequência de execução:



Dessa forma, observamos que o tempo T é utilizado para a execução de uma outra tarefa, não sendo portanto, um tempo ocioso já que a CPU não fica desperdiçando tempo, esperando um retorno do dispositivo.

Note que quando o dispositivo externo estiver pronto para ser utilizado, a controladora enviará um sinal de requisição de interrupção à CPU. Em resposta a esse sinal, a CPU suspende a execução do programa atual, desviando o controle para uma rotina que controla a operação do disco, apenas retomando a execução do original, após os dados terem sido obtidos do disco.

Diante desse exemplo que acabamos de apresentar, você percebe qual o benefício que as interrupções podem introduzir nos ambientes computacionais?

Bem, embora o uso de interrupções aumente o desempenho dos sistemas computacionais, para que o processador passe a utilizá-las, alguns controles de fluxos precisam ser incluídos, conforme observamos no exemplo acima. Por exemplo, após tratar a operação de leitura, o processador precisa saber a partir de qual instrução do programa, ele precisará continuar.

Sendo assim, um ciclo de interrupção precisará ser incluído no ciclo de instrução original, levando este ciclo ao esquema apresentado na Figura 3.



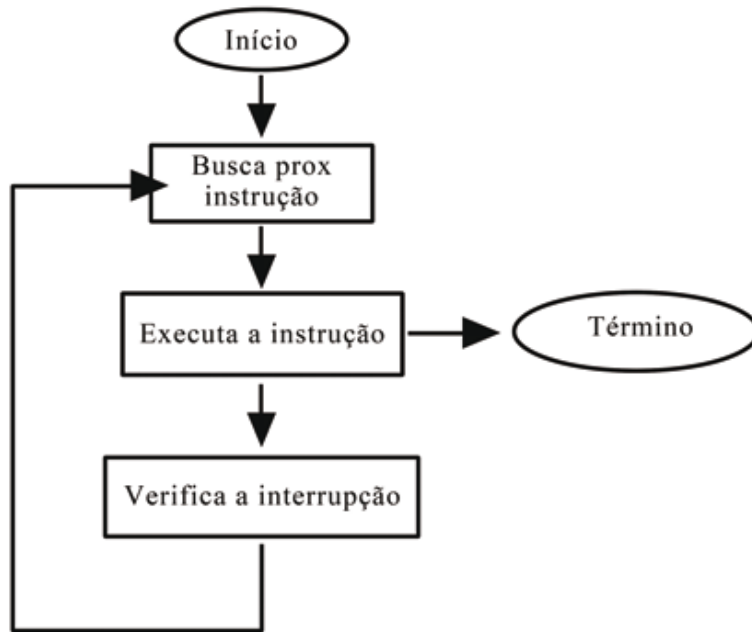


Figura 3 – Ciclo de execução com Interrupções

A Figura 3 nos mostra que o processador busca a próxima instrução, a executa e em seguida, verifica se ocorreram interrupções. Caso não haja interrupção pendente, o processador prossegue buscando a instrução seguinte do programa corrente.

Nos casos onde há interrupção pendente, o processador realiza as seguintes tarefas:

- » Suspende a execução do programa corrente e salva o endereço da próxima instrução a ser executada;
- » Configura o registrador PC para iniciar a rotina de interrupção;
- » O processador procede, buscando a primeira instrução no programa tratador de interrupção.

Você pode estar pensando que a introdução de interrupções aumenta a sobrecarga de processamento, já que instruções extras devem ser executadas para determinar a natureza da interrupção para então decidir a medida a ser adotada. Se você pensou assim, você está certo. Por outro lado, do ponto de vista da CPU, há um grande intervalo de tempo na espera de uma operação de entrada e saída, levando o processador a obter mais eficiência com o uso de interrupções.



#### Saiba Mais

O PC (Program Counter), é um registrador cujo objetivo é armazenar o endereço da próxima instrução.

### 3.2 Múltiplas Interrupções

Observe que até o momento estamos trabalhando na ocorrência de uma única interrupção. Você já parou para pensar no que acontece quando uma nova interrupção é levantada durante o tratamento de outra?

Para ficar mais fácil de entender, imagine um programa que recebe dados por uma placa de rede e envia para escrita em disco. A controladora do disco gera uma interrupção toda vez que completa uma operação de escrita. A controladora da placa de rede, por sua vez, gera uma interrupção toda vez que recebe um pacote de dados.

Para resolver esses problemas, podem ser utilizadas duas soluções: A solução mais simples sugere que durante o tratamento de uma interrupção ocorra a desabilitação de demais interrupções. Dessa forma, enquanto as interrupções estiverem desabilitadas, o processador ignora qualquer sinal de requisição de interrupção. As interrupções ocorridas durante este intervalo ficarão pendentes e serão verificadas pela CPU apenas após a habilitação das interrupções. Quando o tratador de interrupções termina de manipular a primeira interrupção, as interrupções são novamente habilitadas, possibilitando ao processador verificar a ocorrência de novas interrupções.

Essa solução lhe parece simples? Ela de fato é simples, mas não leva em consideração prioridades dos dispositivos. Como sabemos, alguns dispositivos periféricos são mais rápidos que outros. Facilmente observamos que uma impressora é um dispositivo lento quando comparada a uma placa de rede ou teclado. Essa solução tratará todos os dispositivos e as interrupções geradas pelos mesmos com igual prioridade e essa é a sua desvantagem, já que não resolve o problema de diferentes taxas de transmissão de dispositivos.

Outra solução é atribuir e considerar prioridades de interrupções. Dessa forma, se uma interrupção está sendo tratada e uma outra ocorre durante a sua manipulação, caso a prioridade da segunda supere a da primeira, a primeira interrupção será suspensa temporariamente para tratar a interrupção de maior prioridade. Terminado o processamento da segunda interrupção, o processador retorna ao tratamento da anterior. Caso contrário, se a segunda tiver prioridade menor que a primeira, ela ficará aguardando a conclusão da mesma.



## Aprenda Praticando

Agora que você já leu a teoria e os exemplos apresentados, vamos tentar entender os exercícios resolvidos a seguir. É interessante que antes de observar a solução você tente sozinho resolver, para em seguida comparar a sua solução com a proposta pelo livro.

### Exercício 1

Desconsiderando o uso de interrupções, esquematize através de um cenário a execução de um programa que possui 2 operações envolvendo E/S.

### Exercício 2

Considere o programa sendo executado em uma máquina com interrupções. Compare os dois programas, identificando os trechos de execução que causam perda de desempenho.

### Exercício 3

Considere uma versão com interrupções que demoram a ser disparadas (situação típica de dispositivos lentos). Que situação nova deve ser considerada no cenário?

### Exercício 4

De acordo com estes cenários, o que podemos esperar, em geral, com a introdução de interrupções.

Inicialmente você deve desconsiderar o uso de interrupções, e esquematizar através de um cenário a execução de um programa que possui 2 operações envolvendo entrada e saída. Lembre-se que operações de entrada e saída podem ser leitura ou escrita de dados, por exemplo. Você pode utilizar o mesmo raciocínio explicado na seção 3.1 para o programa que continha uma única operação de leitura em disco. Lembra?

Bom, então de posse de caneta e papel, vamos desenhar o esquema de execução de nosso programa, resolvendo, portanto o **Exercício 1**. Representaremos os blocos de comandos do programa do usuário por 1, 2, e 3 e os blocos de comandos do programa de

entrada e saída pelos trechos 4 e 5. Veja o esquema na Figura 4.

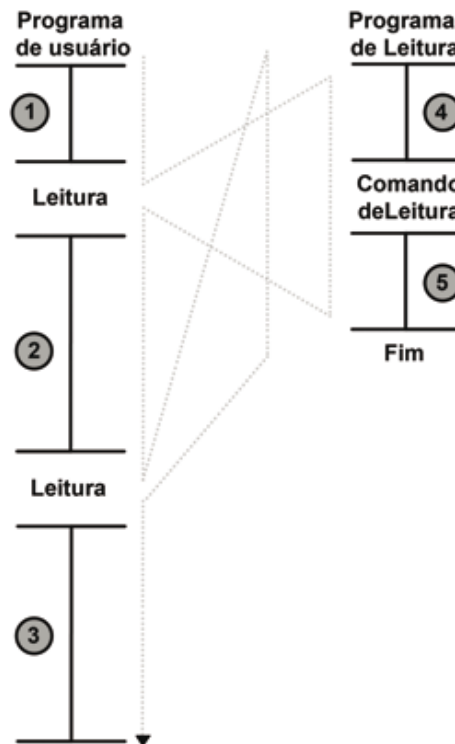


Figura 4 – Programa com 2 operações de E/S

O programa do nosso exemplo segue a sua execução e ao encontrar um comando de E/S, desvia a operação de leitura para a controladora. Durante a operação de leitura há um tempo ocioso da CPU, pois ela continua parada, aguardando o retorno da operação de E/S por parte da controladora. Quando a controladora concluiu a operação de E/S, o fluxo do programa retorna novamente à CPU que seguirá com a execução dos demais comandos.

