Universidade de Brasília
Departamento de Ciência da Computação

Assembly MIPS

Marcus Vinicius Lamar

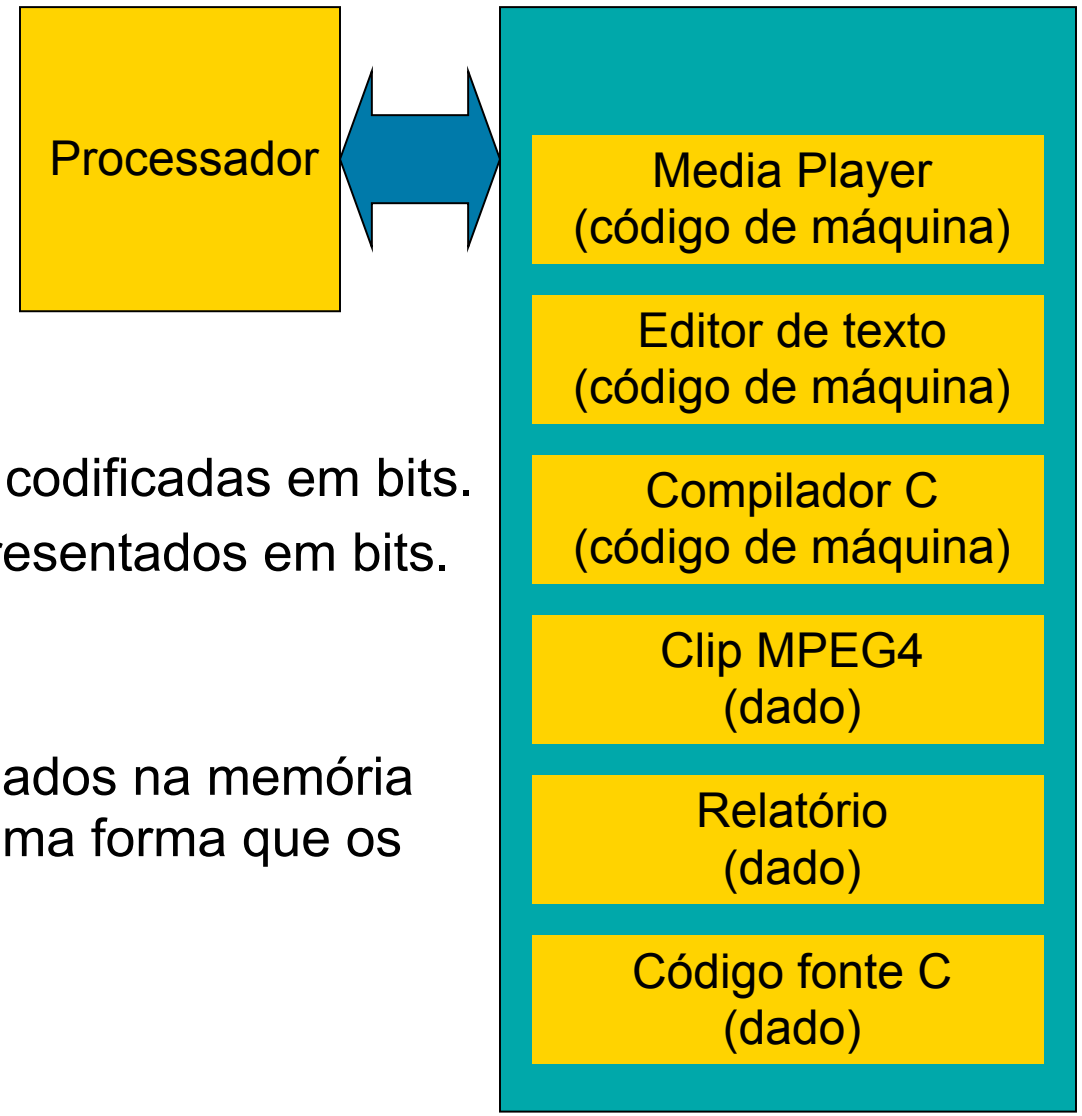
Até agora:

- MIPS
  - ⇨ Banco de 32 registradores de 32 bits cada
  - ⇨ lw Carrega *words* mas endereça *bytes* na memória
  - ⇨ Aritmética somente entre registradores ou imediato

