

Codificação das instruções RV32

Cada instrução é codificada com 32 *bits*, em um dos seguintes formatos

	31	27	26	25	24	20	19	15	14	12	11	7	6	0		
R	funct7				rs2			rs1			funct3		rd		opcode	
I	imm[11:0]						rs1			funct3		rd		opcode		
S	imm[11:5]				rs2			rs1			funct3		imm[4:0]		opcode	
B	imm[12 10:5]				rs2			rs1			funct3		imm[4:1 11]		opcode	
U	imm[31:12]										rd		opcode			
J	imm[20 10:1 11 19:12]										rd		opcode			

R: sll, srl, sra, add, sub, xor, or, and, slt, sltu

I: slli, srli, srai, addi, xori, ori, andi, slti, sltiu, jalr, lw, lh, lb

U: lui, auipc

B: beq, bne, blt, bge, bltu, bgeu

J: jal

S: sw, sh, sb

Codificação das instruções RV32

Cada instrução é codificada com 32 *bits*, em um dos seguintes formatos

	31	27	26	25	24	20	19	15	14	12	11	7	6	0			
R	funct7					rs2			rs1			funct3		rd		opcode	
I	imm[11:0]								rs1			funct3		rd		opcode	
S	imm[11:5]					rs2			rs1			funct3		imm[4:0]		opcode	
B	imm[12 10:5]					rs2			rs1			funct3		imm[4:1 11]		opcode	
U	imm[31:12]											rd		opcode			
J	imm[20 10:1 11 19:12]											rd		opcode			

add rd, rs1, rs2

$$x[rd] = x[rs1] + x[rs2]$$

Add. R-type, RV32I and RV64I.

Adiciona o registrador $x[rs2]$ ao registrador $x[rs1]$ e grava o resultado em $x[rd]$. O overflow aritmético é ignorado.

Formas comprimidas: **c.add** rd, rs2; **c.mv** rd, rs2

31	25 24	20 19	15 14	12 11	7 6	0
0000000	rs2	rs1	000	rd	0110011	

Codificação das instruções RV32

Cada instrução é codificada com 32 *bits*, em um dos seguintes formatos

	31	27	26	25	24	20	19	15	14	12	11	7	6	0			
R	funct7					rs2			rs1			funct3		rd		opcode	
I	imm[11:0]								rs1			funct3		rd		opcode	
S	imm[11:5]					rs2			rs1			funct3		imm[4:0]		opcode	
B	imm[12 10:5]					rs2			rs1			funct3		imm[4:1 11]		opcode	
U	imm[31:12]											rd		opcode			
J	imm[20 10:1 11 19:12]											rd		opcode			

addi rd, rs1, immediate $x[rd] = x[rs1] + \text{sext}(\text{immediate})$

Add Immediate. I-type, RV32I and RV64I.

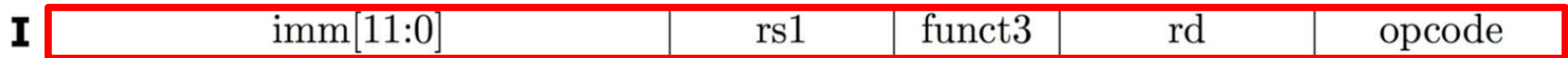
Adiciona o *valor imediato* de sinal estendido ao registrador $x[rs1]$ e escreve o resultado em $x[rd]$. O overflow aritmético é ignorado.

Formas comprimidas: **c.li** rd, imm; **c.addi** rd, imm; **c.addi16sp** imm; **c.addi4spn** rd, imm

31	20 19	15 14	12 11	7 6	0
immediate[11:0]	rs1	000	rd	0010011	

Limitações dos operandos imediatos

Formato I: `slli, srli, srai, addi, xori,`
`ori, andi, slti, sltiu, jalr`



Campo de imediato (`imm`) é codificado na instrução com apenas 12 *bits*.

Valores válidos: $-2048 : 2047$

Formatos U e J: `imm` tem 20 *bits*

Limitações dos operandos imediatos

Ao tentar montar um programa que contenha imediatos que não podem ser codificados, o montador reclama.