Assim como no programa representando na seção anterior, durante a operação de E/S haverá um intervalo de tempo (T) no qual a CPU não poderá executar ações deste programa, devido o aguardo do retorno da operação de E/S. Sendo assim, concluímos que a execução do programa seguirá a seguinte sequência de execução:

1, 4, T, 5, 2, 4, T, 5, 3

Onde T = Tempo de leitura do disco

Para dar continuidade ao nosso exercício, vamos então considerar agora que este mesmo programa esteja sendo executado em uma CPU que faça uso de interrupções. Esse é o questionamento do

## Exercício 2.

Ao encontrar um comando de entrada e saída, a CPU desvia o controle para a controladora, mas enquanto aguarda pelo retorno da operação, ela poderá executar outras instruções daquele mesmo programa.

Note agora que a operação de entrada e saída acontece em paralelo ao primeiro conjunto de comandos do bloco 2. Quando a operação de leitura termina, a controladora irá colocar um sinal no barramento, avisando que a operação acabou.

Neste ponto, a CPU salva o contexto de execução do programa (para saber onde o programa parou) e vai verificar o retorno da operação de entrada e saída (Tratador de Interrupções). Após o tratamento da interrupção, o fluxo de execução continua na instrução imediatamente após aquela do momento da interrupção, executando assim, o bloco de instruções representado por 2b.

Em seguida, o programa encontra a segunda operação de entrada e saída, e novamente a CPU desvia o controle para a controladora, e enquanto aguarda pelo retorno da operação, ela executa outras instruções daquele mesmo programa. Vamos observar o esquema de execução na Figura 5.

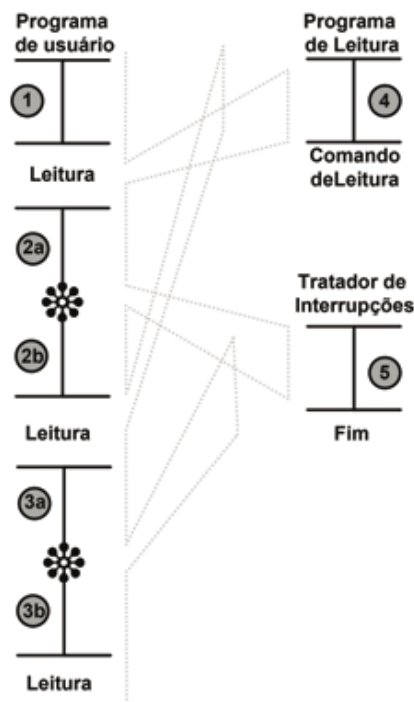
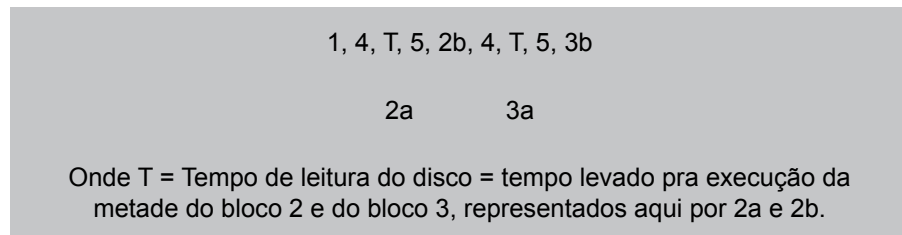


Figura 5 – Programa em Execução com 2 operações de E/S

O programa da Figura 5 seguirá a seguinte sequência de execução:



Vamos agora comparar o programa executando nas duas situações (**Exercício 1 e Exercício 2**). Vamos identificar quais os trechos de execução que causam perda de desempenho.

Observamos que o tempo T é utilizado para a execução de outra tarefa, não sendo portanto, um tempo ocioso já que a CPU não fica desperdiçando tempo esperando um retorno do dispositivo. Quando o dispositivo externo estiver pronto para ser utilizado, a controladora enviará um sinal de requisição de interrupção à CPU. Em resposta a esse sinal, a CPU suspende a execução do programa atual, desviando o controle para uma rotina que controla a operação do disco, apenas retomando a execução do original, após os dados terem sido enviados ao disco.

Dessa forma, a situação ilustrada no **Exercício 2** apresenta um **tempo total** de execução menor, demonstrando que o desempenho da execução do mesmo programa no **Exercício 2** foi melhor que no **Exercício 1**, já que não houve tempo ocioso.

Agora vamos considerar uma versão com interrupções que demoram a ser disparadas (situação típica de dispositivos lentos, por exemplo impressoras). Que situação nova deve ser considerada no cenário? Esta é a pergunta do **Exercício 3**.

Você sabe que alguns periféricos são mais lentos que outros, não é? Uma impressora ou um *scanner*, por exemplo, são dispositivos lentos. Neste caso, para o nosso cenário, vamos considerar que o tempo levado para um dispositivo lento executar a operação de E/S igual ao tempo levado para executar um bloco de comandos completo (representado por 2 e 3) da Figura 6.

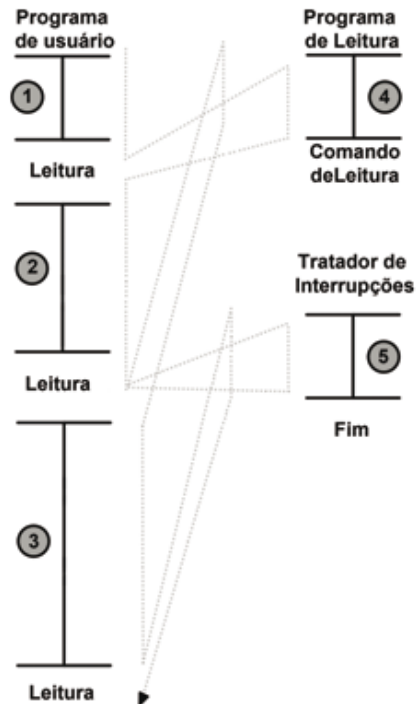


Figura 6 – Programa em Execução com 2 operações de E/S

Dessa forma, a sequência de execução seguirá o fluxo exposto abaixo, respondendo, portanto, o **Exercício 3**:

1, 4, T, 5, 4, T, 5

2      3

Onde T = Tempo de leitura do disco = tempo levado pra execução do bloco 2 e do bloco 3.

Por fim, de acordo com estes cenários, o que podemos esperar, em geral, com a introdução de interrupções? Essa pergunta está no **Exercício 4** e você sem dúvida já está habilitado a responder. Veja que a introdução de interrupções agrega mais complexidade à execução dos programas, entretanto, os ganhos de desempenho são notáveis, pois todo tempo de espera em operações de entrada e saída são utilizados para a execução de outras tarefas.



### Atividades e Orientações de Estudo

Dedique, pelo menos, 2 horas de estudo para o Capítulo 3.

Organize uma metodologia de estudo que inicie com a leitura dos conceitos e acompanhamento dos exercícios resolvidos.

Você poderá esclarecer suas dúvidas com o professor e os tutores utilizando os chats e os fóruns tira-dúvidas no ambiente virtual de seu curso.

Não esqueça de ler atentamente o guia de estudo da disciplina, pois nele você encontrará a divisão de conteúdo semanal, ajudando-o a dividir e administrar o seu tempo de estudo.

Observe os prazos estabelecidos pelo seu professor para essas atividades virtuais. Lembre-se que as atividades somativas propostas pelo professor no ambiente virtual são importantes para o aprendizado e para a composição da sua nota.



### **Vamos Revisar?**

Neste capítulo, você estudou as interrupções e o seu impacto na execução dos programas.

Aprendemos que as interrupções acrescentam complexidade à execução embora permitam que os tempos de espera por retorno das operações de entrada e saída (executadas por periféricos) sejam aproveitadas para a execução de outras tarefas, possibilitando, assim, uma melhoria de desempenho.

Estudamos as técnicas existentes para o tratamento de múltiplas interrupções e quais os impactos sofridos pelo ciclo de busca e execução com a inserção de interrupções.

No próximo e último capítulo deste volume, você estudará os barramentos ou estruturas de interconexões.





