"Assim que começamos a programar, nos surpreendemos que não era tão fácil quanto pensamos ... eu lembro o exato instante que percebi que grande parte da minha vida dali em diante seria gasta encontrando problemas nos meus programas" (Maurice Wilkes, 1949).

Revisão Conjuntos de Instruções RISC-V

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Instruções de Máquina

Para nos comunicar com o processador precisamos “falar a sua língua”.

Alguns exemplos:

O seu computador pessoal.

x86-64, ARM.

Smartphone, Smartwatches, microcontroladores, ...

ARM, RISC-V, MIPS, PIC instruction set, AVL instruction set, ...

Instruções de Máquina

O Conjunto de instruções está diretamente relacionado com o hardware.

- Como o hardware interpreta as instruções.
- O quão complexa é a interpretação.
- A quantidade de instruções disponíveis.
- Como as instruções são armazenadas e requisitadas da memória.
- ...

RISC-V

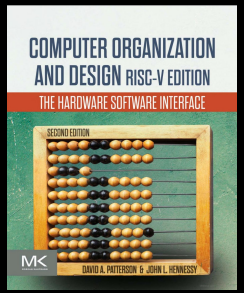
Abordaremos o conjunto de instruções e implementação do RISC-V.

Discutido em Patterson e Henessy (2020).

Conjunto de instruções Livre.

Pode ser usado por qualquer um, para qualquer finalidade.

Patterson, Hennessy. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. 2020.



Registradores

A vasta maioria das arquiteturas atuais (x86-64, MIPS, ARM) operam somente na CPU.

Precisamos carregar os dados para os registradores da CPU.

Porções de memória na CPU as quais podemos utilizar para realizar operações.

Os registradores são visíveis ao programador.

Ao menos quando programamos em baixo nível.

Existem registradores não visíveis, mas não trataremos deles agora.

Registradores

Registradores são os dispositivos de memória mais rápidos disponíveis no computador.

Enquanto temos uma abundância relativa de memória principal, os registradores são escassos.

Em RISC-V, por exemplo, temos 32 registradores de 32 ou 64 bits cada.

Na disciplina, vamos assumir 32 bits.

Curiosidade : quantos registradores seu processador x86 possui?

Pesquise.

Registradores

Cada registrador precisa ter um endereço.

Quantos bits são necessários para endereçar todos os 32 registradores do RISC-V?

Registradores

São necessários 5 bits.

Registrador	# Decimal	# Binário	Descrição
x0	0	00000	Constante Zero.
x1 (ra)	1	00001	Endereço de Retorno.
x2 (sp)	2	00010	Stack Pointer.
x3 (gp)	3	00011	Global Pointer.
x4 (tp)	4	00100	Thread Pointer.
x5-x7	[5..7]	[00101..00111]	Temporários (para realizar operações).
x8-x9	[8..9]	[01000..01001]	Salvos (para realizar operações).
x10-x17	[10-17]	[01010..10001]	Argumentos e resultados.
x18-x27	[18-27]	[10010..11011]	Salvos (para realizar operações).
x28-x31	[28-31]	[11100..11111]	Temporários (para realizar operações).

Assembly

A máquina entende somente zeros e uns.

Difícil para nós humanos interpretarmos que o valor 10001_2 em uma instrução faz referência ao registrador x17.

Por essa razão programamos em Linguagem de Montagem – **Assembly.**

Nos referenciamos aos registradores (e operações) por seus nomes.

Chamamos de **mnemônicos**.

Exemplo: o registrador x17 é o registrador 10001_2, ou 17_{10}.

O montador (Assembler) traduz de x17 para 10000_2 em linguagem de máquina.

Word Size

Um dado de tamanho padrão que um processador lida é denominado palavra (**word**).

O tamanho da palavra é denominado ***word size***.

O tamanho da palavra (*word*) do RISC-V que vamos usar é de 32 bits (obs.: existem versões de 64 bits).

