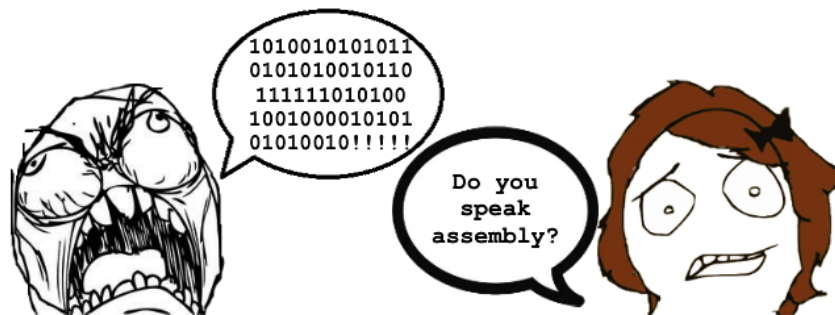
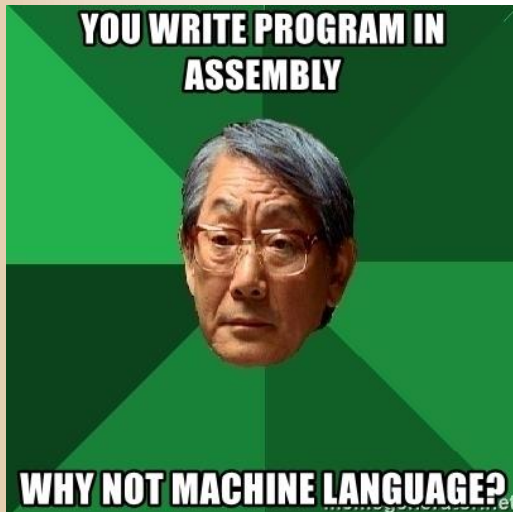




Aula 5

Assembly RISC-V

Linguagem de Máquina





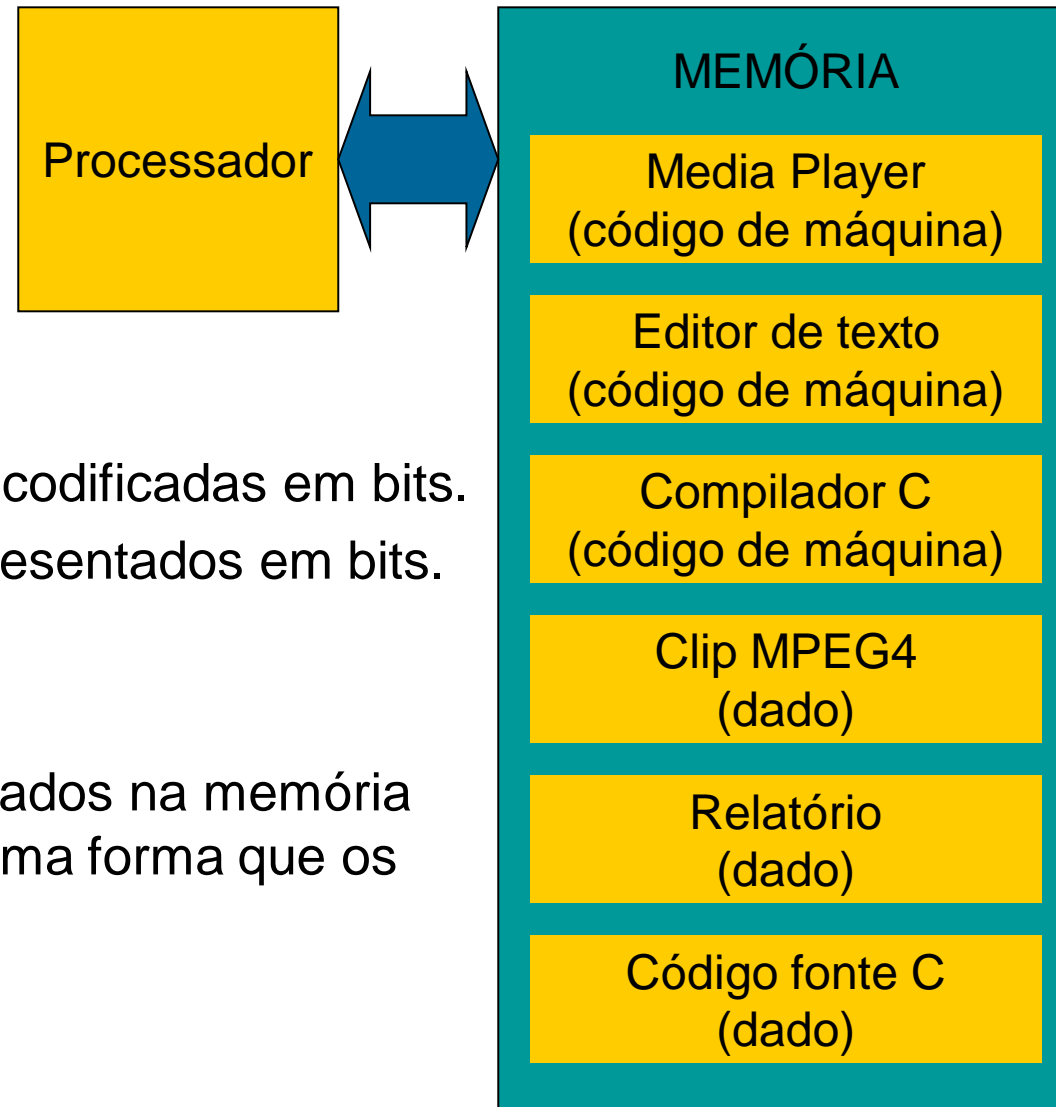
Revisão rápida: Representação Numérica Inteiros