## Capítulo 4

### O que vamos estudar?

Neste capítulo, vamos estudar os seguintes temas:

- » Princípios básicos de funcionamento das estruturas de interconexão;
- » Principais padrões comerciais de barramentos;
- » Evolução histórica dos padrões comerciais de barramentos;

### Metas

Após o estudo deste capítulo, esperamos que você consiga:

- » Aprender os princípios básicos de funcionamento dos barramentos;
- » Compreender a evolução dos padrões de barramento sob o ponto de vista tecnológico;
- » Conhecer um pouco dos padrões comerciais de barramentos dos últimos 20 anos até a atualidade.

## Capítulo 4 – Estruturas de Interconexão



### Vamos conversar sobre o assunto?

Como conversamos anteriormente, o computador é uma rede constituída de módulos básicos: dispositivos de E/S (entrada e saída), memória e CPU. Tais módulos precisam se conectar, para que possam trocar dados e sinais de controle, com o objetivo de executar programas. Então, como seria possível conectá-los?



Podem ser utilizados para isso, um conjunto de caminhos conectando os diversos módulos. Os módulos podem transferir entre si diversos tipos de informações. Vejamos alguns exemplos:

- » Memória → CPU
  - A memória transfere dados para a CPU, por exemplo, no momento da edição de um documento;
- » E/S → CPU
  - Um dispositivo de E/S transfere dados para a CPU quando, por exemplo, um documento está sendo editado, via teclado ou mouse;
- » CPU → Memória
  - A CPU transfere dados para memória como resultado da edição do documento (resultado do processamento);

## » CPU → E/S

- A CPU transfere dados para um dispositivo de E/S, quando, por exemplo, um documento está sendo visualizado no monitor;

## » Memória → E/S

- O ato de salvar um documento no disco rígido é um exemplo de transferência de dados da memória para E/S.

## » E/S → Memória

- A abertura de um documento representa uma retirada do arquivo do disco rígido para colocá-lo na memória. O disco rígido representa o dispositivo de E/S.

Sendo assim, uma estrutura ou barramento de interconexão pode ser definido como um caminho conectando dois ou mais dispositivos. Esse caminho é um meio de transmissão compartilhado, onde diversos dispositivos estão conectados e um sinal transmitido por qualquer um deles é recebido por todos os outros conectados ao barramento. Para que a transmissão de dados ocorra com sucesso apenas um dispositivo, por vez, poderá transmitir. Podemos observar uma imagem genérica desse meio compartilhado de transmissão de dados na Figura 1. Nesta ilustração, representamos todos os dispositivos periféricos por E/S. Representamos as estruturas de memórias na caixa denominada Memória, enquanto que a caixa denominada CPU representa a unidade de processamento de dados.

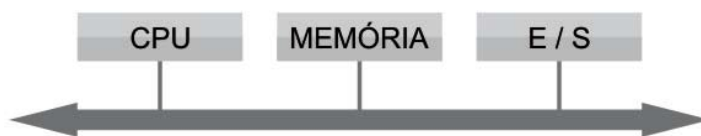


Figura 1 – Barramento

## 4.1 Fundamentos Básicos

Um barramento consiste em vários caminhos ou linhas de comunicação. Cada linha transmite sinais que representam um único dígito binário (0 ou 1). Em geral, as linhas do barramento são agrupadas de acordo com o tipo de informação que elas transportam. Ao longo do tempo, uma sequência de dígitos pode ser transmitida por meio de uma linha. As diversas linhas podem ser usadas em conjunto

para transmitir vários dígitos binários simultaneamente (*em paralelo*). Por exemplo, uma unidade de dados de 8 bits, pode ser transmitida por 8 linhas do barramento.

As linhas do barramento podem ser classificadas em três grupos funcionais: dados, endereço e controle. A distribuição dessas linhas pode ser observada na Figura 2.

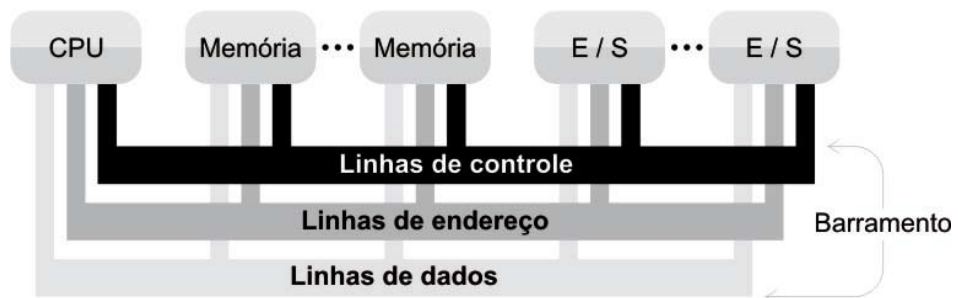


Figura 2 – Linhas do Barramento

As linhas de dados fornecem o meio de transmissão de dados entre os módulos do sistema. O conjunto dessas linhas formam o barramento de dados, que podem conter 8, 16, 32 ou 64 linhas. O número de linhas é conhecido como largura do barramento. Na sua opinião, a largura do barramento interfere de alguma forma no desempenho do sistema?

Suponha uma situação onde o barramento de dados possui largura de 8 bits. Imagine agora que o computador em questão utiliza instruções com 16 bits. Quantas vezes o processador tem que acessar o módulo de memória em cada ciclo de instrução?



Neste caso, o processador teria que realizar dois acessos ao módulo de memória, pois o barramento de dados só suporta transmitir 8 bits por vez. Como a instrução tem 16 bits, os dados são colocados no

barramento agrupados de 8 em 8. Isso significa dizer que quanto mais linhas o barramento de dados possui, ou seja, quanto mais largo ele for, mais bits poderão ser transmitidos simultaneamente, levando-nos a concluir que a largura do barramento tem um impacto direto no desempenho do sistema.

As linhas de endereço são utilizadas para designar fonte e destino dos dados transferidos pelo barramento de dados. Quando um processador deseja ler uma palavra da memória, ele coloca o endereço da palavra desejada nas linhas de endereço do barramento. Como as linhas de dados e de endereços são usadas por todos os componentes, deve existir uma maneira de controlar sua utilização. Esta é uma das funções das linhas de controle do barramento. Elas referem-se ao acesso e o uso das linhas de dados e endereços. Os sinais de controle são utilizados para transmitir comandos, explicitando quais ações serão executadas. São considerados sinais que trafegam no barramento de controle, os seguintes exemplos:

- 1 Escrita na Memória (CPU para memória): Faz com que os dados existentes nas linhas de dados do barramento sejam gravados na posição de memória especificada nas linhas de endereço.
- 2 Leitura de Memória (memória para CPU): Faz com que o valor armazenado no endereço da memória especificado nas linhas de endereço seja colocado nas linhas de dados do barramento.
- 3 Escrita de E/S (CPU para E/S): Faz com que os dados no barramento de dados sejam enviados para uma porta de saída (dispositivo de E/S).
- 4 Leitura de E/S (E/S para CPU): Faz com que os dados em um dispositivo de E/S sejam colocados no barramento de dados.

Como o barramento é um meio de acesso compartilhado, quando um módulo do sistema deseja enviar dados para outro, ele deverá inicialmente obter o controle do barramento através das linhas de controle, já que apenas um componente pode utilizar o meio em um determinado instante. Após a obtenção do controle de acesso, o módulo poderá transferir os dados por meio do barramento.

Da mesma forma, quando um módulo deseja requisitar dados de outro módulo, ele também deverá inicialmente obter o controle do barramento, para então iniciar a transferência da requisição

para o outro módulo, por meio das linhas de endereço e de controle apropriadas. Em seguida, o módulo requisitante deverá esperar que o outro módulo envie os dados requisitados.

Toda essa operação de controle de acesso ao barramento faz parte do protocolo do barramento. O nome que se dá a maneira pela qual os componentes obtêm acesso ao barramento é **Arbitragem**. A arbitragem irá designar um dispositivo para ser o **mestre** na comunicação. O dispositivo mestre é aquele que pode iniciar uma transferência de dados com outros dispositivos, que atuam como escravos nessa atividade específica. Mas o que você acha que acontece se dois ou mais dispositivos quiserem se tornar mestres ao mesmo tempo?

Para resolver esse tipo de impasse, existem dois tipos de técnicas de arbitragem de barramento. São elas: **centralizada e distribuída**.

Na arbitragem centralizada, existe um único dispositivo de hardware, denominado árbitro que comanda a transferência de dados. Ele é responsável por alocar tempo de utilização do barramento a cada módulo do sistema. Já na arbitragem distribuída, não existe um controlador central, ou seja, quando um dispositivo quer usar um barramento, ele ativa sua linha de requisição. A linha de requisição do barramento faz parte do conjunto de linhas de controle.

