





●		

- 1 Introdução



- 1 Introdução
- 2 Registradores RISC-V
- 3 Instruções RISC-V



## Introdução

A linguagem de máquina, ou código de máquina, é a linguagem de programação mais básica que existe no nível do hardware de um computador. É composta por sequências de bits (0s e 1s) que são interpretadas diretamente pelo processador de um computador.

Cada sequência de bits representa instruções específicas que comandam o computador a realizar operações elementares, como somar, subtrair, mover dados e manipular a memória.

Para a comunicação mais direta e eficiente com o hardware, a linguagem Assembly foi criada como uma camada de abstração sobre a linguagem de máquina.

Assembly é essencialmente uma representação legível por humanos do código de máquina, onde cada instrução de Assembly corresponde diretamente a uma instrução em linguagem de máquina.





RISC-V especifica 32 registradores de propósito geral, rotulados de **x0** a **x31**. Cada registrador tem 32 bits em sistemas RV32 (RISC-V 32-bit), 64 bits em sistemas RV64 (RISC-V 64-bit), e 128 bits em sistemas RV128 (RISC-V 128-bit), refletindo a capacidade de processamento de dados do processador.

A função destes registradores inclui:

- **x0:** Sempre contém o valor 0. Qualquer tentativa de escrever neste registrador não terá efeito.
- **x1 - x31:** Usados para várias operações, incluindo mas não limitado a armazenamento temporário de dados, resultados de operações aritméticas, endereços de memória para operações de carga e armazenamento, e para passar argumentos de função e valores de retorno em convenções de chamada de função.

Dentre os registradores de **x1** a **x31**, alguns têm usos especiais em convenções de chamada de função ou pelo sistema operacional:

- **x1 (ra):** Registrador de retorno (return address) para armazenar o endereço de retorno de chamadas de função.

**x2 (sp):** Ponteiro de pilha (stack pointer), usado como o topo da pilha em convenções de chamada.

**x3 (gp):** Ponteiro global (global pointer), usado para acessar dados globais.

**x4 (tp):** Ponteiro de thread (thread pointer), usado para dados específicos do thread.

**x5-x7, x10-x17, x28-x31:** Registradores temporários ou para passagem de argumentos em chamadas de função.

●		

Continuação:

- **x8 (fp/s0):** Frame pointer ou s0, usado em gerenciamento de frames de pilha.
- **x9 (s1):** Registrador salvo, para preservar valores entre chamadas de função.
- **x18-x27 (s2-s11):** Mais registradores salvos para uso geral.



## Registradores RISC-V

Em Assembly RISC-V, os registradores são tipicamente referidos por nomes numéricos (x0 a x31) ou por seus pseudônimos, que indicam usos convencionais específicos. Aqui está uma lista dos registradores em RISC-V com seus labels e usos convencionais:

- **x0 (zero):** Valor constante 0.
- **x1 (ra):** Registrador de retorno de chamada de função.
- **x2 (sp):** Ponteiro de pilha.
- **x3 (gp):** Ponteiro global.
- **x4 (tp):** Ponteiro de thread.
- **x5-x7 (t0-t2):** Temporários, não preservados entre chamadas de função.

●			

Continuação:

- **x8 (s0/fp):** Ponteiro de frame ou salvo, preservado entre chamadas de função.
- **x9 (s1):** Salvo, preservado entre chamadas de função.
- **x10-x11 (a0-a1):** Argumentos de função e valores de retorno para funções.
- **x12-x17 (a2-a7):** Argumentos de função.
- **x18-x27 (s2-s11):** Salvos, preservados entre chamadas de função.
- **x28-x31 (t3-t6):** Temporários, não preservados entre chamadas de função.





●		

Cada formato especifica como os bits da instrução são divididos para representar diferentes partes, como o código de operação (opcode), registradores fonte e destino, constantes imediatas e offsets de endereçamento. Os principais formatos de instrução em RISC-V são:

- Formato R (Tipo Registrador);
- Formato I (Tipo Imediato);
- Formato S (Tipo Store);
- Formato B (Tipo Branch);
- Formato U (Tipo Upper Immediate);
- Formato J (Tipo Jump).



## Formato R (Tipo Registrador)

### Formato R (Tipo Registrador)



Para instruções aritméticas e lógicas que operam em dois registradores fonte e armazenam o resultado em um registrador destino.