Ex: Programa prog.s com as seguintes instruções

```
addi a0, a5, 2048  
addi a0, a5, 10000  
addi a0, a5, -3000
```

} prog.s

```
$ as prog.s -o prog.o  
prog.s: Assembler messages:  
prog.s:1: Error: illegal operands `addi a0,a5,2048'  
prog.s:2: Error: illegal operands `addi a0,a5,10000'  
prog.s:3: Error: illegal operands `addi a0,a5,-3000'
```

Pseudo-instruções

Pseudo-instruções são instruções que existem na linguagem de montagem mas não existem na arquitetura do conjunto de instruções do processador.

O montador mapeia pseudo-instruções em instruções do processador. Ex:

`nop`


É uma pseudo-instrução mapeada em:

`addi x0, x0, 0`

Pseudo-instruções: Movimentação de dados

Pseudo-instrução: `mv rd, rs`


- Exemplo: `mv a0, a1`
- Copia o valor do registrador fonte (`rs`) para o registrador destino (`rd`)

`mv a0, a1`  `addi a0, a1, 0`

Pseudo-instruções: Movimentação de dados

Pseudo-instrução: $l\{w|h|hu|b|bu\} \text{ rd, rótulo}$

- Exemplo: `lw a0, var_x`
- Carrega um valor da memória usando como endereço um rótulo.
- Rótulos representam endereços de 32 *bits*, que não podem ser codificados em um campo de uma instrução de 32 *bits*. Esta pseudo-instrução é expandida pelo montador em 2 instruções. Ex:

`lw a0, var_x`  `auipc a0, var_x[31:12]`
`lw a0, var_x[11:0](a0)`

20 *bits* mais significativos de `var_x`

12 *bits* menos significativos de `var_x`

Pseudo-instruções: Movimentação de dados

Pseudo-instrução: $s\{w|h|b\} \text{ rd, rotulo, rs}$

- Exemplo: `sw a0, var_x, t1`
- Grava o valor de `a0` na memória usando como endereço um rótulo (o segundo registrador é usado como temporário).
- Esta pseudo-instrução é expandida pelo montador em 2 pseudo-instruções. Ex:


`sw a0, var_x, t1`  `auipc t1, var_x[31:12]`
`sw a0, var_x[11:0](t1)`

Diagram illustrating the expansion of the pseudo-instruction `sw a0, var_x, t1` by the assembler (Montador). The original instruction is expanded into two instructions:

- `auipc t1, var_x[31:12]`: This instruction loads the upper 20 bits of `var_x` (bits 31:12) into register `t1`. The annotation indicates these are **20 bits mais significativos de var_x**.
- `sw a0, var_x[11:0](t1)`: This instruction stores the value of `a0` into the lower 12 bits of `var_x` (bits 11:0), using `t1` as the base address. The annotation indicates these are **12 bits menos significativos de var_x**.

Pseudo-instruções: Movimentação de dados

Pseudo-instrução: `la rd, rótulo`

- Exemplo: `la a0, var_x`
- Grava no registrador o endereço do rótulo.
- Rótulos representam endereços de 32 *bits*, que não podem ser codificados em um campo de uma instrução de 32 *bits*. Esta pseudo-instrução é expandida pelo montador em 2 pseudo-instruções. Ex:


`la a0, var_x`  `auipc a0, var_x[31:12]`
`addi a0, a0, var_x[11:0]`

Diagram illustrating the expansion of the `la a0, var_x` pseudo-instruction into two instructions:


- `auipc a0, var_x[31:12]`: The operand `var_x[31:12]` is highlighted with a red bracket and labeled "20 bits mais significativos de var_x".
- `addi a0, a0, var_x[11:0]`: The operand `var_x[11:0]` is highlighted with a red bracket and labeled "12 bits menos significativos de var_x".

Pseudo-instruções: Movimentação de dados

Pseudo-instrução: `li rd, imediato`

- Exemplo: `li a0, 1969`
- Carrega um valor de até 32 *bits* no registrador `rd`.
- Valores pequenos (que podem ser representados com poucos *bits*) podem ser carregados com uma única instrução (`addi`) enquanto que valores grandes (que precisam ser codificados com muitos *bits*) podem exigir 2 instruções.