Para a arbitragem distribuída, quando nenhum dispositivo quer utilizar o barramento, a linha de arbitragem ativada é propagada através de todos os dispositivos. Para obter o acesso ao barramento, um dispositivo primeiro verifica se o mesmo está disponível e se a linha de arbitragem que está recebendo IN está ativada. Se a linha de arbitragem IN estiver desativada, ele não poderá tornar-se mestre do barramento. Se IN estiver ativada, o dispositivo requisita o barramento, desativa OUT, levando todos os dispositivos seguintes na cadeia a desativarem IN e OUT. Esse esquema pode ser observado na Figura 3.

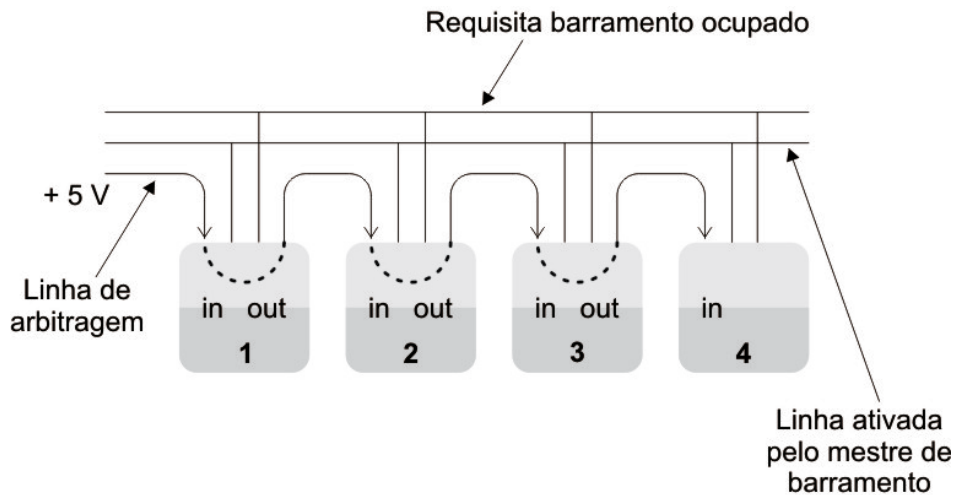


Figura 3 – Arbitragem Distribuída

Baseando-se na arbitragem centralizada, podem ser utilizadas 2 técnicas: **Arbitragem Centralizada com 1 nível** e **Arbitragem Centralizada com 2 níveis**. Na primeira delas (Figura 4), todos os dispositivos são ligados em série, assim a permissão dada pelo árbitro pode ou não se propagar através da cadeia. Cada dispositivo deve solicitar acesso ao barramento, porém o dispositivo mais próximo do árbitro tem maior prioridade. Uma vez que o dispositivo recebe a permissão para acesso ao barramento, ele bloqueia o acesso dos demais.

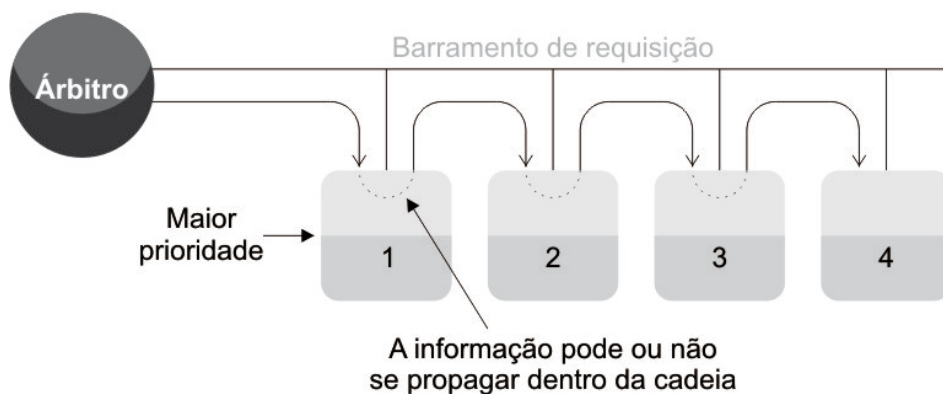


Figura 4 – Arbitragem Centralizada com 1 nível

A arbitragem centralizada com 2 níveis (Figura 5) possui uma pequena variação em relação à técnica anterior, pois existem diferentes níveis de requisição e cada dispositivo se liga a um dos níveis. Os dispositivos com tempos mais críticos se ligam aos níveis de maior prioridade. Se múltiplos níveis de prioridade são requeridos ao mesmo tempo, o árbitro solta a permissão apenas para os de

prioridade mais alta.

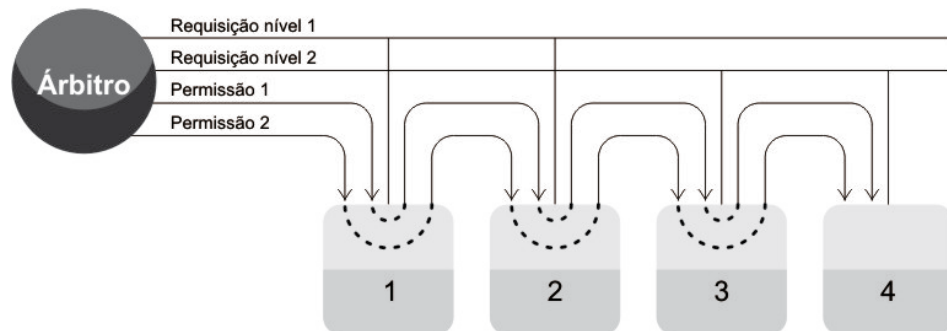


Figura 5 – Arbitragem Centralizada com 2 níveis

## Hierarquia de Barramentos

O uso de barramentos traz consigo diversas vantagens que vão desde o baixo custo na comunicação entre componentes até a facilidade de se adicionar novos dispositivos ao computador. Entretanto, existem desvantagens no uso dos barramentos, no sentido de criação de gargalos na comunicação, pois a vazão máxima no envio de dados aos dispositivos de E/S fica limitada.

Além disso, à medida que aumentamos o número de dispositivos conectados ao barramento, a probabilidade de um dispositivo encontrar o barramento livre para iniciar a transmissão diminui. Dessa forma, quanto mais dispositivos conectados ao barramento, maiores as chances de engarrafamentos e atrasos na transmissão.

Atualmente experimentamos uma tendência de crescimento do número de dispositivos de E/S disponíveis no mercado. Se pararmos para observar o nosso computador, constataremos que temos em média 10 periféricos conectados (monitor, teclado, mouse, impressora, pen drive, cd-room, disco rígido, caixas de som, microfone, webcam). Essa realidade era diferente há 10 anos atrás.

E então, será que existe alguma forma de contornar esse problema? Alternativas devem ser criadas para minimizar os gargalos obtidos devido ao grande número de módulos que compartilham o barramento. Por exemplo, podem ser utilizados barramentos de maior largura, ou seja, mais linhas no barramento de dados, possibilitando a transferência de mais bits por vez.

Essa primeira proposta de aumentar a largura do barramento seria uma solução limitada, pois logo chegaríamos a uma situação de



gargalo. Não sabemos quantos periféricos teremos nos próximos 5 anos!

Sendo assim, outra alternativa baseia-se no uso de múltiplos barramentos dispostos de maneira hierárquica. A Figura 6 apresenta uma hierarquia de barramentos.

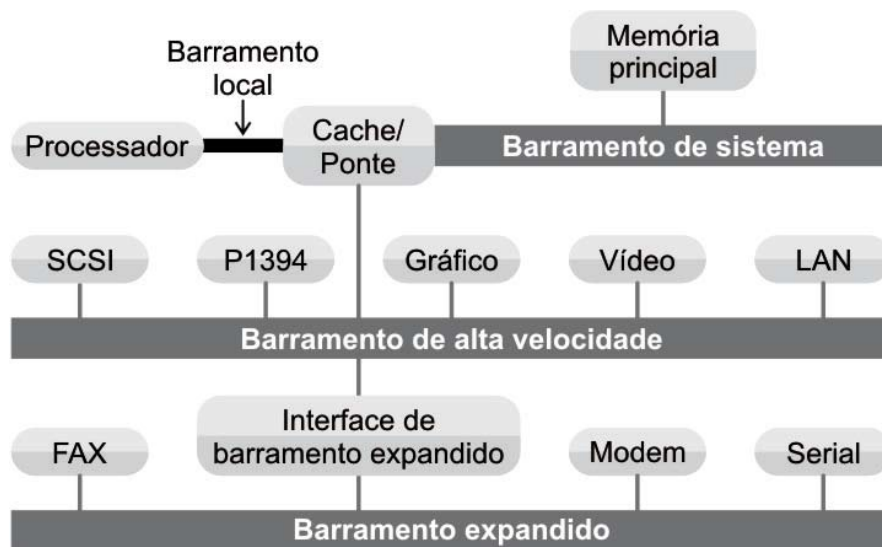


Figura 6 – Hierarquia de Barramento

Na figura 6, observamos 4 níveis da hierarquia do barramento. O barramento que interliga a memória e o processador se dá através de uma ponte na memória cache (vamos estudar esse tipo de memória em volume posteriores) e é denominado barramento local (do inglês, *local bus*). O barramento do sistema (do inglês, *system bus*) conecta a memória principal ao barramento de alta velocidade. Neste último barramento estão conectados os dispositivos periféricos que trabalham em velocidades maiores enquanto que no barramento de expansão estão conectados os periféricos mais lentos, tais como modems e interfaces seriais.