- Binário sem sinal em N bits:
$$X = \sum_{i=0}^{N-1} b_i 2^i$$
 - Binário complemento de 2 em N bits
 - **Origem:** $X + (-X) = 2^N$
 - **Interpretação:**
$$X = -b_{N-1} 2^{N-1} + \sum_{i=0}^{N-2} b_i 2^i$$
 - **Negação:** inverter e somar 1
$$X + \bar{X} = 111 \dots 111 = -1$$

$$-X = \bar{X} + 1$$
- Ex.: $5 = (0101)_2$
 $-5 = (1010 + 1)_2 = (1011)_2 = -2^3 + 2^1 + 2^0$
- **Extensão de Sinal :** repetir o MSB
- Ex.: $5 = (0000\ 0101)_2$
 $-5 = (1111\ 1011)_2$



Programa armazenado (conceito)



Todas as instruções são codificadas em bits.
Todos os dados são representados em bits.

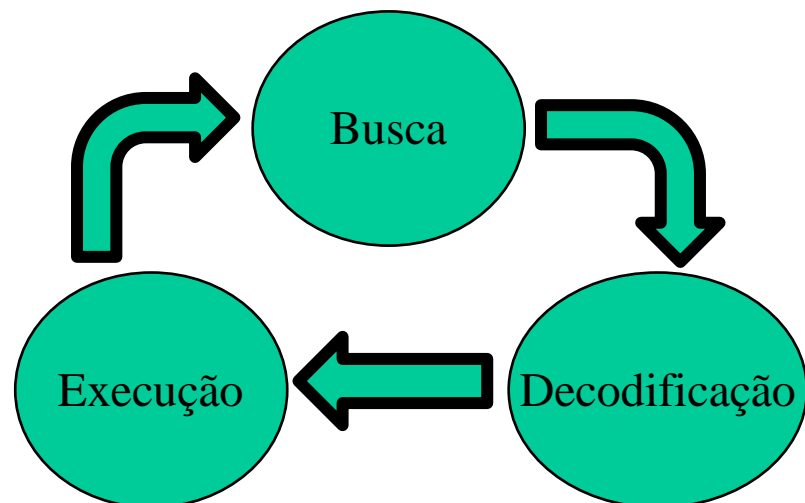
Programas são armazenados na memória para serem lidos da mesma forma que os dados.



Programa armazenado (conceito)

Ciclos de busca e execução:

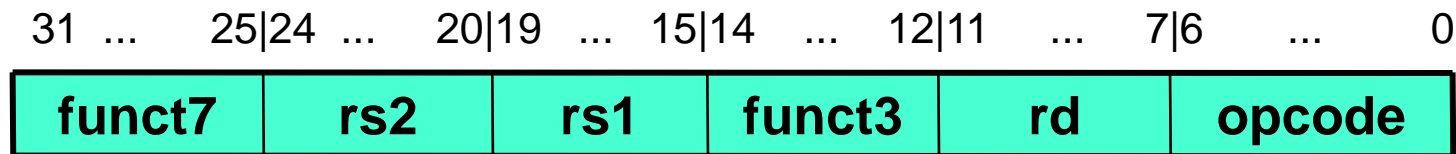
- Instruções são buscadas na memória do endereço armazenado no registrador **PC : *Program Counter*** e colocadas no registrador **IR : *Instruction Register***
- Bits do registrador IR controlam as ações subsequentes necessárias à execução da instrução.
- Busca a próxima instrução e continua...