`li a0, 1000`  `addi a0, x0, 1000`

`li a0, 10000`  `lui a0, 0x2` } $0x2 = 10[31:12] \ll 12$
`addi a0, a0, 1808`

* $10000_{10} == 0000000000000000000010011100010000_2$

$1808 == 011100010000[11:0]$

Pseudo-instruções: Controle de fluxo

Pseudo-instrução: `ret`

- Retorna de função

```
...  
8000    jal ra, foo      # Invoca foo  
8004    sub a3, a1, a0  
...  
9000    foo:              # Função foo  
9000    add a0, a0, a1  
9004    ret              # Retorna de foo
```

`jal` grava 8004 (PC+4)
no registrador ra e salta
para foo (9000)

“`ret`” é uma
pseudo-instrução para
“`jalr x0, x1, 0`”

“`jalr x0, x1, 0`”
é equivalente a
“`jalr zero, ra, 0`”

Pseudo-instruções: Outras

Outras Pseudo-instruções do RISC-V

<code>nop</code>	<code>addi x0, x0, 0</code>	Operação No
<code>neg rd, rs</code>	<code>sub rd, x0, rs</code>	Complemento de dois
<code>negw rd, rs</code>	<code>subw rd, x0, rs</code>	Palavra em complemento de dois
<code>snez rd, rs</code>	<code>sltu rd, x0, rs</code>	"Seta" se \neq zero
<code>sltz rd, rs</code>	<code>slt rd, rs, x0</code>	"Seta" se $<$ zero
<code>sgtz rd, rs</code>	<code>slt rd, x0, rs</code>	"Seta" se $>$ zero
<code>beqz rs, offset</code>	<code>beq rs, x0, offset</code>	Desvia se $=$ zero
<code>bnez rs, offset</code>	<code>bne rs, x0, offset</code>	Desvia se \neq zero
<code>blez rs, offset</code>	<code>bge x0, rs, offset</code>	Desvia se \leq zero
<code>bgez rs, offset</code>	<code>bge rs, x0, offset</code>	Desvia se \geq zero
<code>bltz rs, offset</code>	<code>blt rs, x0, offset</code>	Desvia se $<$ zero
<code>bgtz rs, offset</code>	<code>blt x0, rs, offset</code>	Desvia se $>$ zero
<code>j offset</code>	<code>jal x0, offset</code>	Pula
<code>jr rs</code>	<code>jalr x0, rs, 0</code>	Registrador de pulo
<code>ret</code>	<code>jalr x0, x1, 0</code>	Retorna da sub-rotina

Pseudo-instruções: Outras

Outras Pseudo-instruções do RISC-V

Pseudo-Instrução	Instrução (ões) Base	Significado
li rd, immediate	<i>Miríades de sequências</i>	Load valor imediato
mv rd, rs	addi rd, rs, 0	Copia registrador
not rd, rs	xori rd, rs, -1	Complemento de um
sext.w rd, rs	addiw rd, rs, 0	Estende o sinal da palavra
seqz rd, rs	sltiu rd, rs, 1	"Seta" se = zero
bgt rs, rt, offset	blt rt, rs, offset	Desvia se >
ble rs, rt, offset	bge rt, rs, offset	Desvia se ≤
bgtu rs, rt, offset	bltu rt, rs, offset	Desvia se >, sem sinal
bleu rs, rt, offset	bgeu rt, rs, offset	Desvia se ≤, sem sinal
jal offset	jal x1, offset	Pula e linka
jalr rs	jalr x1, rs, 0	Jump e linka o registrador

Detecção de *overflow*

Detecção de *overflow* em somas de valores na representação **sem sinal**.

- Saltar para rótulo se houver *overflow*:

```
add  a0, a1, a2      # somamos os valores
bltu a0, a1, trata_ov # salta para trata_ov se
                        # houve overflow

...
trata_ov:             # Tratamento de overflow
...
```

- Indicar *overflow* em registrador

```
add  a0, a1, a2 # somamos os valores
sltu t1, a0, a1 # t1 <= 1 se (a1+a2) < a1 (Overflow)
                # do contrário, t1 <= 0
```

Detecção de *overflow*

Detecção de *overflow* em somas de valores na representação **com sinal**.

