

Arquitetura e Organização de Computadores

Bases numéricas, representação dos dados e instrução de máquinas

Prof. Ms. Wesley Viana

- Unidade de Ensino: 3
- Competência da Unidade: Aprofundar o conhecimento sobre base numéricas e representações de dados
- Resumo: Nesta unidade será apresentado as representações das linguagens incluindo como o computador se comunica em sua linguagem de máquina.
- Palavras-chave: Sistemas numéricos; ciclo de instruções, pipeline; Linguagem Assembly.
- Título da Tele aula: Bases numéricas, Representação dos dados e instruções e Instruções de máquinas
- Tele aula nº: 03

Contextualização

O principal objetivo é conhecer e compreender os conceitos e desenvolvimentos históricos de arquitetura e organização de computadores, assim como a estrutura de computadores.

Para isso, serão apresentados todos os conceitos relacionados à **representação binária de informação**, a composição de outras representações relacionadas ao sistema binário e a classificação de números binários.

Após apresentar os números binários, também serão abordados os números **octais e hexadecimais**, bem como as **conversões de base entre todos estes diferentes sistemas de numeração**.



www.shutterstock.com -700974610

Fonte: Shutterstock

Contextualização

Todo este conhecimento será necessário para que seja possível projetar instruções de máquinas, que nada mais são do que um conjunto de números binários que será o coração do processador que iremos projetar mais adiante e as linguagens de máquina e montagem (Assembly) para projetar as instruções .

Curioso para saber como?

Vamos descobrir juntos!



www.shutterstock.com -700974610

Fonte: Shutterstock

Sistemas numéricos: conceitos, simbologia e representação de base numérica

Sua missão

A sua primeira tarefa é fazer conversões e validações.

Considere que você faz parte de um time de desenvolvimento de hardware de uma empresa X que necessita de uma verificação em sua arquitetura.

Após adquirir o conhecimento desta seção, você é a pessoa indicada para realizar os testes necessários.

A empresa vai apresentar um conjunto de números binários trabalhados pelo processador e você terá de apresentar um parecer de número de acertos e erros de forma que o projeto seja avaliado e as decisões de correção possam ser devidamente tomadas.

Nesse sentido, você deverá avaliar os números apresentados na tabela apresentada a seguir:



Fonte: Shutterstock

Sua missão

Você, então, deverá fazer a conversão, preencher as linhas em branco e, depois, preencher a coluna de *validação*, com o valor CORRETO ou ERRADO.

Resultado esperado binário	Resultado obtido octal	Resultado obtido hexadecimal	Validação
0001 1110		1E	
1101 0111	315		
1111 1101		FF	
1010 0101		A5	
0111 1111	154		
1100 0011	303		
1111 1111		AA	



Fonte: Shutterstock

Sistemas de Numeração

Sistema	Bases	Algarismos
Binário	2	0,1
Octal	8	0,1,2,3,4,5,6,7
Decimal	10	0,1,2,3,4,5,6,7,8,9
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Sistemas de Numeração

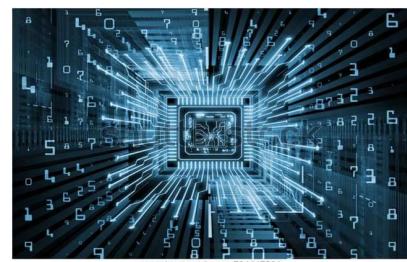
DECIMAL	BINARIO	OCTAL	HEXADECIMAL
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Fonte: Tangon e Santos 2016, p.129

Sistema Decimal

- É o sistema mais comum;
- Composto por **10 símbolos (0,1,2,3,4,5,6,7,8,9);**

10^2	10^1	10^0
3	8	7



Fonte: Shutterstock

Realizando o cálculo , temos:

$$(3 \times 10^2) + (8 \times 10^1) + (7 \times 10^0) =$$

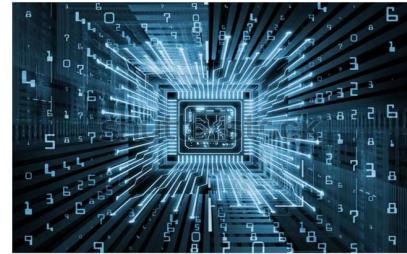
$$= 300 + 80 + 7$$

$$= 387$$

Sistema Binário

- Composto por **dois símbolos: 0s e 1s**;
- O zero no sistema binário representa a ausência de tensão, enquanto o 1 representa uma tensão.

1	0	1	1	0
2^4	2^3	2^2	2^1	2^0



Fonte: Shutterstock

Realizando o cálculo , temos:

$$(1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = \\ = 16 + 0 + 4 + 2 + 0 \\ = \mathbf{22} \text{ em decimal}$$

Sistema Octal

- Composto por 8 símbolos (0,1,2,3,4,5,6,7);
- Este sistema foi utilizado como modelo alternativo ao binário como uma forma mais compacta;
- Hoje utiliza-se mais o hexadecimal.

1	6	7
8^2	8^1	8^0



Fonte: Shutterstock

Realizando o cálculo , temos:

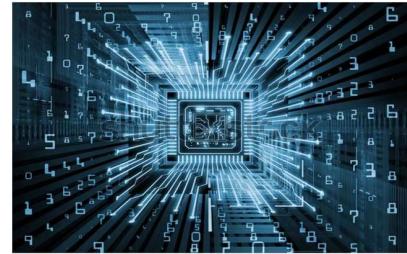
$$(1 \times 8^2) + (6 \times 8^1) + (7 \times 8^0) = \\ = 64 + 48 + 7 \\ = \mathbf{119} \text{ em decimal}$$

Sistema Hexadecimal

Composto por **16 símbolos (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)**;

- Utilizados nos microprocessadores.
- Usados endereços de MAC (Endereços Físicos) encontrados nas etiquetas abaixo dos roteadores.