- Os registradores suportam 32 bits, e as operações geralmente lidam com 32 bits.
- Resultados são de 32 bits.
- Acessos à memória comumente acessam 32 bits.
- ...

Word Size

Um dado de tamanho padrão que um processador lida é denominado palavra (**word**).

O tamanho da palavra é denominado ***word size***.

O tamanho da palavra (*word*) do RISC-V que vamos usar é de 32 bits (obs.: existem versões de 64 bits).

- Os registradores suportam 32 bits, e as operações geralmente lidam com 32 bits.
- Resultados são de 32 bits.
- Acessos à memória comumente acessam 32 bits.
- ...

Processadores diferentes possuem palavras de tamanhos diferentes.

- x86-64 e ARMs utilizados em smartphones atuais possuem palavras de 64 bits.
- Os PICs da família 16F62x possuem palavra de 8 bits.

Instruções

Todas instruções no RISC-V ocupam 32 bits.

A consistência facilita o projeto.

O x86-64, por exemplo, possui instruções de tamanhos variados.

Flexível, mas o hardware se torna muito mais complexo (e muitas vezes lento).

Instruções

Um exemplo de instrução no RISC-V:

```
00000001010110100000010010110011
```

32 bits

Instruções

Um exemplo de instrução no RISC-V:

```
00000001010110100000010010110011
```

32 bits

```
add x9, x20, x21
```

Como vamos escrever em assembly.

```
x9 = x20 + x21
```

O que a operação faz.

Instruções

Um exemplo de instrução no RISC-V:

```
00000001010110100000010010110011
```

```
add x9, x20, x21
```

Em assembly, utilizamos **mnemônicos** ao invés dos bits diretamente para representar uma instrução.

Instruções

Um exemplo de instrução no RISC-V:

```
00000001010110100000010010110011
```

```
add x9, x20, x21
```

O **montador** (***assembler***) traduz diretamente de Assembly para a Linguagem de Máquina, e vice-versa.

Em assembly, utilizamos **mnemônicos** ao invés dos bits diretamente para representar uma instrução.

Instruções do Tipo R

Uma soma (add) é uma instrução do Tipo-R. No RISC-V. Formato (em linguagem de máquina):

opcode: Código da operação que precisa ser realizada.
rd: Registrador de destino
funct3: Adicional do opcode.
rs1: Registrador fonte 1.
rs2: Registrador fonte 2.
funct7: Adicional do opcode.

Exemplos

Instrução	Tipo	funct7	rs2	rs1	funct3	rd	opcode
`add x9, x20, x21`	R	0000000	10101	10100	000	01001	0110011
`sub x8, x21, x22`	R	0100000	10110	10101	000	01000	0110011

Faça você mesmo

Utilize o cartão de conjuntos de instruções disponibilizado na UFPRVirtual como fica a seguinte instrução.

Instrução	Tipo	funct7	rs2	rs1	funct3	rd	opcode
`add x9, x20, x21`	R	0000000	10101	10100	000	01001	0110011
`sub x8, x21, x22`	R	0100000	10110	10101	000	01000	0110011
`or x5, x18, x19`	R						

Resposta

Instrução	Tipo	funct7	rs2	rs1	funct3	rd	opcode
`add x9, x20, x21`	R	0000000	10101	10100	000	01001	0110011
`sub x8, x21, x22`	R	0100000	10110	10101	000	01000	0110011
`or x5, x18, x19`	R	0000000	10011	10010	100	00101	0110011

Imediatos

Valores constantes são chamados de **imediatos**.

Por exemplo:

```
addi x7,x0,1545
```

Tipo-I

Instruções do Tipo-I.

Até 12 bits para imediatos.
Valores em complemento de 2.