Campos:

- **opcode (7 bits):** Especifica o tipo de operação a ser realizada. O opcode determina que a instrução é uma operação aritmética ou lógica entre registradores.
- **rd (5 bits):** Identifica o registrador destino onde o resultado da operação será armazenado.
- **funct3 (3 bits):** Um campo funcional adicional que, junto com o opcode, ajuda a determinar a operação exata a ser realizada.

### Formato R (Tipo Registrador)



Continuação:

- **rs1 (5 bits):** Identifica o primeiro registrador fonte da operação.
- **rs2 (5 bits):** Identifica o segundo registrador fonte da operação.
- **funct7 (7 bits):** Outro campo funcional que fornece informações adicionais para determinar a operação exata, usado em operações aritméticas e lógicas que precisam de mais especificações.

## Formato I (Tipo Immediato)

## Formato R (Tipo Immediato)



Campos:

- **opcode (7 bits):** Define o tipo de operação, indicando que a instrução utiliza um valor imediato em sua execução.
- **rd (5 bits):** O registrador destino onde o resultado da operação será salvo.
- **funct3 (3 bits):** Auxilia na determinação da operação específica a ser realizada, junto ao opcode.
- **rs1 (5 bits):** O registrador fonte da operação.
- **imm (12 bits):** O valor imediato utilizado na operação. Este valor pode ser usado para adições, comparações ou como um offset para operações de carga e armazenamento.

## Formato S (Tipo Store)

## Formato R (Tipo Store)



Campos:

- **opcode (7 bits):** Indica que a instrução é uma operação de armazenamento de dados na memória.
- **imm (12 bits, dividido):** O valor imediato que especifica o offset de endereçamento na memória a partir do endereço base fornecido pelo rs1. Dividido entre os campos superior e inferior para codificação.
- **funct3 (3 bits):** Determina o tamanho da unidade de dados a ser armazenada (por exemplo, byte, halfword, word).
- **rs1 (5 bits):** O registrador que contém o endereço base para o armazenamento de dados.
- **rs2 (5 bits):** O registrador que contém os dados a serem armazenados na memória.

## Formato B (Tipo Branch)



## Formato R (Tipo Branch)



Para instruções de desvio condicional baseadas na comparação de dois registradores.

Campos:

- **opcode (7 bits):** Especifica que a instrução é um desvio condicional.
- **imm (12 bits, dividido):** O valor imediato que representa o offset de desvio caso a condição seja verdadeira. Este valor é dividido e codificado em vários campos.
- **funct3 (3 bits):** Define a condição de desvio (por exemplo, igualdade, menor que).
- **rs1 e rs2 (5 bits cada):** Os registradores cujos valores serão comparados para determinar se o desvio será tomado.



### Formato R (Tipo Upper Immediate)



Para carregar um valor imediato de 20 bits no registrador destino, usado principalmente para operações de alto nível e configuração de endereços.

Campo:

- **opcode (7 bits):** Indica que a instrução carrega um valor imediato de 20 bits no registrador destino.
- **rd (5 bits):** O registrador destino para o valor imediato.
- **imm (20 bits):** O valor imediato de 20 bits que é carregado na parte superior do registrador destino, geralmente usado para formar endereços ou constantes grandes.

## Formato R (Tipo Jump)



Campo:

- **opcode (7 bits):** Denota uma instrução de salto incondicional.
- **rd (5 bits):** O registrador onde o endereço de retorno (normalmente o endereço da instrução seguinte) será armazenado, geralmente x1 para chamadas de função.
- **imm (20 bits):** O valor imediato que representa o offset de salto a partir do endereço atual. Esse valor é utilizado para calcular o endereço de destino do salto.



# Organização de um Programa em Assembly RISC-V

## Organização de um Programa em Assembly RISC-V



Embora a estrutura exata possa variar conforme as necessidades específicas do projeto e as preferências do desenvolvedor, uma organização típica inclui várias seções:

- Seção de dados;
- Seção de texto;
- Declaração do ponto de entrada;
- Instruções.



●		

A seção de dados é usada para declarar variáveis estáticas e inicializar constantes que seu programa irá usar.

Variáveis e constantes são alocadas aqui com nomes simbólicos, facilitando o acesso no restante do programa.