2	F	4
16^2	16^1	16^0

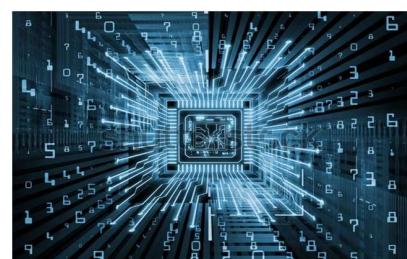
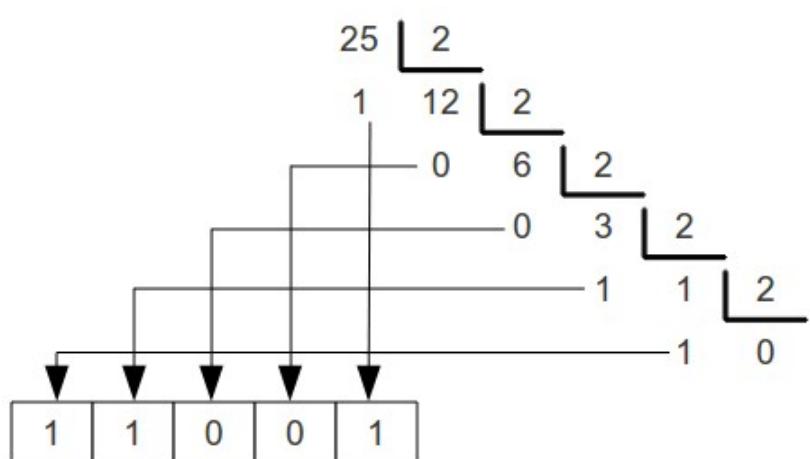


Fonte: Shutterstock

Realizando o cálculo , temos:

$$(2 \times 16^2) + (15 \times 16^1) + (4 \times 16^0) = \\ = 512 + 240 + 4 = \mathbf{756} \text{ em decimal}$$

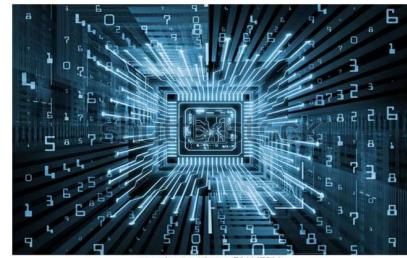
Conversão de decimal para binário



Fonte: Shutterstock

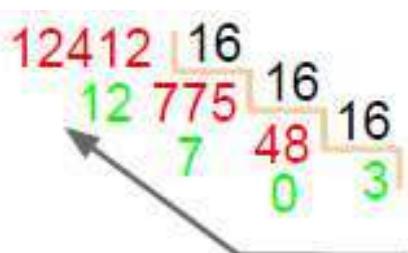
Conversão de binário para decimal

$$\begin{array}{c} \textcolor{red}{1\ 1\ 0\ 1\ 0\ 1}_2 \\ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\ 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \quad \downarrow \\ 32 + 16 + 0 + 4 + 0 + 1 = 53 \\ \textbf{110101}_2 = 53_{10} \end{array}$$



Fonte: Shutterstock

Conversão de decimal para hexadecimal



Fonte: Shutterstock

12412 Decimal = 307C Hexadecimal

Aquitetura de um processador

Uma definição de arquitetura de um processador deve vir acompanhada, obrigatoriamente, da indicação de como o número binário é considerado, sendo que existem duas possibilidades.

Para ler o número binário da esquerda para a direita (dizemos que o MSB é o primeiro e o LSB é o último. **MSB** é uma sigla para o **bit mais significativo** e **LSB** é o **bit menos significativo**.

Se o primeiro é o **MSB**, ele é o que representa o maior valor, sendo que o número deve ser lido da esquerda para direita. Entretanto, existem arquiteturas que o primeiro bit é o **LSB** e o último é o MSB, fazendo com que o número seja interpretado de maneira invertida.



Fonte: Shutterstock

Conversão de bases e validação

Ao ser informado do trabalho a ser realizado, você foi apresentado a uma tabela que continha em cada linha o resultado de uma operação feita pelo processador e o resultado que era esperado.

Como são diversos testes realizados e cada um por uma pessoa diferente cada resultado estava em uma base diferente dificultando seu trabalho. **O resultado é apresentado na tabela seguinte:**

Binários	Conversão decimal	Resultado obtido octal	Resultado obtido hexadecimal	Validação
0001 1110	30	36	1E	Correto
1101 0111	215	315 (deveria ser 327)	CD (deveria ser D7)	Errado
1111 1101	253	377 (deveria ser 375)	FF (deveria ser FD)	Errado
1010 0101	165	245	A5	Correto
0111 1111	127	154 (deveria ser 177)	F7 (deveria ser 7F)	Errado
1100 0011	195	303	C3	Correto
1111 1111	255	358 (deveria ser 377)	AA (deveria ser FF)	Errado

Introdução a Instruções de máquinas

Sua missão

A sua **segunda tarefa** é melhorar o **tamanho e qual será a forma da instrução escolhidos para o projeto de um processador**.

Considere que você, profissional de computação, foi contratado para desenvolver um novo núcleo para um processador e precisa agora resolver diversos problemas relacionados à arquitetura do processador.

É necessário que você, como projetista, crie todo um conjunto novo de instruções que será utilizado como base para a construção do processador.

Questionamentos:

Qual seria o tamanho desta instrução?

O que é necessário representar nesta instrução?



Fonte: Shutterstock

Sua missão

Sabendo que **uma instrução é nada mais que um conjunto de algarismos binários**, sendo assim, o que cada um destes algarismos representa?

Qual parte da informação deve colocada em cada parte dos bits? Existe uma regra?

A **instrução** referencia **espaços na memória** que serão utilizados para recuperar informações para o processador e, também, para salvar os resultados de processamento.

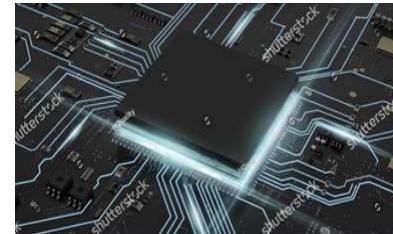
Estas informações deverão também ser incluídas no projeto da instrução, porém, como fazer isso da melhor forma possível?