Exemplos

Instrução	Tipo	funct7	rs2	rs1	funct3	rd	opcode
`add  x9, x20, x21`	R	0000000	10101	10100	000	01001	0110011
`sub  x8, x21, x22`	R	0100000	10110	10101	000	01000	0110011

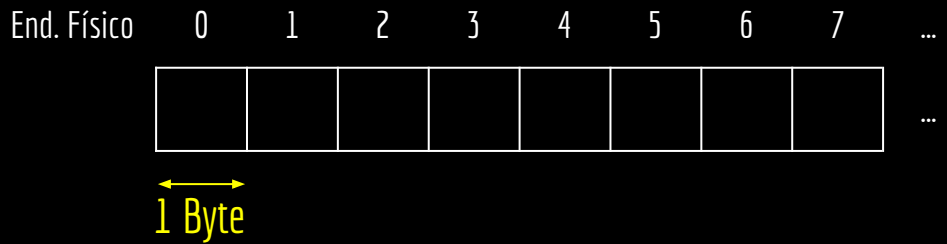
Instrução	Tipo	imediato	rs1	funct3	rd	opcode
`addi x7,x0,1545`	I	0110 0000 1001	00000	000	00111	0010011

Lidando com a memória

A memória principal é um **vetor**, onde cada posição possui um **endereço físico**.

As memórias são comumente endereçadas a Byte.

Cada endereço suporta exatamente 1 Byte.

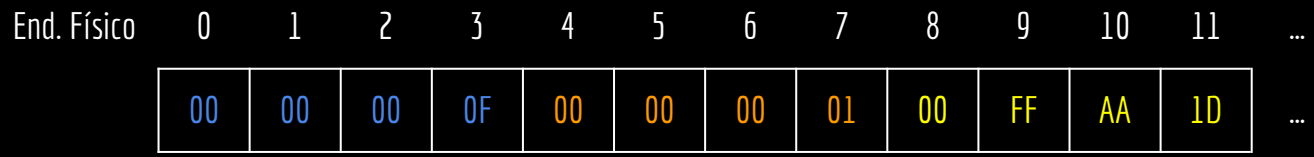


Considere o exemplo em C

```
int x = 0x0F;

int v[2] = {0x01,0x00FFAA1D};
```

Considerando inteiros de 32 bits.

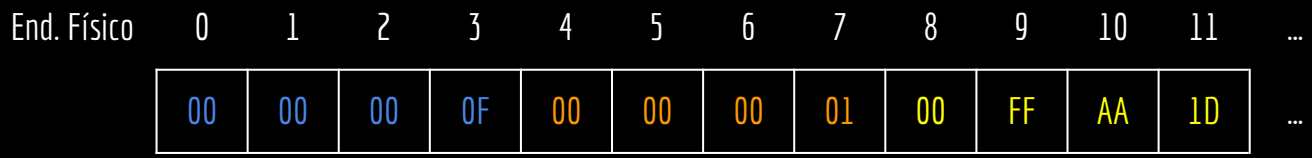


Considere o exemplo em C

```
int x = 0x0F;

int v[2] = {0x01,0x00FFAA1D};
```

Um vetor é algo que inicia em uma posição de memória. Cada nova posição é um deslocamento da posição inicial.



Considere o exemplo em C

v começa no endereço 4.

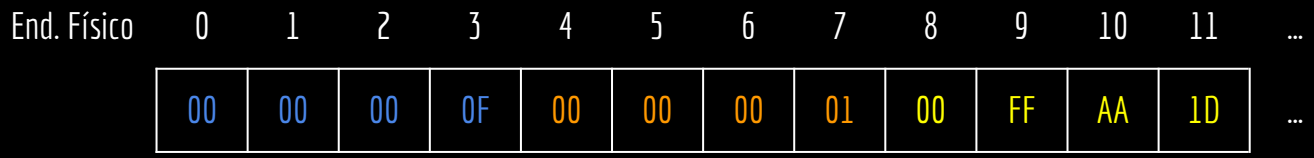
Ponteiro com endereço base aponta para 4.

v[0] é o mesmo que v deslocado 0 endereços (4+0).

v[1] é o mesmo que v deslocado 4 endereços de (4+4).