.data

```
msg: .ascii "Ola, mundo!\n" #Declara variavel do tipo string
numero: .word 123 #Declara variavel do tipo inteiro
```

## Organização de um Programa em Assembly RISC-V: Seção de Dados



Esta seção começa com a diretiva **.text** e geralmente inclui uma etiqueta (label) para o ponto de entrada do programa, frequentemente denominado `main` em programas simples.

```
.text
li a7 4 #Carrega codigo de chamada de sistema PrintString
la a0 msg #Carrega a string armazenada no endereco de msg
ecall #Realiza a chamada de sistema
```

# Assembly RISC-V

## Organização de um Programa em Assembly RISC-V: Declaração do Ponto de Entrada



O ponto de entrada do programa é onde a execução começa. Em muitos programas Assembly RISC-V, isso é explicitamente marcado com um label **main**:. Esse ponto de entrada é frequentemente declarado como global para que o *linker* possa identificá-lo.

## Código

```
.text
.global main

main:
    li a7, 4 #Carrega código de chamada de sistema PrintString
    la a0, msg #Carrega a string armazenada no endereço de msg
    ecall #Realiza a chamada de sistema
```

## Organização de um Programa em Assembly RISC-V: Instruções



Isso inclui operações lógicas e aritméticas, controle de fluxo, chamadas de função e manipulação de dados.

## Código

```
main:
    li a7 4 #Carrega codigo de chamada de sistema PrintString
    la a0 msg #Carrega a string armazenada no endereco de msg
    ecall #Realiza a chamada de sistema
```

# Tipos de Datos Básicos

## Tipos de Datos Básicos



- Inteiros;
- Números de ponto flutuante;
- Caracteres e Strings;
- Vetores (Arrays).







Para caracteres e strings, você pode usar **.byte** para um único caractere e **.ascii** ou **.string** para strings (sequências de caracteres terminadas em null).

```
.data
    varChar: .byte "A"
    varString: .ascii "Variavel String"
```

●		

Arrays podem ser declarados especificando o tipo de dado e a quantidade. Para um array de inteiros, você repetiria a diretiva de tipo o número necessário de vezes ou usaria uma diretiva para reservar um bloco de espaço diretamente.

```
.data
    arrayInt: .word 1, 2, 3, 4
```

Para um array de tamanho fixo, mas sem inicializar os valores, você pode usar **.space** para reservar uma quantidade específica de espaço em bytes.

```
.data
    array: .space
```

## Carregamento e Armazenamento de Dados

## Carregamento e Armazenamento de Dados



Os valores precisam ser carregados primeiramente nos registradores para que, em seguida, as instruções possam ser executadas.

## Carregamento e Armazenamento de Dados



Aqui estão as instruções básicas de carregamento de dados em Assembly RISC-V, juntamente com uma breve descrição do que cada uma faz:

- **Inteiros:**

- **lb:** Carrega um byte da memória para o registrador, com extensão de sinal.
- **lh:** Carrega uma halfword (2 bytes) da memória para o registrador, com extensão de sinal.
- **lw:** Carrega uma word (4 bytes) da memória para o registrador.
- **lbu:** Carrega um byte da memória para o registrador, sem extensão de sinal (zero-extend).
- **lhu:** Carrega uma halfword (2 bytes) da memória para o registrador, sem extensão de sinal (zero-extend).
- **ld:** (somente RV64I): Carrega uma doubleword (8 bytes) da memória para o registrador.

## Carregamento e Armazenamento de Dados



- **Números de Pontos Flutuantes:**

- **Endereços de Memória:**

- **la:** Carrega o valor de um endereço em um registrador.

## Carregamento e Armazenamento de Dados



Aqui estão as instruções básicas de armazenamento de dados em Assembly RISC-V, juntamente com uma breve descrição do que cada uma faz:

- **Inteiros:**

- **sb:** Store Byte - Armazena o byte menos significativo de um registrador na memória.
- **sh:** Store Halfword - Armazena os dois bytes menos significativos de um registrador na memória.
- **sw:** Store Word - Armazena os quatro bytes de um registrador na memória.
- **sd:** Store Doubleword - Armazena os oito bytes de um registrador na memória (apenas em RV64I).