Fonte: Shutterstock

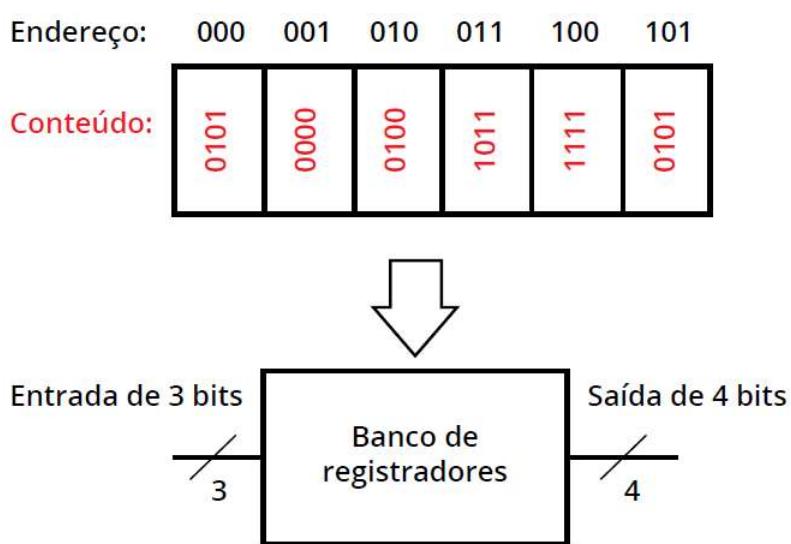
Contextualizando

- Toda a **informação utilizada em um processador deve ser armazenada**, por isso sempre que compramos um computador verificamos algumas informações tais como: sua capacidade de memória RAM, seu HD, se o processador possui um, dois ou três níveis de memória cache .
- O tamanho da **memória** está diretamente relacionado à complexidade de seu **gerenciamento**.
- A unidade básica de armazenamento é o registrador.
- Existe no processador um banco de registradores.
- Este banco de registradores funciona da seguinte maneira: você passa um endereço de um determinado registrador e o bloco de *hardware* **retorna seu conteúdo**.



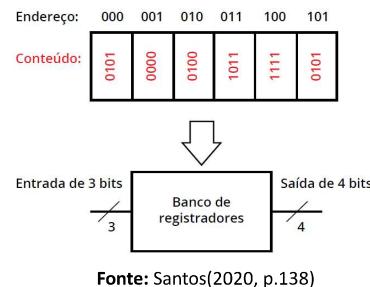
Fonte: Shutterstock

Banco de Registradores



Manipulação dos registradores

- O acesso ao **dado** é feito por meio do **endereçamento direto**, ou seja, **o endereço enviado é o próprio endereço do registrador** e ativa apenas o setor da memória correspondente, tornando este tipo de acesso extremamente **rápido** e permitindo que seja **implantado** dentro do **processador para o armazenamento de informações**.
- A **manipulação dos registradores** é realizada por meio de **instruções dos processadores, do tipo I, tipo R e tipo J**.
- As instruções do **tipo I** manipulam **endereços de memória ou constantes**.
- As instruções do **tipo R** podem realizar **a manipulação e as operações entre registradores**.
- As instruções do **tipo J** realizam **desvios de instruções a serem executados**.



Processadores MIPS

- Hennessy e Patterson (2017) propõem um processador chamado **MIPS**, cuja arquitetura foi utilizada como base de diversos **processadores comerciais**.
- O **MIPS possui 32 registradores** (do registrador 0 ao registrador 31) que armazenam dados de 32 bits de tamanho.
- Sendo assim, como deve ser o projeto deste banco de registradores considerando os bits de entrada e de saída?
- Para representar os 32 registradores é necessário um barramento de endereços de **5 bits**, uma vez que $2^5 = 32$, ou seja, (registrador 0 – 00000b, ao registrador 31 – 11111b) e um **barramento de saída de dados de 32 bits**.



Fonte: Shutterstock

Arquitetura de Von Neumann x Arquitetura de Harvard

- Nas duas arquiteturas, o computador possui quatro componentes básicos: **memória, unidade de controle, unidade lógica aritmética (ULA) e entrada/saída de dados.**
- A **diferença** básica entre estas duas arquiteturas é o fato de que na arquitetura de **Von Neumann** a **memória de programa e a memória de dados estão fisicamente no mesmo chip.**
- No caso da arquitetura de **Harvard** existe uma separação física real da **memória de dados e da memória de programas** o que gera um custo maior de implantação porém, a proteção é maior visto que se um vírus for executado, ele irá afetar apenas a memória de programa sem que os dados sejam perdidos.



Fonte: Shutterstock

Hierarquia de memórias

- Após entender os **registradores** e os **tipos de arquiteturas**, o próximo passo é entendermos a **unidade de controle** e a parte de **entrada/saída de dados**.
- Sem este entendimento seria complexo demais compreender como é formado um **conjunto de instruções**.
- Na hierarquia de memória, quando um **dado** não está armazenado em um **registraror** é possível que esteja ou na **memória principal** (representada em nosso sistema por níveis **secundário de cache e também pela memória RAM**) ou na memória **secundária** (representada pelo **HD**).

Instrução

- Em uma **instrução** é possível também acessar **endereços de memória** que não são endereços de **registradores** e, nestes casos, é necessário que se tenha o endereço propriamente dito do dado ou então que se passe o valor numérico a ser trabalhado pelo processador.
- Este tipo de **endereçamento** é chamado de **imediato** e irá gerar um tipo específico de instruções.
- Após apresentar alguns conceitos básicos, vamos à **definição de instrução**:
- Segundo Hannessy e Patterson (2017), uma **instrução** é um conjunto de **bits (0's e 1's)**, entendidos pelo **processador** como sinais eletrônicos, que podem ser representados por um **conjunto de números**.