Instrução	Significado
add $s1, $s2, $s3	$s1 = $s2 + $s3
sub $s1, $s2, $s3	$s1 = $s2 – $s3
addi $s1,$s2,imm	$s1=$s2+imm
muli $s1,$s2,imm	*$s1=$s2×imm (pseudo)*
lw $s1, imm($s2)	$s1 = Memory[$s2+imm]
sw $s1, imm($s2)	Memory[$s2+imm] = $s1

Obs.: x86 http://home.comcast.net/~fbui/intel.html e http://en.wikipedia.org/wiki/X86

Programa armazenado (conceito)



Todas as instruções são codificadas em bits.

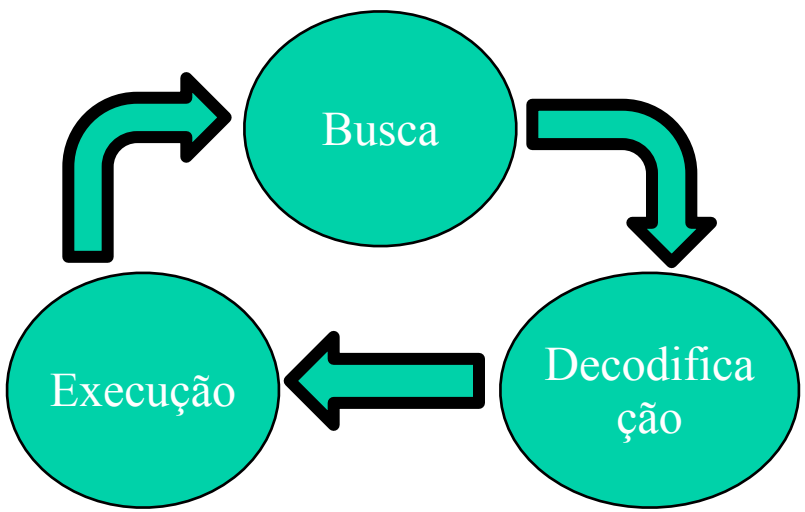
Todos os dados são representados em bits.

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 e colocadas num registrador especial (IR – *Instruction Register*).
- Bits deste registrador "controlam" as ações subseqüentes necessárias à execução da instrução.
- Busca a próxima instrução e continua...



Linguagem de máquina

No MIPS, as instruções, assim como os registradores, também têm 32 bits de comprimento dividido em campos.

- *op* 6*bits* operação básica da instrução: *opcode*
- *rs* 5*bits* primeiro registrador de operando origem
- *rt* 5*bits* segundo registrador de operando origem
- *rd* 5*bits* registrador de operando destino: resultado
- *shamt* 5*bits* deslocamento: *shift amount*
- *funct* 6*bits* variação da operação: *function code*

Linguagem de máquina

Exemplo: **add $t0, $s1, $s2**

- Instrução **add**: opcode=0 funct=32 (vide guia de referência)
- registradores são identificados por seus números (vide tabelas): $t0=8, $s1=17, $s2=18

- **Formato Tipo-R de instrução:**

Campo	op	rs	rt	rd	shamt	funct
decimal	0	17	18	8	0	32
binário	000000	10001	10010	01000	00000	100000
Tamanho	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Linguagem de máquina

O que acontece quando uma instrução necessita de campos maiores?

Ex.:

addi $t0,$t1,Imm

lw $t0,Imm($t1)

4° Princípio de Projeto:

Um bom projeto exige bons compromissos.

Linguagem de máquina

Novo tipo de formato de instrução para instruções com dados Imediatos.

Exemplo: **lw $t0, 32($s3)**

- **Formato Tipo-I de instrução:**

op	rs	rt	Imm
35	19	8	32
100011	10011	01000	0000000000100000
6 bits	5 bits	5 bits	16 bits

Obs.: MIPS não tem instrução “subi $t0,$t1,Imm” por que?

Exemplo de compilação manual

- Suponha que $t1 tenha o endereço base de *A* e que $s2 corresponda a *h*, traduza a seguinte linha em C para código de máquina MIPS: *A[300] = h + A[300];*

- Primeiro, temos que o código em assembly correspondente é:

```
lw  $t0,1200($t1)      # $t0 = A[300]
add $t0, $s2, $t0      # $t0 = h + A[300]
sw  $t0, 1200($t1)     # A[300] = h + A[300]
```

- Qual o código de máquina destas 3 instruções?