## Temporização de Barramentos

Os barramentos coordenam a ocorrência de eventos de diferentes formas. Esse detalhe de projeto de barramentos refere-se à **Temporização**. A Temporização do barramento indica o modo pelo qual os eventos nesse barramento são coordenados. Os barramentos podem ser classificados como **Síncrono** ou **Assíncrono**, no que se refere à sincronização.

Um barramento síncrono é aquele que exige que todo o tráfego de

dados e controle seja sincronizado sob uma mesma base de tempo chamada relógio (do inglês, *clock*). O barramento síncrono inclui uma linha de relógio, e através dela, um relógio transmite uma sequência alternada de 1s e 0s de igual duração. Uma transmissão de um 1 e de um 0 (denominada ciclo de relógio ou ciclo de barramento) é definida em um intervalo de tempo.

Todos os eventos no barramento devem começar no início de um ciclo de relógio e todos os dispositivos conectados ao barramento síncrono podem ler a linha de relógio. As atividades que ocorrem no barramento gastam um número inteiro destes ciclos. O comportamento temporal deste tipo de barramento é ilustrado na Figura 7.

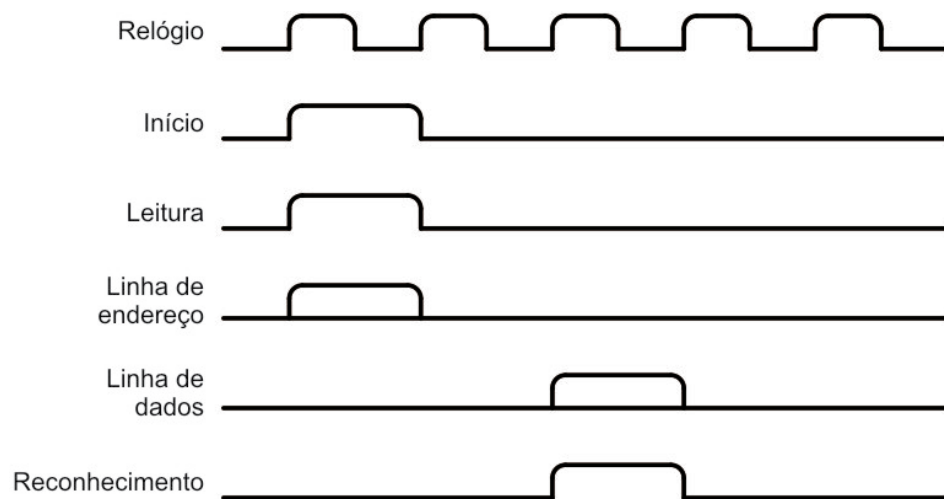


Figura 7 – Barramento Síncrono

Analisaremos as linhas da Figura 7, de cima para baixo. Consideraremos a primeira linha como a mais superior e assim sucessivamente. A primeira linha da Figura 7, indica o comportamento do relógio. A segunda linha indica o início da transferência de dados que são apresentados na quinta linha (linha de dados). O processador emite um sinal de leitura (terceira linha) e coloca um endereço nas linhas de endereço. As informações contidas na quarta linha referem-se ao endereço onde o dado será lido e por fim, a linha de reconhecimento indica que um sinal de controle para confirmar o recebimento foi enviado ao barramento. O processador também emite um sinal de início para marcar a presença do endereço e de informação de controle no barramento. Esse sinal está representado na segunda linha superior da Figura 7.

Diferentemente do barramento síncrono, para um barramento assíncrono não existe um relógio mestre. Os ciclos podem ter

qualquer duração requerida. A ocorrência de um evento no barramento depende de um evento ocorrido anteriormente. O funcionamento desse barramento pode ser observado na Figura 8. Observa-se que o processador coloca os sinais de endereço e de leitura no barramento. Após uma pausa para estabilização desses sinais, ele emite um sinal representado na primeira linha (de cima para baixo) indicando que os sinais apresentados nas linhas de endereço e controle são válidos. Esse sinal é emitido pelo dispositivo mestre. O dispositivo escravo na comunicação responde enviando um sinal na linha SSYN (segunda linha), indicando o envio da resposta e dos dados propriamente ditos, contido nas linhas de dados do barramento (última linha da Figura 8).

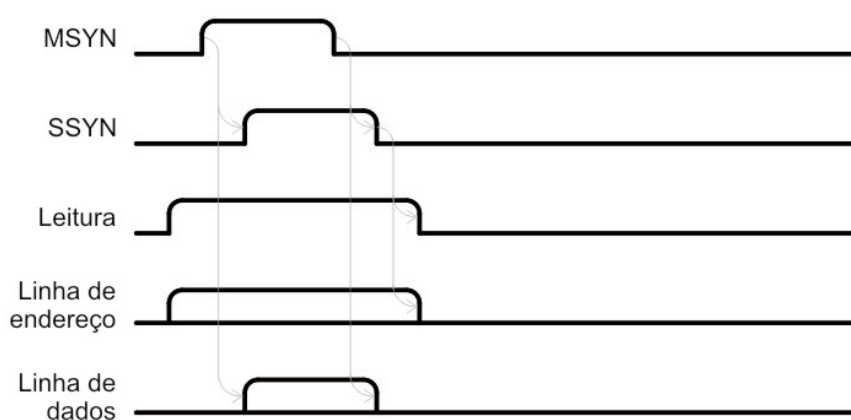


Figura 8 – Barramento Assíncrono

Em geral, os barramentos assíncronos possuem um desempenho superior, pois não perdem tempo com a sincronização, para permitir o início da transferência de dados.

Tanto os dispositivos lentos quanto os rápidos, que utilizam tecnologia mais nova ou mais antiga podem compartilhar o uso do barramento assíncrono, pois não precisam operar segundo a velocidade fixa do relógio. Entretanto os barramentos síncronos são mais facilmente implementados e testados.

## 4.2 Barramentos Comerciais

Agora que você já conhece os princípios básicos de operação e funcionamento dos barramentos, você irá conhecer alguns padrões de barramentos comerciais que foram e que ainda são utilizados nos computadores atuais.

No início dos anos 80, novas categorias de periféricos foram

surgindo, com o desenvolvimento do primeiro computador pessoal (PC-IBM). Sendo assim, foi definido o padrão de barramento ISA (*Industry Standard Architecture*) desenvolvido no início dos anos 80 pela IBM nos laboratórios em Boca Raton, Florida. O ISA surgiu no computador IBM PC (1981), na versão de 8 bits e posteriormente, chegou ao IBM PC-AT (1984), passando a usar 16 bits de dados por vez. A ilustração de uma placa ISA é apresentada na Figura 9.

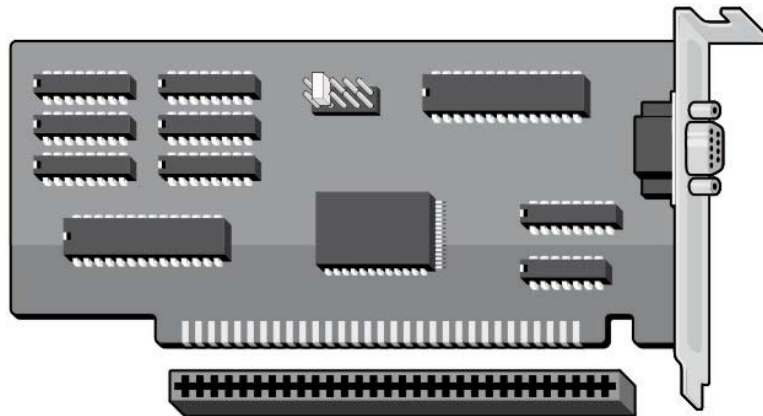


Figura 9 – Estrutura Geral da Placa ISA: 8 bits

No barramento ISA, com 8 bits de dados, a frequência de operação é de 8 MHz, sendo portanto, um barramento síncrono. A sua velocidade atinge cerca de 8 Mbps e foi bastante utilizado em placas de som e fax-modem.