Fonte: Shutterstock

Instrução

- Cada um destes **números** representa **algo específico para o processador** e a **junção** destes **números** em uma **palavra única** é o que chamamos de **instrução**.
- Para entender o papel de uma instrução vamos analisar os blocos que compõem o processador e o que cada um deve receber e como se dá o caminho dos dados através dos blocos.

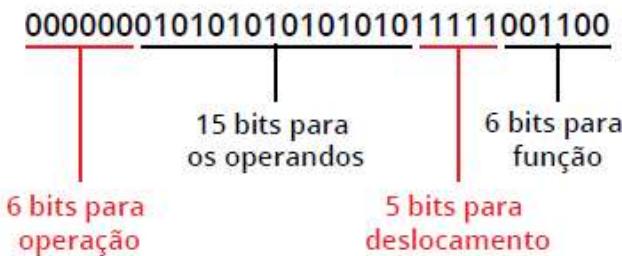


Fonte: Santos(2020, p.140)

Projeto de Instrução

O conceito de instrução deve ser assimilado de forma completa.

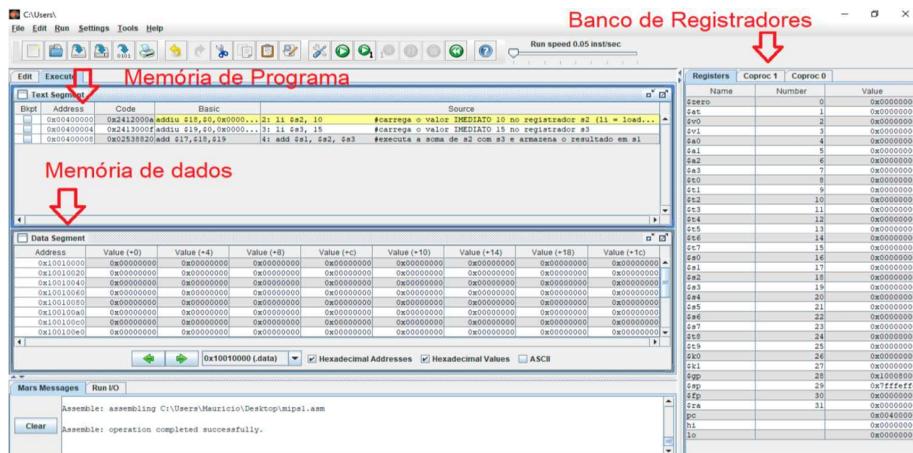
Portanto vamos utilizar uma imagem que representa uma instrução de 32 bits com a mesma divisão proposta por Hennessy e Patterson (2017) para visualizar um projeto real.



Fonte: Santos(2020, p.141)

Modos de endereçamento em processadores

O simulador é um programa gratuito gerado em java (.jar) que deve ser apenas executado. Ex.: **os valores nos registradores e realizar a sua soma.**



Criação de um modelo para um conjunto de instruções

O **processador** que iremos projetar deve ser simples, porém pode ser pensado para um tipo de operações específicas.

34

Os **processadores** possuem sempre uma **unidade** que deve ser especializada na execução de **operações aritméticas como soma, subtração, divisão, multiplicação**, dentre outras .

A segunda questão importante é que este conjunto de instruções deve ter definido **um tamanho de palavra que o processador** será capaz de aceitar.

Considerando que o processador terá de se comunicar com outros blocos é possível escolher um tamanho de palavra padrão. Os processadores mais novos estão operando em **64 bits (que é o tamanho da sua palavra)**, porém, o nosso processador pode ser projetado considerando que **um dado de 64 bits** irá precisar de toda uma estrutura de suporte maior como o barramento de transmissão de dados, os registradores para armazenar os dados, dentre outras estruturas.

Sendo assim, para manter um tamanho de arquitetura pequeno e compatível com os projetos já executados no mercado vamos optar por uma **arquitetura de 32 bits**.

As duas primeiras etapas foram então cumpridas e justificadas. Veja: você como projetista poderia escolher qualquer número de bits e um núcleo especializado em outras funções.

Porém, modificar estas características do problema exige normalmente conhecimentos mais avançados sobre as aplicações para as quais o processador seria desenvolvido, e um tamanho maior de palavra deve ser justificado com a necessidade de mais bits para representação. Um número muito pequeno destes bits pode causar um problema caso sua arquitetura do processador precise passar por um upgrade e isso também deve ser considerado.

Agora, **para cada um dos bits, ou conjunto de bits**, devemos escolher uma funcionalidade correspondente.

Como ainda não conhecemos a fundo as funcionalidades necessárias, é possível tentar sem medo definir este modelo de instrução.

Mais à frente no estudo de arquitetura de computadores poderemos analisar a resposta que demos aqui e entender o motivo de ser boa ou ruim para determinados aspectos.

O processador lida normalmente com **operações numéricas**, sendo que os números envolvidos estarão armazenados em uma memória.

Assim existe a necessidade de **dois tipos de campos**: um onde acessamos o número em um registrador (como nas instruções do tipo R apresentadas) e outro que o número é obtido diretamente na instrução (como nas instruções do tipo I apresentadas).

Existem processadores inteiros como o **MIPS** (HENNESSY; PATTERSSON, 2017) que são definidos com 32 instruções e outros como os processadores que possuem centenas.

O número de bits que reservarmos para o tipo de instrução deve ser capaz de identificar unicamente cada uma delas.

Sendo assim podemos deixar **1 byte para a identificação de instruções, ou seja, 8 bits**.

Também é comum reservar **bits para controle da arquitetura que no nosso caso serão 6 bits**.

Os outros 18 bits podem ser divididos da seguinte maneira: para instruções do **tipo R, três grupos de 6 bits**, e para instruções do **tipo I dois campos um de 6 bits e um de 12 bits**.

Vamos analisar esta resposta. Reservando 1 byte para instruções poderemos identificar unicamente instruções, ou seja, 256.