Exemplo de compilação manual ...

```
lw   $t0,1200($t1)
add  $t0, $s2, $t0
sw   $t0, 1200($t1)
```

op	rs	rt	Imediato		
			rd	shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

op	rs	rt	Imediato		
			rd	shamt	funct
100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Na Memória:

```
0x00400000  8D  28  04  B0
0x00400004  02  48  40  20
0x00400008  AD  28  04  B0
```

Linguagem de Máquina

- Operações Lógicas

Operação	C	Instrução MIPS	Opcode / Funct
Shift à esquerda	<<	sll	0 / 0
Shift à direita	>>	srl	0 / 2
AND	&	and andi	0 / 36 12
OR	|	or ori	0 / 37 13
XOR	^	xor xori	0 / 38 14
NOR		nor	0 / 39

E as outras? Ex.: not?

Linguagem de Máquina

- Controle de Fluxo
  - Desvio Incondicional

Registrador Especial PC (*Program Counter*):
indica qual o endereço da próxima instrução a ser buscada na memória

Instruções MIPS:

```
jr  $t0       # Jump Register:  PC=[$t0]        ← Obs.: Tipo-R !
j Label       # Jump Label: PC=Label
jal Label     # Jump and Link: $ra=PC+4; PC=Label
```

- **Formato Tipo-J de instrução:** Ex.: j 1200

op	Endereço
2	1200
000010	00000000000000010010110000
6 bits	26 bits

Linguagem de Máquina

Desvio Condicional

Instruções MIPS de desvio condicional:

bne $t0, $t1, *Label* # Branch if Not Equal: $t0!=$t1 ? PC=*Label*

beq $t0, $t1, *Label* # Branch if Equal: $t0==$t1 ? PC=*Label*

Exemplo:

```
if (i!=j)                      beq $s4, $s5, Label1
   h=i+j;                      add $s3, $s4, $s5
else                           j Label2
   h=i-j;              Label1: sub $s3, $s4, $s5
                       Label2: ...
```

Exercício: Implementar um Loop: for(i=0;i!=10;i++) {....}

Linguagem de Máquina

- Implementadas as comparações: == e !=
- Como implementar: <, >, <=, >= ?

Instrução MIPS: *Set on Less Than*

```
slt $t0,$t1,$t2      # $t0=1 se $t1<$t2;    $t0=0 caso contrário
slti $t0,$t1,Imm     # $t0=1  se $t1<Imm; $t0=0 caso contrário
```

Apenas com estas instruções podemos montar várias estruturas de controle.
Ao montador é reservado o registrador $1 ($at) para essa tarefa

Ex.: Construa a pseudo-instrução Branch If Less Than

```
blt $t0,$t1,Label    # se $t0 < $t1 então PC = Label
```

Constantes

- Constantes são usadas frequentemente

  Por exemplo:
  ```
  A =  7283891;
  B = A + 1881729383;
  C = B / 91827261287854;
  ```

- Soluções?

  - colocar “constantes” na memória e carregá-las (lw).
  - criar registradores *hardwired* (como $zero) para constantes como **um**.
  - colocar as constantes na própria instrução

- Princípio de projeto: agilizar o caso comum.

Constantes “pequenas”

- Constantes pequenas são usadas muito frequentemente (50% dos operandos)

  Por exemplo: A = A + 1;

  - colocar as constantes na própria instrução

  Instruções com imediato MIPS:

```
addi $29, $29, 4     # R[29]=R[29]+4
slti $8, $20, 10     # R[8]=(R[20]<10?1:0)
andi $29, $29, 6     # R[29]=R[29] & 6
ori $29, $29, 4      # R[29]=R[29] | 4
```

Formato Tipo-I:

Ex.: Implemente a pseudo instrução *Load Immediate*:

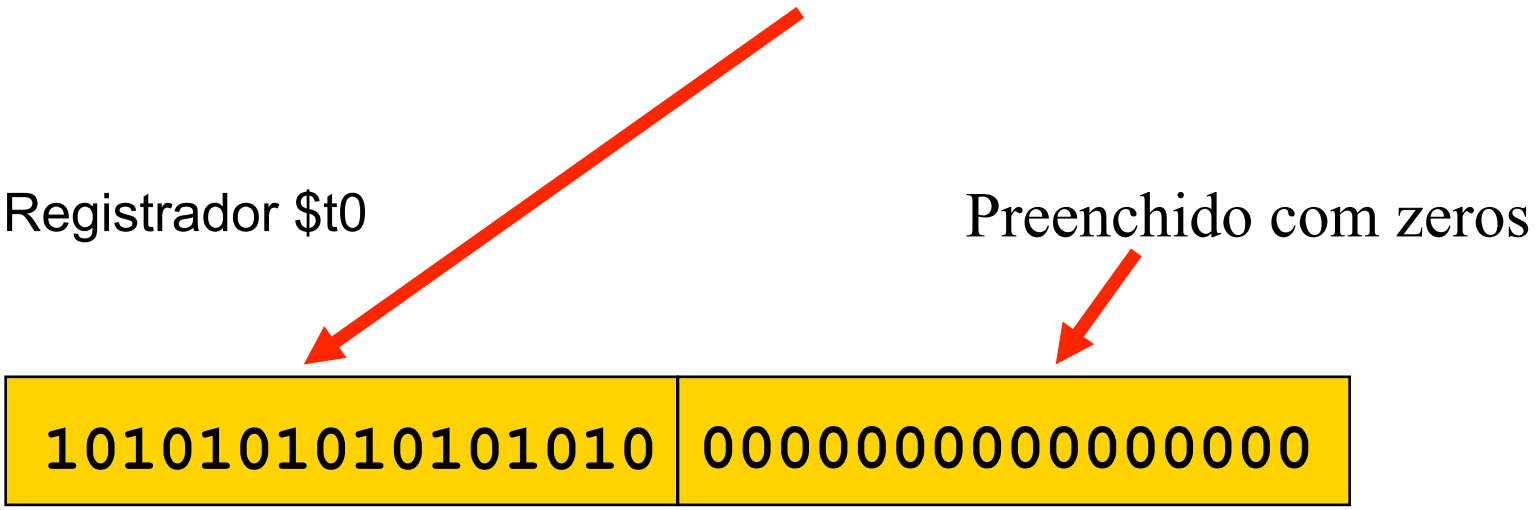
```
li $t0, Imm        # $t0 = Imm
```

E constantes maiores?

Para carregar uma constante de 32 bit num registrador são necessárias duas instruções.

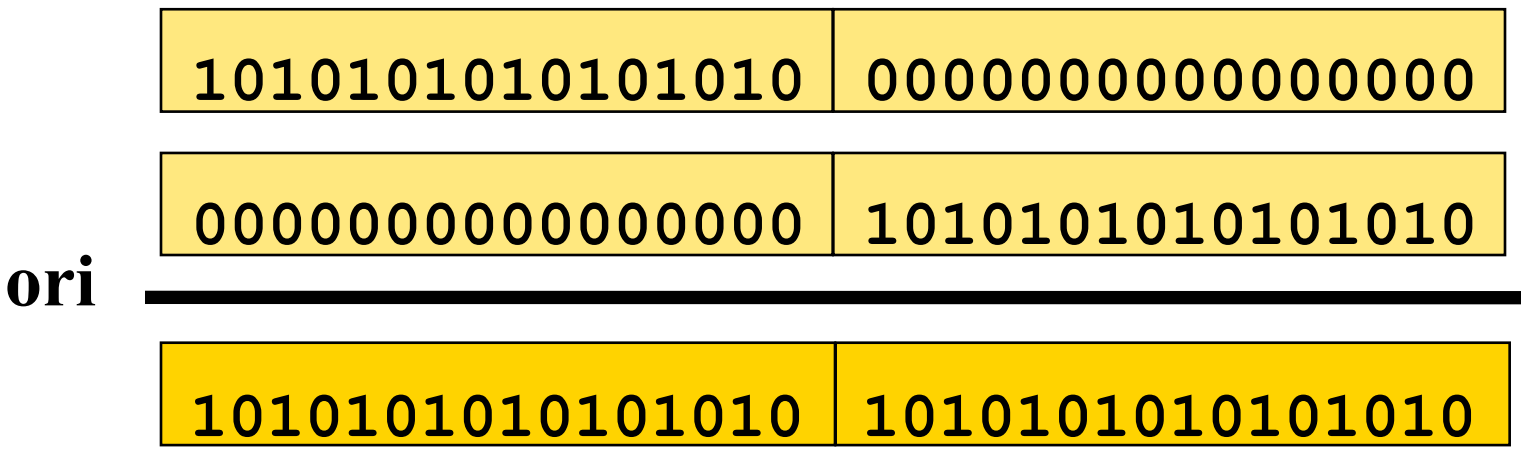
Nova instrução: *Load Upper Immediate*

```
lui $t0, 1010101010101010
```



