

# Introdução à Linguagem Assembly



# ÍNDICE DE CONTEÚDO

**01**

## INTRODUÇÃO

O que é e como funciona a linguagem assembly.

**02**

## CARACTERÍSTICAS

Principais características, Vantagens e Desvantagens.

**03**

## INSTRUÇÕES

Campos de instrução, Opcode e Operandos.

**04**

## IMPLEMENTAÇÃO

Arquitetura MIPS e Aplicação da Cifra de César



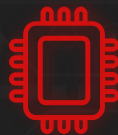
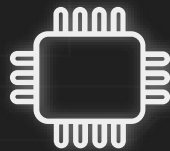


01

# INTRODUÇÃO

# O QUE É A LINGUAGEM ASSEMBLY?

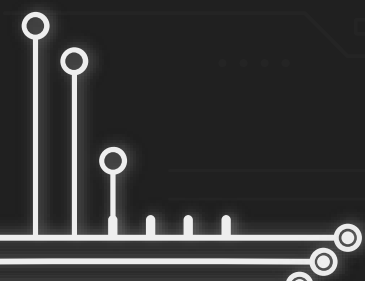

A linguagem Assembly é uma linguagem de baixo nível que é usada para escrever programas que são executados diretamente pela CPU de um computador. Ela é mais próxima da linguagem de máquina que a CPU entende, o que a torna mais eficiente e rápida do que as linguagens de programação de alto nível, mas também mais difícil de aprender e escrever.





# COMO A LINGUAGEM ASSEMBLY FUNCIONA?

Em Assembly, os programas são escritos usando instruções que correspondem diretamente às operações executadas pela CPU. Cada instrução é composta por um código de operação e zero ou mais operandos que especificam os dados a serem manipulados. Os programas em Assembly são geralmente escritos em um editor de texto simples e, em seguida, traduzidos para linguagem de máquina usando um montador.





02

# CARACTERÍSTICAS

# PRINCIPAIS CARACTERÍSTICAS

## BAIXO NÍVEL

É considerada de baixo nível porque se aproxima da linguagem de máquina, que é a linguagem que o processador entende.

## POUCA ABSTRAÇÃO

Por ser uma linguagem de baixo nível oferece pouca abstração em relação ao hardware e às operações do processador.

## TAMANHO COMPACTO

Instruções simples e diretas da linguagem assembly geram um programa mais compacto.

## INSTRUÇÕES SIMPLES

as instruções são relativamente simples e correspondem a operações básicas do processador, como mover dados, realizar operações aritméticas e lógicas, etc.

## CONTROLE PRECISO DO HARDWARE

Permite um controle preciso sobre o hardware do computador.

## DESEMPENHO

Alto desempenho uma vez que as instruções são executadas diretamente pela CPU e há um controle preciso sobre o hardware.

# VANTAGENS E DESVANTAGENS DA LINGUAGEM ASSEMBLY



## VANTAGENS

### Desempenho

Linguagem altamente otimizada

### Controle de hardware

Oferece controle preciso sobre o hardware subjacente

### Acesso direto à memória

Permite que os programadores acessem e manipulem dados diretamente na memória do sistema.



## DESVANTAGENS

### Portabilidade

Linguagem específica para um tipo de arquitetura e não é portavel entre sistemas

### Manutenção

Difícil de fazer alterações no código e corrigir bugs

### Dificuldade de aprendizagem

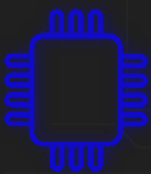
Sintaxe completa e de difícil compreensão





03

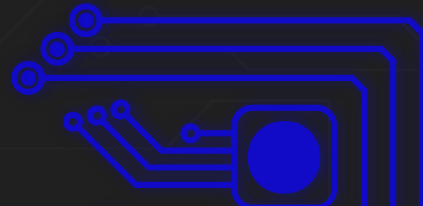
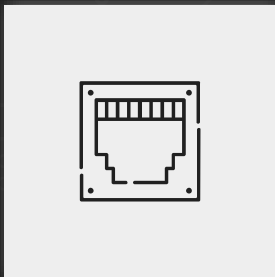
# INSTRUÇÕES



# CAMPOS DE INSTRUÇÃO

As instruções em linguagem de máquina geralmente são compostas por varios campos. Cada campo especifica informações diferentes para o computador. Os dois campos principais são:

- **Campo Opcode**: significa código de operação e especifica a operação que deve ser executada.
  - Cada aplicação tem seu opcode exclusivo.
- **Campo de Operandos**: especificam onde obter os operandos de origem e destino para a operação especificada pelo opcode.
  - A origem/destino dos operandos pode ser uma constante, a memória ou um dos registradores de uso geral.





# INSTRUÇÕES DA LINGUAGEM ASSEMBLY



Add R1, R3, 3

CONSTRUIDO A PARTIR DE  
DUAS ETAPAS

**OPCODE**

O QUE FAZER COM OS  
DADOS (OPERAÇÃO ALU)

Add

R1, R3, 3

**OPERANDOS**

ONDE OBTER DADOS E COLOCAR OS  
RESULTADOS



# TIPOS DE OPCODE

- Opcode de aritmética:  
add, sub, mult, div;
- Opcode de movimentação de dados:  
mov, lea;
- Opcode de controle de fluxo:  
jmp, jne;
- Opcode memória load/store:  
ld, st;

# OPERANDOS

- Cada operando retirado de um determinado modo de endereçamento.