Este número é razoável já que estamos projetando um processador apenas para cálculos matemáticos, pois são poucas as possíveis operações matemáticas.

Os outros **6 bits para controle do sistema** são importantes já que podem ser utilizados para configurar outras partes do processador.

Neste caso permitimos, com 6 bits, que existam opções de configurações totalizando 64 o que é suficiente também para o controle dos blocos lógicos.

O tipo R possui 3 campos de 6 bits cada.

Um campo de **6 bits para o endereço de um registrador** (lugar onde fica armazenado um valor na **memória**) permite endereçar **64 registradores** (do 0 ao 63).

Normalmente, um projeto de processador simples não possui mais que **40 registradores** sendo um número aceitável.

Por fim, para a **instrução do tipo I**, temos novamente os 6 bits para endereçar um registrador mantendo o padrão anterior.

No caso da parte imediata deixamos 12 bits, isso quer dizer que quando o programador do processador quiser passar um número direto para o processador poderá usar apenas do 0 ao 4095.

Pronto, projetamos assim a nossa instrução de **32 bits** com:

8 bits para identificação da instrução.

6 bits para controle do processador.

18 bits para informações sobre os operandos.

No **tipo R** são três endereços de 6 bits para os registradores.

No **tipo I** é um conjunto de 6 bits para um endereço e 12 bits para um número imediato.

Entenderam as formas de funcionamento de um computador?



Fonte: <https://gifer.com/en/XIOL9>

Pipeline de instruções

Sua missão

A sua **terceira tarefa** é identificar **as etapas de execução da instrução**.

Visualizando o caminho de dados de um determinado processador com barramentos de *8 bits*, ou seja, a quantidade de dados que flui é de *8 bits*, é possível não só identificar as etapas do processador, como também propor modificações e verificar possíveis otimizações.

Você, como pertence ao grupo de projetistas de *hardware* da X recebeu uma nova tarefa.

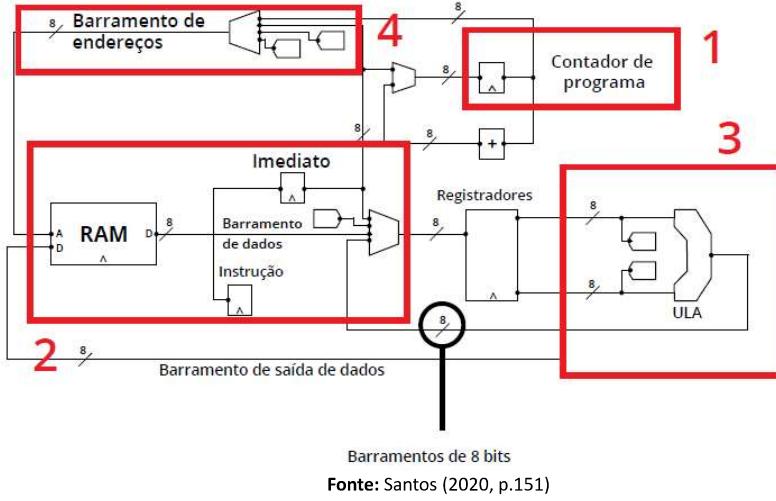
A equipe de engenheiros recebeu a imagem **do caminho de dados de um processador** que será fabricado pela empresa concorrente Y apresentado na Figura:



Fonte: Shutterstock

Sua missão

Arquitetura do processador da empresa X:



Barramentos de 8 bits

Fonte: Santos (2020, p.151)

Sua missão

Para que a sua empresa não seja prejudicada, sua tarefa é identificar neste diagrama as etapas de execução da instrução e se este processador tem potencial para otimizações.

Com o conhecimento adquirido será possível realizar a tarefa e proporcionar uma análise detalhada do caminho de dados do processador da empresa Y.

Você está preparado para aprender como fazer isso?



Fonte: Shutterstock

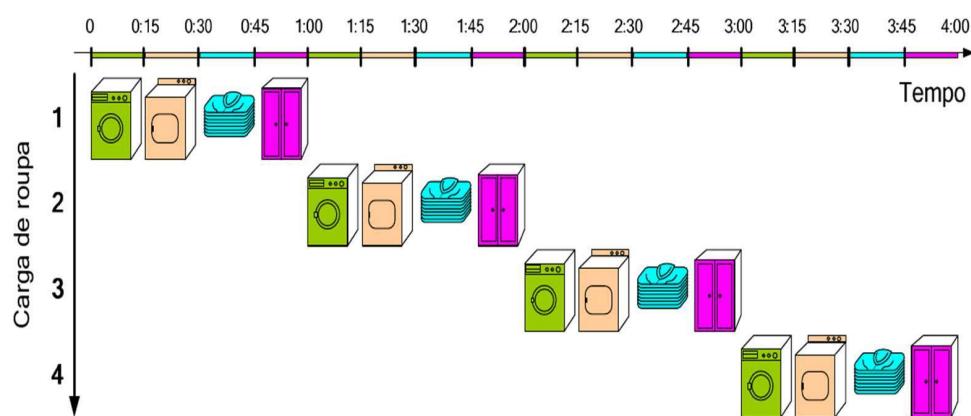
Pipeline

A segmentação de **instruções** (em inglês, **pipeline**) é uma técnica de hardware que permite que a CPU realize a busca de uma ou mais **instruções** além da próxima a ser executada.