- Saltar para rótulo se houver *overflow*:

```
add    a0, a1, a2          # somamos os valores
slti   t1, a2, 0           # t1 = (a2 < 0)
slt    t2, a0, a1          # t2 = (a1+a2 < a1)
bne    t1, t2, trata_ov    # overflow se (a2<0) && (a1+a2>=a1)
                                # ou se (a2>=0) && (a1+a2<a1)
...
trata_ov:                  # Tratamento de overflow
...
```


Soma de valores multi-palavras

A soma de valores de 64 *bits* (long long)

- Utilizaremos a notação $a1 : a0$ para indicar que o par de registradores $a1$ e $a0$ armazena um número de 64 *bits* sendo que os 32 *bits* menos (mais) significativos estão em $a0$ ($a1$).

Desejamos somar dois números de 64 *bits* armazenados em $a1 : a0$ e $a3 : a2$ e armazenar o resultado em $a5 : a4$.

- Podemos somar desta forma?

```
add a4, a0, a2    # somamos a parte menos significativa
add a5, a1, a3    # somamos a parte mais significativa
```

Soma de valores multi-palavras

A soma de valores de 64 *bits* (long long)

- Utilizaremos a notação de registradores $a1$ e $a2$ de 64 *bits* sendo que os 32 *bits* menos significativos estão em $a0$ ($a1$).

Não! Desta forma o Código não leva em consideração a propagação de “vai um” entre o *bit* 31 e o *bit* 32.

Desejamos somar dois números de 64 *bits* em $a1:a0$ e $a3:a2$ e armazenar o resultado em $a5:a4$.

- Podemos somar desta forma

```
add a4, a0, a2 # somamos a parte menos significativa
add a5, a1, a3 # somamos a parte mais significativa
```

Soma de valores multi-palavras

Ao somarmos a parte menos significativa, verificamos se houve *overflow* considerando a representação sem sinal. Caso positivo, adicionamos 1 à soma da parte mais significativa.

Dessejamos somar dois valores de 64 bits armazenados em a1:a0 e a3:a2. Queremos armazenar o resultado em a5:a4.

- Podemos somar desta forma?

```
add  a4, a0, a2 # somamos a parte menos significativa
sltu t1, a4, a2 # t1 <= 1 se (a0+a2) < a2 (Overflow)
                # do contrário, t1 <= 0
add  a5, a1, a3 # somamos a parte mais significativa
add  a5, t1, a5 # somamos o "vai um"
```

Soma de valores multi-palavras

Exercício 1: Mostre o código para somar dois valores de 64 *bits* armazenados na memória, identificados pelos rótulos *x* e *y*, e armazenar o resultado no rótulo *z*

- Dica 1: Você pode carregar os valores da memória com a pseudo-instrução “lw rd, rotulo”
- Dica 2: Você pode carregar o valor armazenado na posição de memória rotulo+4 com a pseudo-instrução “lw rd, rotulo+4”

○ Exemplo:

```
x:
...
lw a0, x
lw a1, x+4
```

Soma de valores multi-palavras

Exercício 1: Mostre o código para somar dois valores de 64 *bits* armazenados na memória, identificados pelos rótulos *x* e *y*, e armazenar o resultado no rótulo *z*

```
lw    a0, x          # Carrega a parte menos sig. de x
lw    a1, y          # Carrega a parte menos sig. de y

add   a1, a0, a1      # Soma partes menos significativas
sltu  t1, a1, a0      # computa o "vai um"
sw    a1, z, a0       # Armazena resultado parcial em z

lw    a0, x+4         # Carrega a parte mais sig. de x (x+4)
lw    a1, y+4         # Carrega a parte mais sig. de y (x+4)

add   a1, a0, a1      # Soma partes mais significativas
add   a1, a1, t1       # Adiciona o "vai um"
sw    a1, z+4, a0     # Armazena resultado parcial em z+4
```

Soma de valores multi-palavras

Exercício 2: Mostre o código para somar dois valores de **128** *bits* armazenados na memória, identificados pelos rótulos x e y, e armazenar o resultado no rótulo z

- Dica 1: Agora o número é composto por quatro palavras de 32 *bits*.
- Dica 2: Lembre-se de levar em consideração o "vai um" das palavras menos significativas para as mais significativas.