No barramento ISA, com 16 bits de dados (utilizados em processadores 286), a frequência de operação do barramento também era 8 MHz e a velocidade atingia o dobro do ISA 8 bits, chegando a 16 Mbps. A ilustração de uma placa ISA é apresentada na Figura 10.

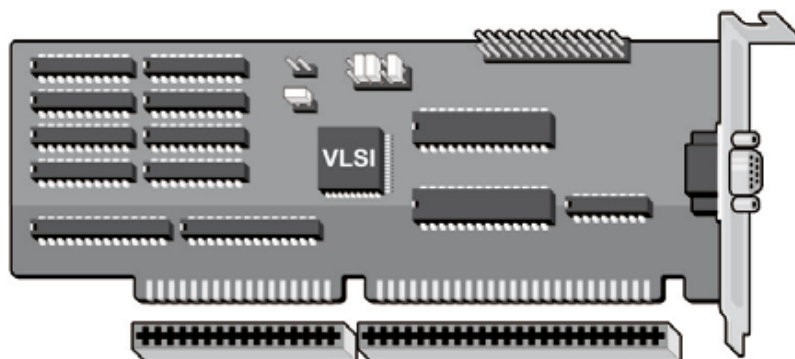


Figura 10 – Estrutura Geral da Placa ISA: 16 bits

Em 1987, a IBM introduziu o padrão MCA (*Micro Channel Architecture*) propondo um novo barramento de dados para substituir o

ISA. Esse barramento é síncrono e utiliza comunicação com palavras binárias de 32 bits (disponível para os processadores 386) e a sua frequência de operação é de 10 MHz. A sua velocidade de operação é 32 Mbps e foi bastante utilizada em placas de vídeo e HDs.

A COMPAQ e outros fabricantes de hardware não concordaram com o padrão MCA da IBM, gerando um novo padrão, em 1988, denominado EISA (*Extended Industry Standard Architecture*), como resposta ao modo como a IBM pretendia fazer o licenciamento do barramento MCA. Foi inicialmente desenvolvido pela COMPAQ com a intenção de retirar a IBM da condução do futuro dos PCs.

Uma das grandes preocupações dos fabricantes durante o desenvolvimento do EISA, foi manter a compatibilidade com o ISA. O resultado foi um slot com duas linhas de contatos, capaz de acomodar tanto placas EISA quanto placas ISA de 8 ou 16 bits, conforme observado na Figura 11.

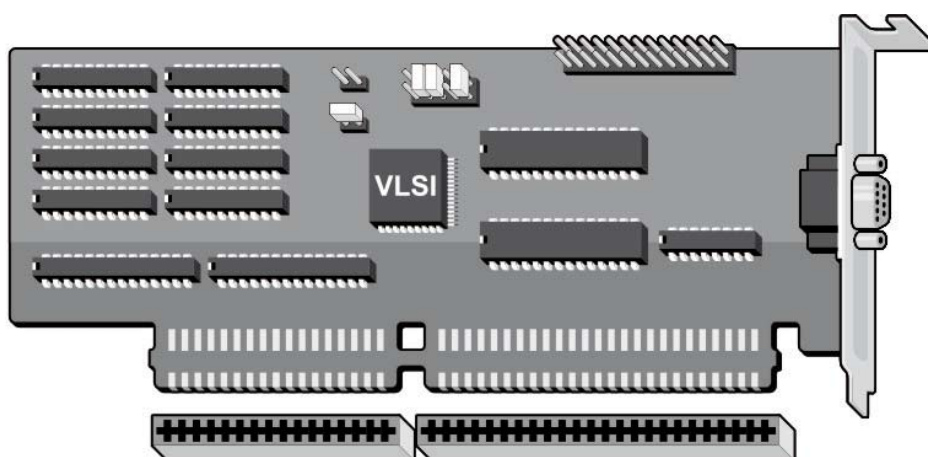


Figura 11 – Estrutura Geral da Placa EISA: 32 bits

Esse barramento síncrono também utiliza comunicação com palavras binárias de 32 bits (compatível para os processadores 386) com frequência de operação é de 8 MHz. A velocidade de operação é 16 Mbps e era utilizado em placas de vídeo e HDs com eficiência comparável à do padrão MCA.

Quatro anos depois (1992) surgiu um barramento específico para uso em aplicações de vídeo, com o objetivo de melhorar o desempenho de vídeo nos computadores pessoais. Esse padrão foi denominado VESA (*Video Electronics Standards Association*) e utilizava 32 bits no barramento de dados, sendo, portanto, compatível com os processadores 486, disponíveis na época. A frequência

de operação é de 25/33 MHz, alcançando velocidade de 25/33 Mbps e sendo utilizado em dispositivos como placas de vídeo e controladoras de disco. Apesar de ser um barramento relativamente rápido, o VESA apresentou alguns problemas, que levaram dentre outras consequências ao surgimento do barramento PCI (*Peripheral Component Interconnect*). Este barramento manteve a mesma frequência de operação, mas incorporou suporte nativo a *plug-and-play* rompendo com o padrão ISA, o que simplificou muito a pinagem do barramento.

O PCI foi desenvolvido pela Intel sendo também classificado como síncrono, possuindo arbitragem centralizada, com dois níveis para utilizar as prioridades dos dispositivos conectados ao barramento. A Figura 12 apresenta a imagem de uma placa PCI.

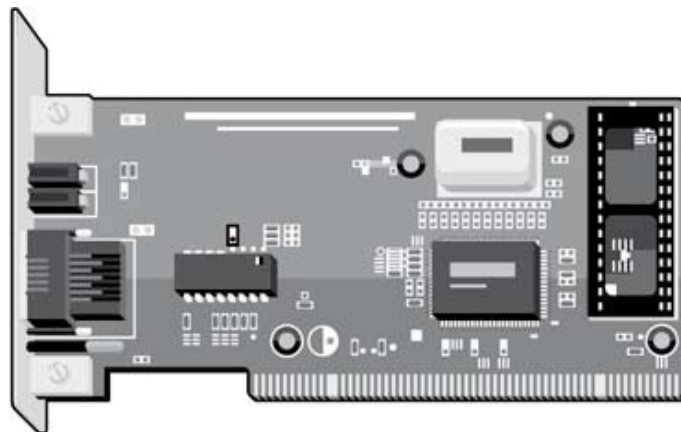


Figura 12 – Estrutura Geral da Placa PCI

O PCI opera nativamente a 33 MHz, o que resulta em uma taxa de transmissão teórica de 133 Mbps. Entretanto, assim como em outros barramentos, a frequência do PCI está vinculada à frequência de operação da placa-mãe. Conforme a frequência das placas foi subindo, passaram a ser utilizados divisores cada vez maiores, de forma a manter o PCI operando à sua frequência original. Em uma placa-mãe operando a 133 MHz, a frequência é dividida por 4 e, em uma de 200 MHz, é dividida por 6.

Com a evolução dos processadores, o barramento PCI foi se tornando cada vez mais lento com relação ao processador e outros componentes, de forma que, com o passar do tempo, os periféricos mais rápidos migraram para outros barramentos, como o AGP (*Accelerated Graphics Port*) e o PCI-Express.

O AGP foi desenvolvido com o propósito específico para o uso das

placas 3D de alto desempenho. O AGP demorou a se popularizar, pois em meados dos anos 90, as placas 3D ainda eram bastante primitivas, de forma que ainda não existia uma demanda tão grande por um barramento mais rápido. O padrão AGP inicial não chegou a ser muito usado, surgindo então o padrão AGP 2X, que introduziu o uso de duas transferências por ciclo, permitindo assim a duplicação da taxa de transferência. Os padrões posteriores foram o AGP 4X e o 8X, que realizam, respectivamente, 4 e 8 transferências por ciclo.

Quando comparamos o AGP com o PCI, podemos dizer que o primeiro padrão de barramento é reservado unicamente à placa de vídeo, enquanto a taxa de transmissão do barramento PCI é compartilhada por todas as placas PCI instaladas. Uma semelhança entre eles é que a frequência de ambos os padrões está atrelada à frequência de operação da placa-mãe.

O PCI Express, por sua vez, é um barramento serial, que pouco tem em comum com os barramentos anteriores, pois é um barramento ponto a ponto, onde cada periférico possui um canal exclusivo de comunicação. A maioria dos barramentos estudados anteriormente utiliza a comunicação paralela. Na comunicação paralela é possível se transmitir vários bits por vez, enquanto que na comunicação serial é transmitido apenas um bit por vez.