Suponha que você tem que arrumar a sua roupa suja, para isso você necessita: (a) Lavar, (b) Secar, (c) Passar e (d) dobrar cada peça de roupa



Pipeline -Pior forma



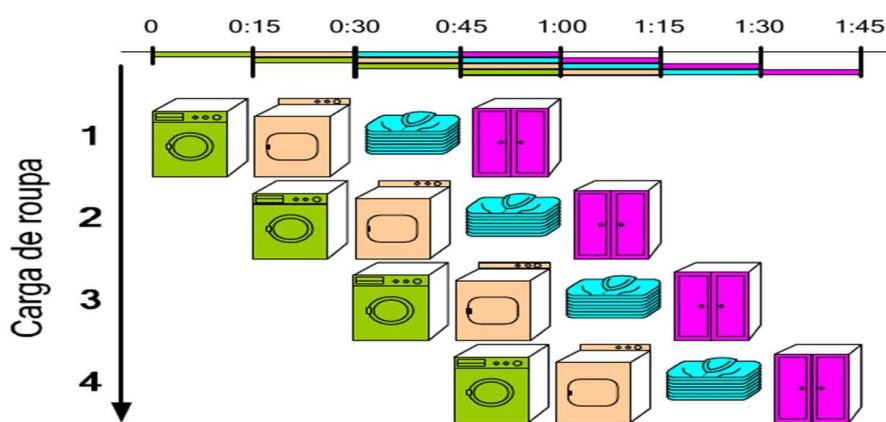
Pipeline

Para incrementar a eficiência de nosso processo podemos realizar o seguinte processo: primeiro lavamos a I peça; tiramos a I peça e colocamos no secador e simultaneamente lavamos a II peça; tiramos para passar a I peça, colocamos para secar a II e adicionamos uma III na lavadora; finalmente dobramos a I, passamos a II, secamos a III e lavamos a IV; repetiremos esse processo sobre todas as peças e vemos que temos um sistema trabalhando em **paralelo**.

Faz o paralelismo de instruções, aumentando a eficiência na execução das instruções.

O pipeline não reduz o tempo de uma instrução, mas agiliza o desempenho de um conjunto de tarefas.

Pipeline



Pipeline

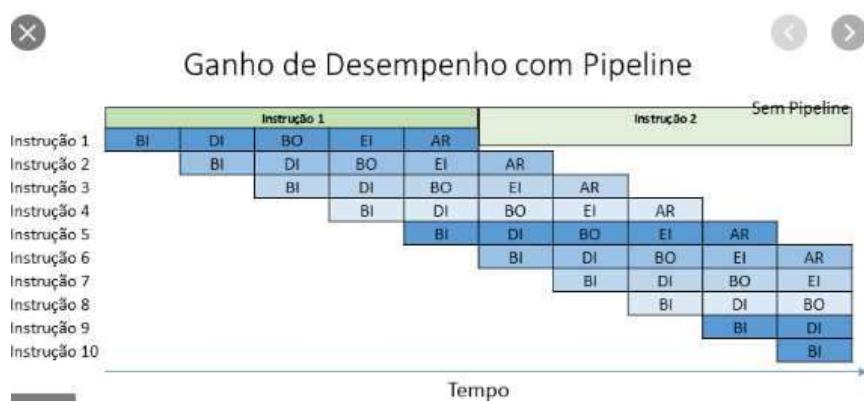
- A divisão das **etapas de execução da instrução** permite que as áreas que já foram utilizadas possam executar a próxima instrução e este conceito é o que chamamos de pipeline.
- Quanto maior o número de estágios no pipeline, maior o número de instruções que podem ser executadas.

Pipeline (no exemplo de 4 estágios):

- Para ele funcionar as **instruções** deviam ser do mesmo tamanho o que impossibilitou os processadores **CISC** aderir a essa estratégia a solução foi colocar um decodificador de instruções que tomava as complexas instruções e transformava para um conjunto resumido de instruções:

Instruction Fetch, Instruction Decode, Execute, Memory access, register Write Back.

Pipeline



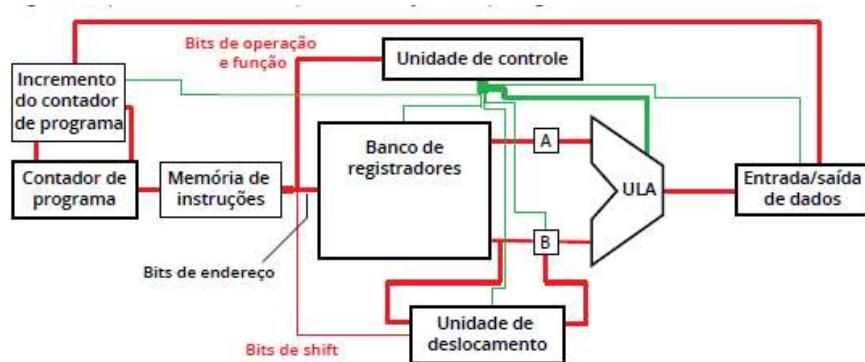
Fonte: sites.google.com

Pipeline

- O conceito de **paralelismo** atualmente é muito difundido, principalmente, pelo **aumento de núcleos dos processadores**.
- Esses núcleos são utilizados em paralelo para a execução de diversas tarefas do computador.
- Porém, a tecnologia de **se colocar vários núcleos em um processador** foi desenvolvida após terem se esgotado as possibilidades de otimização em apenas um núcleo.
- Esta otimização ocorreu por muito tempo considerando o **aumento no clock do processador e no número de instruções** que se conseguia executar em um mesmo ciclo de clock.
- As limitações encontradas no desenvolvimento foram físicas.