Deslocamos 4, pois cada inteiro ocupa 4 bytes no exemplo.

Os deslocamentos mudariam dependendo do tipo da variável. Exemplo: 1 byte para chars



lw

lw - Load Word.

```
lw x9, 4(x22)
```

`x9 = *(x22+4)` //carregue o valor de 4 bytes presente no endereço apontado por x22 somado com 4.

lw

Endereço

Deslocamento (armazenado em complemento de 2).

```
lw x9, 4(x22)
```

```
x9 = *(x22+4) //carregue o valor presente no endereço apontado por x22 somado com 4
```

Exemplo

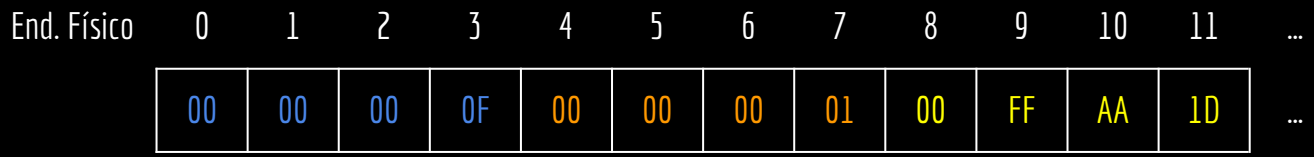
```
int x = 0x0F;

int v[2] = {0x01,0x00FFAA1D};
```

```
addi x22, x0, 4 # x22 aponta para o início do vetor

lw x8, 0(x22)   #x8 = v[0]

lw x9, 4(x22)   #x9 = v[1]
```



Exemplos

Instrução	Tipo	funct7	rs2	rs1	funct3	rd	opcode
`add x9, x20, x21`	R	0000000	10101	10100	000	01001	0110011
`sub x8, x21, x22`	R	0100000	10110	10101	000	01000	0110011

Instrução	Tipo	imediato	rs1	funct3	rd	opcode
`addi x7,x0,1545`	I	0110 0000 1001	00000	000	00111	0010011
`lw x8, 4(x22)`	I	0000 0000 0100	10110	010	01000	0000011

SW

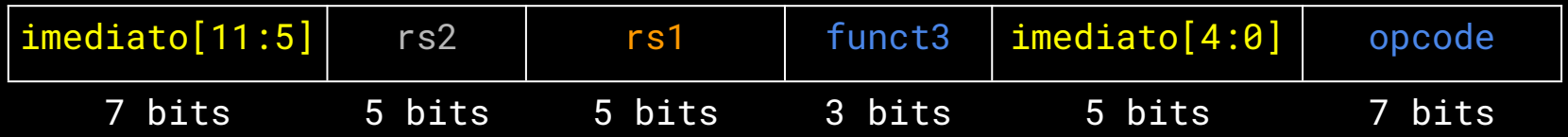
sw - Store Word

```
sw x9, 4(x22)
```

`*(x22+4) = x9`//armazene os bytes de x9 no endereço apontado por x22 somado com 4.

Tipo-S

Stores são instruções do Tipo-S.



Instrução não tem registrador de destino.
Imediato quebrado em duas partes.

Faça você mesmo

Considere que a variável *h* está armazenada no endereço 0x4 de memória, e que o vetor A começa no endereço 0xF. Traduza o seguinte trecho programa C para Assembly do RISC-V.

```
A[30] = h + A[30] + 1
```

Resposta

Considere que a variável *h* está armazenada no endereço 0x4 de memória, e que o vetor A começa no endereço 0xF. Traduza o seguinte trecho programa C para Assembly do RISC-V.