- **Endereços de Memória:**

- **Números de Ponto Flutuante:**

- **fsw:** Store Word Floating-Point - Armazena um valor de ponto flutuante de precisão simples de um registrador de ponto flutuante na memória.

- **fsd**: Store Doubleword Floating-Point - Armazena um valor de ponto flutuante de precisão dupla de um registrador de ponto flutuante na memória.





## Chamadas de Sistema



O procedimento exato para realizar uma chamada de sistema varia dependendo do sistema operacional (por exemplo, Linux), mas o conceito básico é similar entre diferentes ambientes.

## Chamadas de Sistema



1 **Definir o Número da Chamada de Sistema:** Cada chamada de sistema tem um número único associado. Esse número deve ser colocado em um registrador específico. Em sistemas Linux sobre RISC-V, o número da chamada de sistema é colocado no registrador **a7**.

**2 Configurar Argumentos:** Se a chamada de sistema requer argumentos, eles devem ser colocados nos registradores **a0** a **a6**, conforme a convenção de chamadas do sistema operacional. Por exemplo, **a0** pode ser usado para um descritor de arquivo, **a1** para o endereço de um buffer de dados, e assim por diante.

## Chamadas de Sistema



- Executar a Instrução *ecall*:** A instrução *ecall* é usada para invocar a chamada de sistema. O kernel do sistema operacional lê o número da chamada de sistema do registrador **a7**, interpreta os argumentos dos registradores **a0** a **a6**, e executa a operação solicitada.
- Obter o Resultado:** Após a execução da chamada de sistema, o resultado (se houver) é geralmente retornado no registrador **a0**. Se a chamada de sistema falhar, o valor retornado pode indicar o tipo de erro ocorrido.





- **Operações com Inteiros:**

- **add:** Soma dois registradores.
- **sub:** Subtrai um registrador de outro.
- **mul:** Multiplica dois registradores.
- **div:** Divide um registrador por outro, resultando no quociente.
- **rem:** Divide um registrador por outro, resultando no resto.
- **addi:** Soma um registrador com um valor imediato.



Continuação:

- **Operações com Números de Ponto Flutuante:**
  - **Precisão Simples (Extensão "F"):**
    - **fadd.s:** Soma de ponto flutuante de precisão simples.
    - **fsub.s:** Subtração de ponto flutuante de precisão simples.
    - **fmul.s:** Multiplicação de ponto flutuante de precisão simples.
    - **fdiv.s:** Divisão de ponto flutuante de precisão simples.
    - **fsqrt.s:** Raiz quadrada de ponto flutuante de precisão simples.
    - **fmin.s:** Mínimo de dois números de ponto flutuante de precisão simples.
    - **fmax.s:** Máximo de dois números de ponto flutuante de precisão simples.



Continuação:

- **Operações com Números de Ponto Flutuante:**
  - **Precisão Simples (Extensão "D"):**
    - **fadd.d:** Soma de ponto flutuante de precisão dupla.
    - **fsub.d:** Subtração de ponto flutuante de precisão dupla.
    - **fmul.d:** Multiplicação de ponto flutuante de precisão dupla.
    - **fdiv.d:** Divisão de ponto flutuante de precisão dupla.
    - **fsqrt.d:** Raiz quadrada de ponto flutuante de precisão dupla.
    - **fmin.d:** Mínimo de dois números de ponto flutuante de precisão dupla.
    - **fmax.d:** Máximo de dois números de ponto flutuante de precisão dupla.



## Exemplos 1

# Assembly RISC-V

## Exemplos 1



### Somando dois números

```
.data
msg_1: .asciz "Informe o primeiro numero: "
msg_2: .asciz "Informe o segundo numero: "
resultado: .asciz "Resultado: "
.text
.global main
main:
    li a7 4
    la a0 msg_1
    ecall #Imprime mensagem para primeiro numero
    li a7 5
    ecall #Leitura do teclado do segundo numero
    mv t0 a0 #Copia o valor de a0 e t0
    li a7 4
    la a0 msg_1
    ecall #Imprime mensagem para segundo numero
    li a7 5
    ecall #Leitura do teclado do segundo numero
    mv t1 a0 #Copia o valor de a0 e t1
    ...
```