Instrução regular

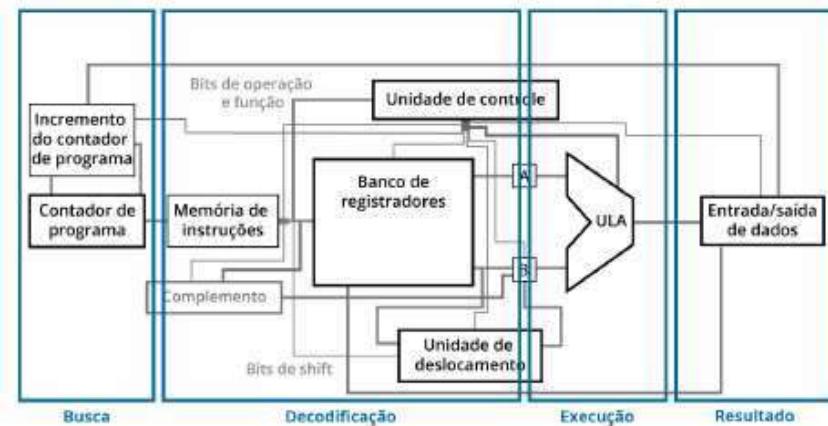
Caminho de dados para a instrução do tipo regular



Fonte: Santos (2020, p.153)

Instrução regular

Caminho de dados com representação da divisão



Fonte: Santos (2020, p.155)

Pipeline

O **bloco 1** representa a fase de **busca**, identificável pela presença do **contador de programa (PC)**, o **bloco 2** seria a **decodificação** pois além de receber informações do bloco 1 está antes da unidade lógica aritmética (em inglês, ALU – arithmetic logic unit), o **bloco 3** por conter a **ALU** é o **bloco de execução** e, por fim, o **bloco 4** é o responsável pela **configuração da gravação em memória**.

Sendo assim, o processador proposto possui as 4 etapas implementadas podendo conter, portanto, os **4 estágios básicos de pipeline**. O diferencial principal deste processador é que todos os **barramentos são de 8 bits** indicando que este é o tamanho da palavra do processador. Com isso temos a análise esperada da arquitetura do processador da empresa Y.

Introdução à Linguagem Assembly

Sua missão

Agora, você, como parte da equipe de desenvolvimento da X, teve uma nova atribuição.

Considerando os problemas solucionados das arquiteturas anteriores, agora você foi designado para a função de **elaboração dos mnemônicos da linguagem de montagem do processador que está sendo desenvolvido pela empresa**.

Nesta etapa você será apresentado ao **diagrama** do processador e aos tipos de instrução que ele irá suportar.

Baseando-se nessas informações, e considerando as otimizações já discutidas em seções anteriores, você deverá **criar as instruções** a serem utilizadas para o desenvolvimento dos programas em linguagem de montagem e, futuramente, dos compiladores de linguagem de alto nível para a arquitetura do

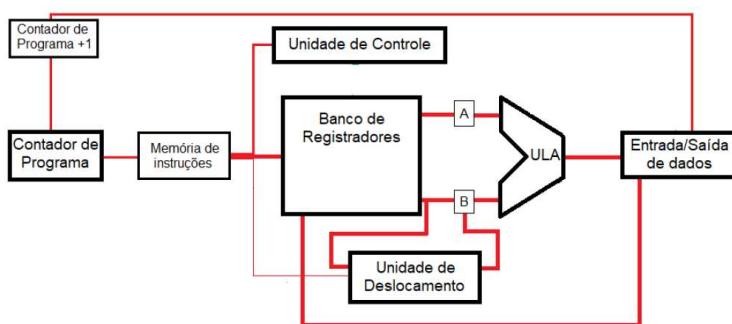


Fonte: Shutterstock

Sua missão

processador da X, mostrado na Figura.

O **processador** possui apenas instruções do tipo R e do tipo de desvio incondicional (j). Dessa forma, não será possível ter instruções imediatas. O diagrama do processador apresentado é mostrado a seguir:



Sua missão

Considerando que o hardware está pronto, crie uma tabela com possíveis instruções – os ajustes a serem feitos na arquitetura serão realizados pelos outros idealizadores do projeto. Lembre-se de que buscamos uma arquitetura otimizada.

Lembre-se de verificar qual seria o conjunto mínimo de instruções necessárias para a construção de programas, pois, caso falte alguma instrução em seu conjunto, pode ser que não seja possível escrever programas corretamente utilizando sua linguagem de montagem.

Preparado para mais este desafio?

Contextualizando

- ✓ Um **software pode ser compilado, interpretado e traduzido** ou pode utilizar uma técnica híbrida para que as instruções sejam **executadas**.
- ✓ Nesse processo, o código escrito em uma **linguagem de programação de alto nível**, como Java, C++, C#, será convertido em um código intermediário e, no caso dos programas compilados, convertido novamente para gerar os comandos que o processador é capaz de executar.
- ✓ Esse código intermediário é chamado de **linguagem de montagem ou assembly**.
- ✓ Essa linguagem foi criada com objetivo de **facilitar** o trabalho para os programadores, que anteriormente deveriam saber os códigos binários de cada instrução para poder programar.



www.shutterstock.com - 161760067

Fonte: Shutterstock

Compiladores

O **compilador** é um programa que possui como entrada um **arquivo** de determinada linguagem de programação e **retorna um programa executável** de uma determinada linguagem, para uma determinada arquitetura.

Existem dois grandes grupos de **operações que são executadas**: as operações de **análise**, que englobam operações específicas do compilador, e as operações de **síntese**, que são voltadas para a geração do código em si em uma arquitetura-alvo e englobam a geração de um código intermediário, a otimização desse código e a geração de um **código-objeto**, que será **executado pelo processador**.



Fonte: Shutterstock

Programação em linguagem de máquina

O **conjunto de instruções de um processador** é a base para a construção de sua **arquitetura**, como visto anteriormente. Não é possível **adicionar uma instrução ao conjunto de instruções sem que exista o caminho de dados necessário para sua execução**.

Caso seja uma instrução básica necessária para o correto funcionamento dos programas, é necessário **modificar o hardware**.

A vantagem de se definir tipos de instrução, como fizemos anteriormente, é que qualquer instrução que utilize um **hardware existente** e possa ser expressa em um dos formatos básicos pode ser adicionada ao conjunto sem problemas.



Fonte: Shutterstock

Programação em linguagem de máquina

Para **acessar um registrador**, utilizamos, em uma **instrução, o cífrão \$ seguido do nome do registrador**.

No caso de um valor armazenado na **memória**, é necessário carregá-lo para um registrador, utilizando a **instrução lw**.

O **resultado** de uma operação, que deve ser gravado na memória, tem que inicialmente ser salvo em um registrador e, em seguida, carregado para a memória com a **instrução sw**.