Sendo assim, em sua opinião, qual dos dois tipos de comunicação apresenta melhor desempenho?

A princípio, a comunicação paralela aparenta-se mais rápida do que a serial, pois transmitirá um maior número de bits por vez. Entretanto, a comunicação paralela sofre alguns problemas impactando o alcance de *clocks* maiores nas transmissões, devido a problemas de interferência magnética e de atraso de propagação.

O atraso de propagação se dá devido ao fato dos dados transmitidos em paralelo não chegarem ao mesmo tempo ao dispositivo de destino. Isso ocorre porque os fios que interligam a placa-mãe ao dispositivo através do barramento não têm exatamente o mesmo tamanho, fazendo com que os dados transmitidos por fios mais curtos cheguem antes dos demais, ocasionando uma espera no dispositivo, podendo vir a comprometer o desempenho. Por esse motivo, o projeto do PCI Express foi implementado em um barramento serial, que permite operar com *clocks* maiores sem sofrer essas interferências e atrasos de propagação.



Com a evolução tecnológica, a diversidade de periféricos foi além de dispositivos como mouse, teclado e impressora. Atualmente temos vários outros periféricos e o número de interfaces conectadas ao barramento de expansão deve ser bem maior para adequar a essa necessidade. A necessidade de se ter um hardware pequeno para não retroceder aos antigos mainframes também é um fator importante. Os computadores não podem ter um número grande de placas conectadas ao barramento de expansão, pois os tornariam máquinas que ocupavam grandes espaços físicos.

Dessa forma, em 1995, para resolver estes problemas, surgiu o padrão USB (*Universal Serial Bus*). Em 1997/1998, todas as placas mães passaram a contemplar, pelo menos, duas portas USB. As primeiras versões estabelecidas datam de 1994, entretanto as versões que entraram para uso comercial em larga escala foram a 1.1 (setembro de 1998) e a 2.0 (abril de 2000).

O padrão USB permite fácil e rápida a conexão de diversos tipos de aparelhos, tais como, câmeras digitais, HDs externos, pendrives, mouses, teclados, MP3-players, impressoras, scanners, leitor de cartões, dentre outros, utilizando o mesmo de conector para conectar todos os dispositivos ao computador.

Além disso, com o advento do USB, deixou de ser necessário abrir o computador e configurar *jumpers* e/ou *IRQs* para conectar dispositivos. Os dispositivos USB são “Plug and Play”, ou seja, são projetados para serem conectados ao computador e utilizados imediatamente. Inclusive, é comum se encontrar portas USB em vários outros aparelhos, como TVs, aparelhos de som, dentre outros.

Para o USB, o computador atua como um hospedeiro podendo conectar até **127 dispositivos**, diretamente ou através de Hubs USB. Se você não sabe o que é um Hub USB não se preocupe que falaremos dele mais adiante.

Os cabos individuais USB podem ter até 5 metros, porém ao serem utilizados conjuntamente com os hubs, os dispositivos conectados via USB podem ficar até 30 metros de distância do hospedeiro, ou seja, utilizando, no máximo, seis cabos de 5 metros cada um.

Os cabos USB contam com quatro fios internos, sendo dois deles responsáveis pela alimentação elétrica. Os outros dois referem-se a um par trançado responsável pela transmissão de dados, conforme podemos observar na Figura 13.



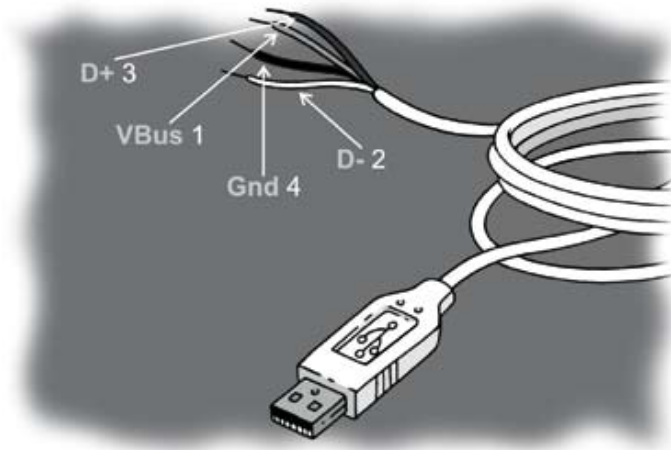


Figura13 – Cabos USB

A alimentação dos dispositivos USB depende do tipo do dispositivo. Para dispositivos como mouse e teclado, a alimentação pode ser retirada diretamente do computador, sendo, portanto fornecida pelos cabos de alimentação elétrica do USB. Para dispositivos com maior consumo de energia, como impressoras e scanners, uma parte da energia é fornecida pelo barramento e a outra parte é proveniente de uma fonte externa.

Como os computadores possuem em geral, no máximo 5 portas USBs, a forma de permitir que muitos dispositivos sejam conectados ao host é através de um dispositivo que multiplique o número de portas disponíveis. Esse dispositivo é exatamente o hub USB que falamos anteriormente. Você poderá identificá-lo na Figura 14.



Figura 14 – Hub USB

Os dispositivos USB são **Hot-Swap** (conectáveis “a quente”), ou seja, podem ser conectados e desconectados a qualquer momento. Eles também podem ser colocados no modo **sleep** (hibernar) pelo computador hospedeiro para não retirarem energia do barramento

quando não estão sendo utilizados, evitando assim o desperdício.

Um exemplo de um computador hospedeiro conectando vários dispositivos USBs diretamente e via hub é ilustrado na Figura 15.

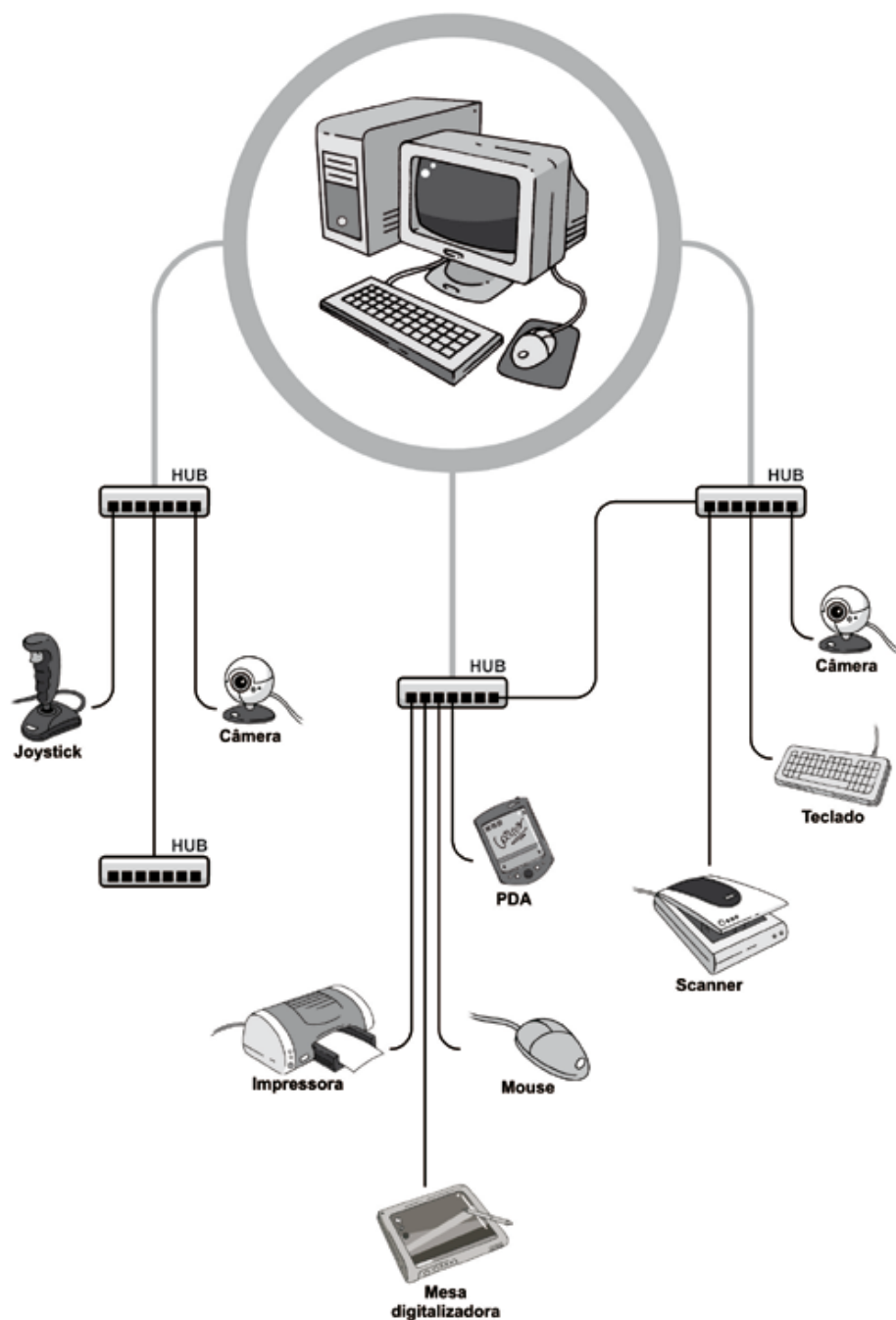


Figura15 – Host com Dispositivos Conectados via USB

Para realizar a comunicação entre os dispositivos conectados via USB e o computador hospedeiro é utilizado um protocolo.