Linguagem de máquina

Na ISA RV32I, as instruções, assim como os registradores, também têm 32 bits de comprimento divididas em campos.



- *opcode* 7 bits operação básica da instrução: *operation code*
- *rd* 5 bits registrador de operando destino: resultado: *destiny*
- *func3* 3 bits campo adicional ao opcode
- *rs1* 5 bits primeiro registrador de operando origem: *source 1*
- *rs2* 5 bits segundo registrador de operando origem: *source 2*
- *funct7* 7 bits campo adicional ao opcode



Linguagem de máquina

Exemplo: `add t0, s0, s1` # `t0=s0+s1`

□ Instrução `add`: `opcode=0x33` `funct3=0x0` `func7=0x00`

□ registradores são identificados por seus números (vide tabelas):
`t0=x5`, `s0=x8`, `s1=x9`

■ Formato Tipo-R de instrução:

Campo	funct7	rs2	rs1	funct3	rd	opcode
Tamanho	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
binário	0000 000	0 1001	0100 0	000	0010 1	011 0011
hexadecimal	0x009402b3					

Outros exemplos de Tipo-R:

`sub t0, s0, s1` # `t0=s0 - s1` subtração
`and t0, s0, s1` # `t0=s0 & s1` and lógico bit a bit (não há not !)
`srl t0, s0, s1` # `t0=s0>>s1` deslocamento lógico à direita



Linguagem de máquina

Formato de instrução para instruções com dados Imediatos.

Exemplo: `addi t0,s0,255` # `t0 = s0 + 255`

Imediato positivo ou negativo, **sempre com extensão de sinal!!!!**

■ Formato Tipo-I de instrução:

Campo	Imm[11:0]	rs1	funct3	rd	opcode
Tamanho	12 bits	5 bits	3 bits	5 bits	7 bits
binário	0000 1111 1111	0100 0	000	0010 1	001 0011
hexadecimal	0x0ff40293				

Imediato = { 20{imm[11]}, imm[11:0] }

Outros exemplos de Tipo-I:

```

ori t0,s0,0x0F0      # t0= s0 | 0x000000F0    or bit a bit com imediato
lw  t0,4(s0)         # t0= Mem[s0+4]    load word
lbu t0,4(s0)         # t0= Mem[s0+4]    load byte unsigned
srai t0,s0,2         # t0 = s0 >>> 2    deslocamento aritmético a direita
  
```



Linguagem de máquina

Formato de instrução para instruções store.

Exemplo: `sw s0, 4(s1)` # $\text{Mem}[4+s1] = s0$

Imediato positivo ou negativo, **sempre com extensão de sinal**.

■ Formato Tipo-S de instrução:

Campo	Imm[11:5]	rs2	rs1	funct3	Imm[4:0]	opcode
Tamanho	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
binário	0000 000	0 1000	0100 1	010	0010 0	010 0011
hexadecimal	0x0084a223					

Imediato = { $20\{\text{imm}[11]\}$, $\text{imm}[11:5]$, $\text{imm}[4:0]$ }

Outros exemplos de Tipo-S:

`sb s0, 4(s1)` # $\text{Mem}[4+s1] = s0$ store byte

`sh s0, 4(s1)` # $\text{Mem}[4+s1] = s0$ store half word

■ Controle de Fluxo: Desvio Incondicional

```
jal ra,Label      # Jump and Link: ra=PC+4; PC=Label
```

■ **Formato Tipo-J de instrução:**

Endereçamento relativo ao PC

PC + 111111111111 11111111 1 11111111100 0 = PC - 8



Linguagem de Máquina

■ Controle de Fluxo: Desvio Incondicional

`jalr ra, t0, imm` # Jump and Link Register : $ra = PC + 4$; $PC = (t0 + imm) \& !1$

`jalr ra, imm(t0)` # formato usado pelo Patterson

■ Formato Tipo-I de instrução:

`jalr ra, t0, 4`

Campo	Imm[11:0]	rs1	funct3	rd	opcode
Tamanho	12 bits	5 bits	3 bits	5 bits	7 bits
binário	0000 0000 0100	0010 1	000	0000 1	110 0111
hexadecimal	0x004280e7				

Imediato = { $20\{imm[11]\}$, $imm[11:0]$ }



Linguagem de Máquina

■ Controle de Fluxo: Desvio Condicional

```

beq  t0,t1,Label  # Branch if EQual:    t0 == t1 ? PC=Label : PC=PC+4
bne  t0,t1,Label  # Branch if Not Equal: t0 != t1 ? PC=Label : PC=PC+4
bge  t0,t1,Label  # Branch if Greater or Equal: t0 ≥ t1 ? PC=Label : PC=PC+4
blt  t0,t1,Label  # Branch if Less Than:   t0 < t1 ? PC=Label : PC=PC+4
  
```

Exemplo em C

```

if (i!=j)
    h=i+j;
else
    h=i-j;
  
```

Assembly RV32:

```

    bne s4,s5,Label1
    sub s3,s4,s5
    jal zero,Label2
Label1: add s3,s4,s5
Label2: ...
  
```

Em outras arquiteturas (ARM, x86) é comum o uso de *Flags* (*Zero, Signal, Overflow, Carry*) para a realização de saltos condicionais.



Linguagem de Máquina

■ Controle de Fluxo: Desvio Condicional

Exemplo: PROC: xxxxxxxx
 xxxxxxxx
 xxxxxxxx
 beq t0, t1, PROC

■ Formato Tipo-B de instrução:

Campo	Imm[12,10:5]	rs2	rs1	funct3	Imm[4:1,11]	opcode
Tamanho	7 bits	5 bits	5 bits	3 bits	5 bits	7 bits
binário	1111 111	0 0110	0010 1	000	1010 1	110 0011
hexadecimal	0xfe628ae3					

Endereçamento relativo ao PC

Label = PC + { 20{imm[12]}, imm[11:1], 0 }

PC + 11111111111111111111 1 111111 1010 0 = PC-12



Linguagem de Máquina

- Outras formas de implementar: <, >, <=, >=

Instrução: *Set on Less Than*

`slt t0,t1,t2` # $t1 < t2$? $t0=1$: $t0=0$ (Tipo-R)

`slti t0,t1,imm` # $t1 < imm$? $t0=1$: $t0=0$ (Tipo-I)

`sltu t0,t1,t2` # comparação considerando $t1$ e $t2$ sem sinal (Tipo-R)

`sltiu t0,t1,imm` # comparação com imediato considerando $t1$ sem sinal (Tipo-I)

Sempre com o Imediato estendido o sinal!



Uso de Constantes

■ Constantes de até 12 bits: Uso das instruções tipo-I

Ex.: `addi t0, t1, 4` # $t0 = t1 + 4$

■ Constantes de 12 até 32 bits: Instruções tipo-U

`lui t0, 0x12345` # Load Upper Immediate $t0 = 0x12345000$

`auipc t0, 0x12345` # Add Upper Immediate to PC $t0 = PC + 0x12345000$

■ Formato Tipo-U de instrução:

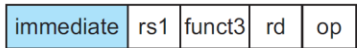
Campo	Imm[31:12]	rd	opcode
Tamanho	20 bits	5 bits	7 bits
binário	0001 0010 0011 0100 0101	0010 1	011 0111
hexadecimal	0x123452b7		

Imediato = { imm[31:12], 000000000000 }

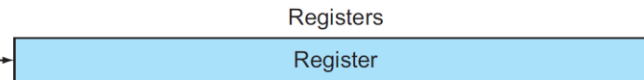
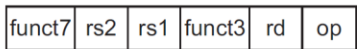


Modos de endereçamento

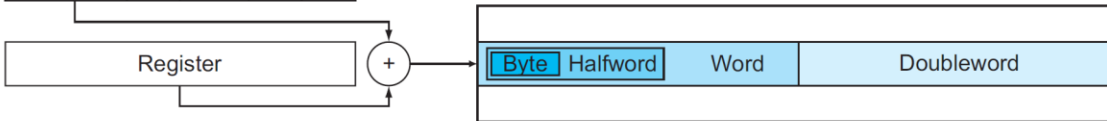
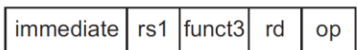
1. Immediate addressing