A[30] = h + A[30] + 1

```
addi x18, x0, 0x4       #endereço de h
lw x18, 0(x18)
addi x19, x0, 0xF       #base do vetor
lw x5, 120(x19)
add x5, x5, x18         #A[30] + h
addi x5, x5, 1          #A[30] + h + 1
sw x5, 120(x19)
```

Exemplos

Instrução	Tipo	funct7	rs2	rs1	funct3	rd	opcode
`add x9, x20, x21`	R	0000000	10101	10100	000	01001	0110011
`sub x8, x21, x22`	R	0100000	10110	10101	000	01000	0110011

Instrução	Tipo	imediato	rs1	funct3	rd	opcode
`addi x7,x0,1545`	I	0110 0000 1001	00000	000	00111	0010011
`lw x8, 4(x22)`	I	0000 0000 0100	10110	010	01000	0000011

Instrução	Tipo	imed [11:5]	rs2	rs1	funct3	imed [4:0]	opcode
`sw x5, 120(x19)`	S	0000011	00101	10011	010	11000	0100011

Imediatos Grandes

Suponha que precisamos carregar o seguinte valor para um registrador:

3998977

Como fazer? Qual o problema?

Imediatos Grandes

Suponha que precisamos carregar o seguinte valor para um registrador:

3998977

Como fazer? Qual o problema?

O valor é maior que 2^{11}. Não cabe nos 12 bits do campo de imediato das instruções do Tipo-I.

lui

lui - Load upper immediate.

Carregue um imediato de 20 bits para os 20 bits mais altos do registrador, e preencha o resto com zero.

Exemplo:

O valor 3998977_{10} é, em hexa, $003D0501_{16}$.

lui x5, 0x003D0
addi x5, 0x501

lui

lui - Load upper immediate.

Carregue um imediato de 20 bits para os 20 bits mais altos do registrador, e preencha o resto com zero.

Exemplo:

O valor 3998977_{10} é, em hexa, $003D0501_{16}$.

```
lui x5, 0x003D0
addi x5, 0x501
```

x5	003D05000

lui

lui - Load upper immediate.

Carregue um imediato de 20 bits para os 20 bits mais altos do registrador, e preencha o resto com zero.

Exemplo:

O valor 3998977_{10} é, em hexa, $003D0501_{16}$.

```
lui x5, 0x003D0
addi x5, 0x501
```

x5	003D05501

Tipo-U

lui é uma instrução do Tipo-U.

Exemplos

Instrução	Tipo	funct7	rs2	rs1	funct3	rd	opcode
`add x9, x20, x21`	R	0000000	10101	10100	000	01001	0110011
`sub x8, x21, x22`	R	0100000	10110	10101	000	01000	0110011

Instrução	Tipo	imediato	rs1	funct3	rd	opcode
`addi x7,x0,1545`	I	0110 0000 1001	00000	000	00111	0010011
`lw x8, 4(x22)`	I	0000 0000 0100	10110	010	01000	0000011

Instrução	Tipo	imed [11:5]	rs2	rs1	funct3	imed [4:0]	opcode
`sw x5, 120(x19)`	S	0000011	00101	10011	010	11000	0100011

Instrução	Tipo	imediato	rd	opcode
`lui x5, 0x003D0`	U	0000 0000 0011 1101 0000	00101	0110111

Venus

Simulador Venus:

https://venus.cs61c.org

Use o seguinte formato para seus programas.

```
# Faça seu programa Aqui
# Sempre mantenha no final o seguinte
# Vai chamar o S.O. com código 0 para finalizar o programa
addi a0, x0, 17
addi a1, x0, 0
ecall
```

Saída

O Venus (e outros simuladores) incluem um simulador de sistema operacional para entrada e saída.

Precisa carregar o valor da chamada para o registrador correto, e realizar a chamada via instrução `ecall`.