Esse protocolo estabelece que para que seja iniciada a comunicação ocorrerá o processo de enumeração. Esse processo baseia-se no

fato que o computador precisa emitir um sinal para encontrar os dispositivos conectados e estabelecer um endereço para cada um deles. O computador hospedeiro (também denominado host) recebe a informação de que tipo de conexão o dispositivo conectado utiliza. Ao se conectar ao barramento USB, o dispositivo informa através desse protocolo qual é o tipo de comunicação que será utilizada com o host. Existem quatro possibilidades:

- » Por Volume: Também conhecido como *Bulk*, é o tipo de transmissão utilizada por dispositivos que lidam com grandes volumes de dados, como impressoras e scanners. Esse método utiliza recursos de detecção de erro, possibilitando preservar as informações;
- » Por Interrupção: Conhecido por *Interrupt*, é utilizado para dispositivos que transferem poucos dados, como mouses, teclados e joysticks
- » Isócrono: Do inglês, *Isochronous* é utilizado em transmissões contínuas, em dispositivos como caixas de som, por exemplo. Não utiliza recursos de detecção de erros, para evitar atrasos na comunicação.
- » Por Controle: Utilizado para transmissão de parâmetros de controle e configuração do dispositivo.

Quando o dispositivo é enumerado pelo host, ele informa o tipo de comunicação que irá utilizar. Dessa forma, o host manterá um registro da total da largura de banda que todos os dispositivos isócronos e de interrupção exigem. Eles podem consumir até 90% da largura de banda disponível. Passando deste limite, o host negará acesso a qualquer outro dispositivo isócrono ou de interrupção. As transferências de grandes volumes e de controle utilizam a largura de banda restante.

Atualmente, o USB encontra-se na versão 2.0, mas o padrão 3.0 já foi especificado, embora ainda não tenhamos disponibilidade de dispositivos no mercado.

Após apresentarmos essa breve evolução dos padrões comerciais e as características principais de cada um, passaremos para o estudo de mais um subsistema que será apresentado no Volume 3: o subsistema de memória.

Mas antes de passarmos ao próximo volume, é importante você verificar os exercícios propostos na seção a seguir e tentar resolver



#### Saiba Mais

Você sabia que no final de 2008 foi especificado o padrão USB 3.0? Para consultar as inovações desse padrão consulte: <http://www.infowester.com/usb.php>.

pelo menos aquelas que foram propostas pelo seu professor, no ambiente virtual.



### **Aprenda Praticando**

Agora que você já leu a teoria e os exemplos apresentados, sugerimos que você tente desenvolver algumas questões sobre barramentos. Nesta seção é apresentada uma lista de exercícios que você poderá tentar resolver. Caso tenha alguma dúvida, entre em contato com o seu professor ou tutor para solucioná-la. É importante que você passe para o módulo seguinte com todas as suas dúvidas esclarecidas.

#### **Lista de Exercícios**

- 1) Explique o que você entende por barramento de um computador
- 2) Com relação aos barramentos, que tipos de sinais poderão trafegar nos mesmos e que linhas do barramento transportam cada um desses elementos?
- 3) Quais os tipos de sinais de controle que podemos encontrar nos barramentos e o que significam cada um deles?
- 4) Quais as vantagens e desvantagens da utilização dos barramentos nos sistemas computacionais?
- 5) Qual a diferença entre barramentos síncronos e barramentos assíncronos?
- 6) Para evitar que dois ou mais dispositivos tornem-se mestres ao mesmo tempo, existe a arbitragem de barramento. Existem duas formas de se fazer isso. Que formas são essas e como cada uma delas funciona?
- 7) Quais os principais tipos de barramentos estudados?
- 8) Quais as principais características dos barramentos ISA e EISA? Existe alguma compatibilidade entre eles?
- 9) Explique porque surgiu o barramento VL-VESA. Quantos cartões ele suporta e como deverão ser feitas as suas expansões?

- 10) Quais as principais características do barramento PCI?
- 11) Explique porque surgiu o barramento USB. Quais problemas ele deveria solucionar?
- 12) Explique para que serve um hub USB. Quantos dispositivos podem ser ligados via barramento USB direta ou indiretamente?
- 13) O que significa um equipamento ser hot-swapped?
- 14) Como é feita a transferência de dados em um barramento USB?
- 15) Cite e explique os três tipos de transferência de dados em barramentos USB.
- 16) Explique qual é a motivação para a utilização de uma hierarquia de barramentos.



### **Atividades e Orientações de estudo**

Dedique, pelo menos, 4 horas de estudo para o Capítulo 4. Você deve organizar uma metodologia de estudo que envolva a leitura dos conceitos que serão ditos apresentados neste volume e pesquisas sobre o tema, usando a Internet e livros de referência.

Os fóruns temáticos desta disciplina podem ser utilizados para troca de informações sobre o conteúdo no ambiente virtual, pois a interação com colegas, tutores e o professor da disciplina irá ajudá-lo a refletir sobre aspectos fundamentais tratados aqui. Os chats também serão muito importantes para a interação em tempo real com o seu tutor virtual, seu professor e seus colegas.

Também é importante que você leia atentamente o guia de estudo da disciplina, pois nele você encontrará a divisão de conteúdo semanal, ajudando-o a dividir e administrar o seu tempo de estudo semanal. Procure responder as atividades propostas como atividades somativas para este capítulo, dentro dos prazos estabelecidos pelo seu professor, pois você não será avaliado apenas pelas atividades presenciais, mas também pelas virtuais. Muitos alunos não acessam o ambiente e isso poderá comprometer a nota final. Não deixe que isso aconteça com você!

## Considerações Finais

Olá, Cursista!

Esperamos que você tenha gostado d o segundo módulo da disciplina Infraestrutura de Hardware. No próximo módulo, passaremos a estudar o subsistema de memória, onde estudaremos diversos tipos de memórias e seus princípios fundamentais.

Aguardamos sua participação no próximo módulo.

Até lá e bons estudos!

Juliana Regueira Basto Diniz

Abner Corrêa Barros

*Professores Autores*



## Referências

STALLINGS, William. **Arquitetura e Organização de Computadores**. 5. ed.

PATTERSON, D. A. e Hennessy, John L. **Organização e Projeto de Computadores**. LTC, 2000.

TANENBAUM, Andrew S. **Organização Estruturada de Computadores**. 4. ed. Tradução Helio Sobrinho. Rio de Janeiro: Prentice-Hall, 2001.

## Conheça os Autores

**Juliana Regueira Basto Diniz** possui graduação em engenharia eletrônica pela Universidade Federal de Pernambuco, mestrado e doutorado em Ciência da Computação pela Universidade Federal de Pernambuco. Atualmente é professora da Universidade Federal Rural de Pernambuco (UFRPE), desenvolvendo trabalhos no grupo de Educação a Distância desta universidade. Seus temas de interesse em pesquisa são: Sistemas Distribuídos, Computação Ubíqua e Ensino a Distância.

**Abner Corrêa Barros** é mestre em Ciência da Computação com foco em Engenharia de Hardware pelo Centro de Informática da Universidade Federal de Pernambuco. Possui graduação em Ciência da Computação pela mesma universidade. Atualmente é professor da disciplina de Organização e Arquitetura de Computadores da Faculdade Maurício de Nassau e Engenheiro de Hardware da Fundação de Apoio ao Desenvolvimento da UFPE (FADE), atuando em um projeto de convênio entre o Centro de Informática da UFPE e a Petrobrás. Suas áreas de interesse e pesquisa são: Hardware Reconfigurável, Arquitetura de Cores Aritméticas e Computação de Alto Desempenho em *Field-Programmable Gate Array* (FPGA).