2. Register addressing



3. Base addressing



4. PC-relative addressing

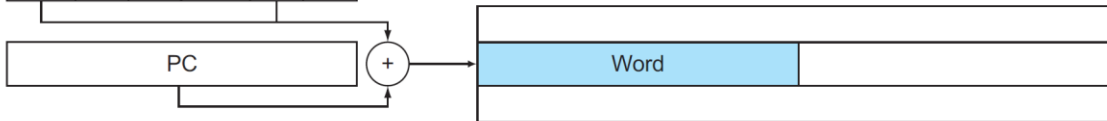
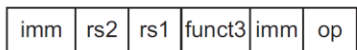


FIGURE 2.17 Illustration of four RISC-V addressing modes. The operands are shaded in color. The operand of mode 3 is in memory, whereas the operand for mode 2 is a register. Note that versions of load and store access bytes, halfwords, words, or doublewords. For mode 1, the operand is part of the instruction itself. Mode 4 addresses instructions in memory, with mode 4 adding a long address to the PC. Note that a single operation can use more than one addressing mode. Add, for example, uses both immediate (*addi*) and register (*add*) addressing.

5. Endereçamento direto (usado no x86)

6. Endereçamento pseudo-direto (usado no MIPS) : $\text{Label} = \{\text{PC}[31:28], \text{imm}, 00\}$

7. Endereçamento indireto (não usado no MIPS ou RISC-V)

O imediato é um ponteiro para um endereço que contém um ponteiro para o dado

Ex.:

```
addi t0,t1,imm
```

```
srai t0,t1,imm
```

```
add t0,t1,t2
```

```
xor t0,t1,t2
```

```
lw t0,imm(t1)
```

```
lhu t0,imm(t1)
```

```
jalr ra,t0,imm
```

```
beq t0,t1,Label
```

```
jal ra,Label
```

Ex.:

```
jump Label
```

```
j Label
```

```
load t0,*(pointer)
```



Pseudo-Instruções

- São instruções que não existem definidas na ISA do processador, mas o montador as traduz para instruções reais

```
Ex.:  mv t0, t1          # t0 = t1
      not t0,t1          # t0 = !t1
      li t0,0x123        # t0=0x00000123
      li t0,0x12345678   # t0=0x12345678
      li t0,0xDEADBEEF   # t0=0xDEADBEEF   obs.: DEADC EEF
      la t0,Label        # t0=Label
      j Label            # PC=Label
      jal Label          # ra=PC+4 PC=Label
      call Label         # ra=PC+4 PC=Label
      ret                # PC=ra
```




Exercício de Compilação

Linguagem C: while(save[i]==k) i++;

Loop:	slli t1,s3,2	0x00400000	000000000010		10011	001	00110	0010011
	add t1,t1,s6	0x00400004	00000000	10110	00110	000	00110	0110011
	lw t0,0(t1)	0x00400008	000000000000		00110	010	00101	0000011
	bne t0,s5, Exit	0x0040000C	00000000	10101	00101	001	01100	1100011
	addi s3,s3,1	0x00400010	000000000001		10011	000	10011	0010011
	j Loop	0x00400014	1111110110111111111					00000
Exit:	...	0x00400018	...					

			+3	+2	+1	+0
	Na Memória:	0x00400000	00	29	93	13
	(little endian)	0x00400004	01	63	03	33
		0x00400008	00	03	22	83
		0x0040000C	01	52	96	63
		0x00400010	00	19	89	93
		0x00400014	FE	DF	F0	6F
		0x00400018	...			

Obs.: j Loop → jal x0,Loop



Exercício de Disassembling

Memória de Código:

Memória	Código						Assembly
0x00400000	0001000000000000010000	01000	0110111				
0x00400004	00000000000000	01000	010	00101	0000011		
0x00400008	000000011111	00101	001	00110	0010011		
0x0040000C	0000000	00000	00110	000	01100	1100011	
0x00400010	0000000	00101	00101	000	00101	0110011	
0x00400014	0000000010000000000000	00000	1101111				
0x00400018	0100000	00101	00101	000	00101	0110011	
0x0040001C	0000000	00101	01000	010	00000	0100011	

Memória de Dados:

0x10010000 0000000A
 0x10010004 00000000
 0x10010008 00000000
 0x1001000C ...

Qual o valor da word
 no endereço 0x10010000
 após a execução do programa?

e se inicialmente fosse 0x0B?