Agora, vamos explicar **os tipos de instruções**, iniciando com as instruções **do tipo R**, que possuem o formato: **nome da instrução, registrador de destino, registrador de origem 1, registrador de origem 2**.



Fonte: Shutterstock

Programação em linguagem de máquina

Os campos de **deslocamento (shamt)** e de **função**, são **preenchidos automaticamente** quando o código binário é gerado a partir da **linguagem de máquina**. Exemplos

	Formato da Instrução	Operação Realizada
1	add \$rd, \$r1, r2	Soma o conteúdo de r1 com r2 e armazena em rd.
2	addu \$rd, \$r1, \$r2	Soma o conteúdo de r1 com r2, sem considerar o sinal dos números (só números positivos), e armazena em rd.
3	sub \$rd, \$r1, \$r2	Subtrai o conteúdo de r2 do conteúdo de r1 e armazena o resultado em rd.
4	or \$rd, \$r1, \$r2	Realiza a operação de “ou” lógico entre os bits armazenados em r1 e r2 e coloca o resultado em rd.
5	and \$rd, \$r1, \$r2	Realiza a operação de “e” lógico entre os bits armazenados em r1 e r2 e coloca o resultado em rd.
6	slt \$rd, \$r1, \$r2	Se o conteúdo de r1 for menor que o conteúdo de r2, armazena 1 em rd; caso contrário, armazena 0.
7	sll \$rd, \$r1, shamt	Armazena em rd os bits que estão armazenados em r1, deslocados para a esquerda na quantidade expressa por shamt (um inteiro).

- **Formato das instruções tipo R (R-type) e seus campos**

op	rs	rt	rd	shamt	funct
----	----	----	----	-------	-------

op ↳ operação básica da instrução (opcode)
rs ↳ o primeiro registrador fonte
rt ↳ o segundo registrador fonte
rd ↳ o registrador destino
shamt ↳ shift amount, para instruções de deslocamento
funct ↳ function. Seleciona variações das operação especificada pelo opcode

Assembly

66

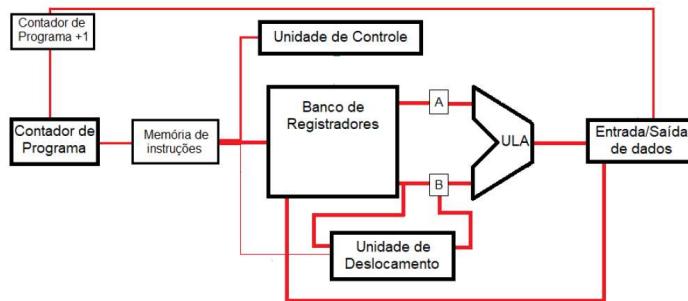
Você, como parte da equipe de desenvolvimento da X, teve uma nova atribuição: foi designado para a função de elaboração dos mnemônicos da linguagem de montagem do processador que está sendo desenvolvido pela empresa. Nessa etapa você será apresentado ao diagrama do processador e aos tipos de instrução que ele irá suportar.

Baseando-se nessas informações e considerando as otimizações já discutidas em seções anteriores, você deverá criar as instruções que serão utilizadas para o desenvolvimento dos programas em linguagem de montagem e, futuramente, dos compiladores de linguagem de alto nível para a arquitetura do processador da X.

Este problema envolve basicamente todo o conteúdo visto até o momento em arquitetura de computadores.

Vamos aos dados: o processador possui apenas instruções do tipo R e do tipo de desvio incondicional (j). Dessa forma, não será possível ter instruções imediatas. O diagrama do processador apresentado é este:

Diagrama para o desenvolvimento da linguagem:



Inicialmente esta situação-problema pode parecer complexa, porém desenvolver o formato e os mnemônicos da linguagem não é o problema.

Considerando que o hardware está pronto, é possível criar uma tabela com possíveis instruções e com os ajustes a serem feitos na arquitetura, que serão realizados pelos outros idealizadores do projeto. Lembre-se de que buscamos uma arquitetura otimizada.

	Formato da Instrução	Operação Realizada
1	add \$r1, r2	R1 recebe a soma entre o que é armazenado em r1 e r2.
2	mult \$r1,r2	R1 recebe a multiplicação entre o que é armazenado em r1 e r2.
3	or \$r1, \$r2	R1 recebe o "ou" lógico entre os bits que estão armazenados em r1 e r2.
4	and \$r1, \$r2	R1 recebe o "e" lógico entre os bits que estão armazenados em r1 e r2.
5	inv \$r1	r1 recebe o inverso do valor que está armazenado em r1, ou seja, -r1.
6	bmr \$r1,\$r2,\$r3	Se o conteúdo de r1 for igual ao conteúdo de r2, desvia-se o contador de programa para a posição de memória de programa indicada em r3.
7	bqr \$r1,\$r2, \$r3	Se conteúdo de r1 for diferente do conteúdo de r2, desvia-se o contador de programa para a posição de memória de programa indicada em r3.
8	jr \$r1	O contador de programa é levado diretamente ao endereço armazenado em r1.
9	jrl \$r1	O contador de programa é levado diretamente ao endereço armazenado em r1 e salva seu estado atual no registrado \$ra.
10	lw \$rd, \$r3, \$r1	O endereço armazenado em r3 é utilizado para deslocar o endereço armazenado em r1 e encontrar o local exato (na memória de dados) de um dado que é salvo em rd.
11	sw \$rd, \$r3, \$r1	O endereço armazenado em r3 é utilizado para deslocar o endereço armazenado em r1 e encontrar o local exato de um endereço (na memória de dados) onde vai ser salvo o conteúdo de rd.

Vasta e diferente a programação de máquina, certo?



Fonte: <https://gifer.com/en/XIOL9>

Recapitulando

- ✓ Sistemas numéricos;
- ✓ Introdução à instruções de máquinas;
- ✓ Pipeline de Instruções;
- ✓ Introdução à linguagem Assembly.