# Assembly RISC-V

## Exemplos 1



### Somando dois números

```
...  
li a7 4  
la a0 resultado  
ecall #Imprime mensagem de resultado  
li a7 1  
mv a0 t2 #Copia o valor de t2 para a0  
ecall #Imprime um inteiro
```

## Desvios Condicionais

## Desvios Condicionais



- **Operações com Inteiros:**

- **beq:** Branch if Equal - Desvia se dois registradores são iguais.
- **bne:** Branch if Not Equal - Desvia se dois registradores não são iguais.
- **blt:** Branch if Less Than - Desvia se o primeiro registrador é menor que o segundo (sinalizado).
- **bge:** Branch if Greater Than or Equal - Desvia se o primeiro registrador é maior ou igual ao segundo (sinalizado).
- **bltu:** Branch if Less Than Unsigned - Desvia se o primeiro registrador é menor que o segundo (não sinalizado).
- **bgeu:** Branch if Greater Than or Equal Unsigned - Desvia se o primeiro registrador é maior ou igual ao segundo (não sinalizado).

## Desvios Condicionais



- **Operações com Números de Ponto Flutuante:**

- **feq.s:** Floating-point Equal Single-precision - Compara se dois números de ponto flutuante de precisão simples são iguais.
- **flt.s:** Floating-point Less Than Single-precision - Compara se um número de ponto flutuante de precisão simples é menor que outro.
- **fle.s:** Floating-point Less than or Equal Single-precision - Compara se um número de ponto flutuante de precisão simples é menor ou igual a outro.

## Desvios Condicionais



- **Operações com Números de Ponto Flutuante:**

- **feq.s:** Floating-point Equal Single-precision - Compara se dois números de ponto flutuante de precisão simples são iguais.
- **flt.s:** Floating-point Less Than Single-precision - Compara se um número de ponto flutuante de precisão simples é menor que outro.
- **fle.s:** Floating-point Less than or Equal Single-precision - Compara se um número de ponto flutuante de precisão simples é menor ou igual a outro.

## Desvios Incondicionais







# Assembly RISC-V

## Exemplos 2



### Comparando números

```
.data
msg_1: .asciz "Informe o primeiro numero: "
msg_2: .asciz "Informe o segundo numero: "
resultado_1: .asciz " e maior que "
resultado_2: .asciz " e igual a "
resultado_3: .asciz " e menor que "
.text
.global main

main:

    li a7 4
    la a0 msg_1
    ecall

    li a7 5
    ecall
    mv t0 a0

    li a7 4
    la a0 msg_1
    ecall
```

## 64



## Exemplos 2



## Exercícios 1



- Implemente um programa que leia dois números e um código de operação (1 para adição, 2 para subtração, 3 para multiplicação, 4 para divisão). Execute a operação correspondente e mostre o resultado. Implemente verificações para prevenir a divisão por zero.
- Escreva um programa que, dado um número que representa a idade de uma pessoa, identifique sua faixa etária: criança (0-12), adolescente (13-18), adulto (19-59) ou idoso (60+). Utilize desvios condicionais para direcionar para o bloco de código apropriado.
- Desenvolva um programa que determine o maior de três números. Se todos os números forem iguais, armazene um valor específico que indique essa igualdade.



## Exercícios 1



- Crie um programa que simula um controle de direção básico com quatro direções (Norte=0, Sul=1, Leste=2, Oeste=3). Para uma entrada de direção, o programa deve indicar a ação correspondente (por exemplo, "Mover para o norte").
- Implemente um programa que verifique se um número fornecido é uma potência de dois. O programa deve armazenar 1 em um registrador se o número for uma potência de dois, e 0 se não for.
- Escreva um programa que calcule o discriminante ( $b^2-4ac$ ) de uma equação quadrática e indique se as raízes são reais e distintas, reais e iguais, ou complexas.

# Assembly RISC-V

## Exercícios 1



- Desenvolva um programa que converta a temperatura de Celsius para Fahrenheit e vice-versa, baseado em um código de operação. Inclua verificações para entradas válidas.
- Implemente um programa que teste se um número  $n$  é divisível por outro número  $m$ , sem deixar resto. Utilize operações aritméticas para encontrar o resto da divisão.
- Crie um programa que classifique um caractere (valor ASCII) como letra maiúscula, letra minúscula, dígito ou símbolo especial. Utilize a tabela ASCII e desvios condicionais para a classificação.