- Exemplos:

Registradores

Imediato

Indireto

Deslocamento

Relativo ao PC

```
add r1, r2, r3
```

```
add r1, r2, 10
```

```
mov r1, (r2)
```

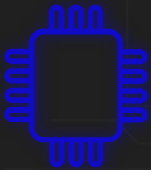
```
mov r1, 10(r3)
```

```
beq 100
```



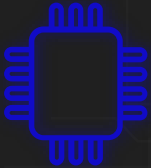
04

# IMPLEMENTAÇÃO



# ARQUITETURA MIPS

- Uma das aplicações da linguagem assembly é na arquitetura MIPS, usada em uma variedade de dispositivos eletrônicos, incluindo roteadores, sistemas de videogame, dispositivos móveis e outros. A linguagem assembly MIPS usa um conjunto de instruções que correspondem diretamente aos conjuntos de instruções do processador MIPS, permitindo que os programadores escrevam códigos de baixo nível que são diretamente executados pelo processador.
- As instruções da linguagem assembly MIPS são divididas em três tipos principais: instruções R, I e J. As instruções R são usadas para operações aritméticas e lógicas em registradores. As instruções I são usadas para operações aritméticas e lógicas em registradores e memória. As instruções J são usadas para operações de salto.

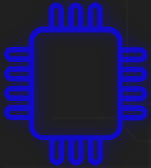


# CIFRA DE CÉSAR E APLICAÇÃO EM ASSEMBLY MIPS

A Cifra de César é uma técnica de criptografia simples que consiste em deslocar cada letra de uma mensagem por um número fixo de posições no alfabeto. O código a seguir pede que o usuário escreva uma mensagem e o deslocamento que ele queira e a aplicação criptografa a mensagem.

```
1 # Aplicação em assembly MIPS para criptografar uma mensagem com a Cifra de César com deslocamento fornecido pelo usuário
2 |
3 .data
4 msg: .space 256 # Espaço para armazenar a mensagem
5 prompt: .asciiz "Digite a mensagem a ser criptografada: "
6 keymsg: .asciiz "Digite o deslocamento da Cifra de César: "
7
8 .text
9 .globl main
10
11 main:
12 # Pede para o usuário digitar a mensagem
13 li $v0, 4 # Código do syscall para imprimir string
14 la $a0, prompt # Endereço da mensagem de prompt
15 syscall
16
17 # Lê a mensagem digitada pelo usuário
18 li $v0, 8 # Código do syscall para ler string
19 la $a0, msg # Endereço do buffer para armazenar a mensagem
20 li $a1, 256 # Tamanho máximo da mensagem
21 syscall
22
23 # Pede para o usuário digitar o deslocamento da Cifra de César
24 li $v0, 4 # Código do syscall para imprimir string
25 la $a0, keymsg # Endereço da mensagem de prompt para o deslocamento
26 syscall
27
28 # Lê o deslocamento da Cifra de César digitado pelo usuário
29 li $v0, 5 # Código do syscall para ler inteiro
30 syscall
```

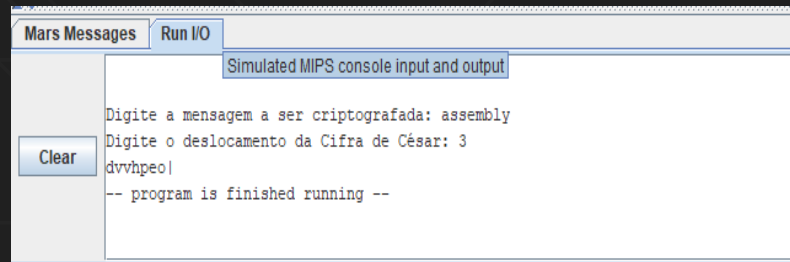




# CIFRA DE CÉSAR E APLICAÇÃO EM ASSEMBLY MIPS

```
30 syscall
31 move $s0, $v0    # Salva o deslocamento em $s0
32
33 # Percorre a mensagem e criptografa cada letra
34 la $t0, msg      # Endereço da primeira letra da mensagem
35 loop:
36 lb $t1, ($t0)    # Carrega a próxima letra da mensagem em $t1
37 beqz $t1, done   # Verifica se chegou ao fim da mensagem
38 add $t2, $t1, $s0 # Adiciona o deslocamento ao valor ASCII da letra
39 sb $t2, ($t0)    # Armazena a letra criptografada de volta na mensagem
40 addi $t0, $t0, 1 # Avança para a próxima letra da mensagem
41 j loop           # Volta para o início do loop
42
43 done:
44 # Imprime a mensagem criptografada
45 li $v0, 4        # Código do syscall para imprimir string
46 la $a0, msg      # Endereço da mensagem criptografada para imprimir
47 syscall
48
49 # Encerra o programa
50 li $v0, 10       # Código do syscall para encerrar o programa
51 syscall
52
```

## Entrada e Saída:



Nesse programa, a mensagem a ser criptografada é armazenada na variável msg e a chave de criptografia (o deslocamento a ser aplicado a cada letra) é armazenada na variável key. O programa usa um loop para percorrer cada letra da mensagem, criptografando-a e armazenando a letra criptografada de volta na mensagem. O resultado final é exibido na tela usando a função syscall com o código 4 para imprimir uma string.



**OBRIGADO!**