Veja como fazer em: https://github.com/61c-teach/venus/wiki/Environmental-Calls

Exemplo

```
addi a1, x0, 16
addi a0, x0, 1  # print_int ecall
ecall

addi a0, x0, 17
addi a1, x0, 0
ecall
```

Exercícios

1. Escreva nas **primeiras páginas do seu caderno** todas as instruções básicas do RISC-V, juntamente com a extensão RV32M para multiplicação (serão 47 instruções ao todo). Escreva a instrução, o formato, um exemplo, e descreva para que serve. Faça também uma tabela com os registradores e seus usos. Exemplo:
   addi reg1, reg2, IMEDIATO Instrução do Tipo I. Soma o valor do imediato, que pode possuir sinal, com reg2, e salve o resultado em reg1. O IMEDIATO pode ser qualquer valor no intervalo $[-2^{11} .. (+2^{11} -1)]$.
2. Considerando instruções do Tipo-I e Tipo-S, qual o maior e menor imediatos que podemos utilizar?
3. Estude sobre operações lógicas (or, and, not, ...) e shifts no RISC-V. Qual a utilidade dessas operações?
4. Na aula, o seguinte exemplo é usado para carregar a constante 0x003D0501
   lui x5, 0x003D0
   addi x5, 0x501

   O programa seguinte faz o mesmo? Por quê?
   lui x5, 0x003D0
   ori x5, 0x501

Exercícios

5. Considere o seguinte programa (valores em hexa para facilitar a leitura) em código de máquina. Quais são as instruções assembly RISC-V? De que tipo são essas instruções? O que o programa faz?

```
0x00B00293
0x01300313
0x00530333
0x00135313
```

6. Faça um programa no Venus que soma dois valores inteiros, e exibe a média desses valores.

7.Carregue os seguintes imediatos para algum registrador do RISC-V. Não utilize pseudo instruções.

A. 255_{10}
B. 987342343_{10}
C. -987342343_{10} <- Dica: utilize complemento de 2

Exercícios

8. Considere um vetor de inteiros *vet* que começa no endereço 0x100080AA. Realize a seguinte operação em assembly do RISC-V (considere que inteiros ocupam 4 bytes):

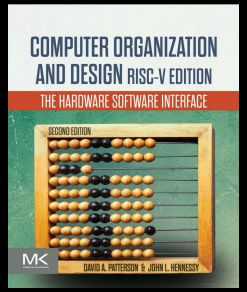
vet[7] = vet[1]+vet[2]+vet[3]+65

9. Mostre o código de máquina para as instruções dos exercícios 6, 7 e 8. Encaixe os bits no formato de cada uma das instruções, como feito em sala.

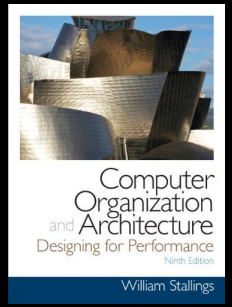
10. “Com a miniaturização cada vez maior dos transistores, veremos CPUs com mais registradores no futuro”. Confirme ou refute essa afirmação. Justifique.

Referências

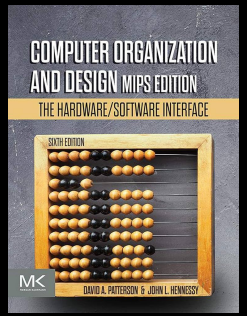
Patterson, Hennessy. Computer Organization and Design RISC-V Edition: The Hardware Software Interface. 2020.



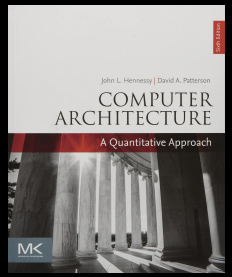
Stallings, W. Organização de Arquitetura de Computadores. 10a Ed. 2016.



Patterson, Hennessy. Computer Organization and Design MIPS Edition: The Hardware/Software Interface. 2020.



Hennessy, Patterson. Arquitetura de Computadores: uma abordagem quantitativa. 2019.



Licença