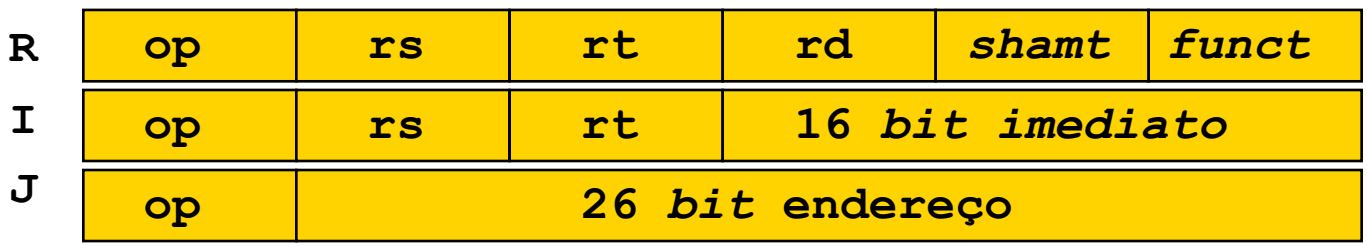
Faltam os bits menos significativos:

```
ori $t0, $t0, 1010101010101010
```



Resumo do MIPS

- Instruções simples, todas de 32 bits
- Bastante estruturada
- Somente três formatos de instruções (inteiros)



Endereços em desvios

Instruções:

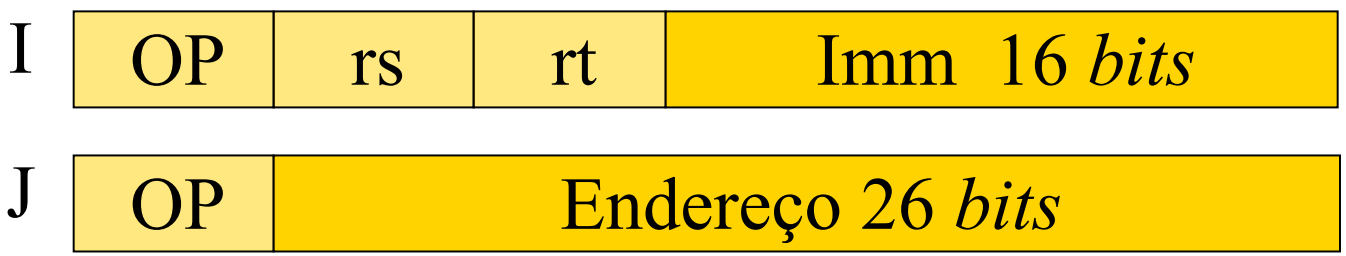
bne $t4, $t5, *Label* # Próxima instrução em *Label* se $t4 ≠ $t5

beq $t4, $t5, *Label* # Próxima instrução em *Label* se $t4 = $t5

j *Label* # Próxima instrução em *Label*

jal *Label* # $ra=PC+4; Próxima Instrução em *Label*

Formatos:



OPS!!!! Endereços não têm 32 bits!!!

Endereços em desvios

- Instruções tipo-I, beq e bne, usam Endereço Relativo
  - maioria dos desvios condicionais são locais (Princípio da Localidade)
  - utilizar *Program Counter(PC)*

    PC = (*PC+4)+ExtSinal[Imm]<<2*

- Instruções tipo-J, *j e jal* , utilizam os 4 bits mais significativos do PC e concatenam ao *Endereço* deslocado 2 bits à esquerda.

  PC = { (PC+4)[31-28] , (*Endereço*<<2) }
  - limites de endereçamento de 256 MB (64M instruções).
  - O montador e o ligador precisam cuidar disso!

Modos de endereçamento

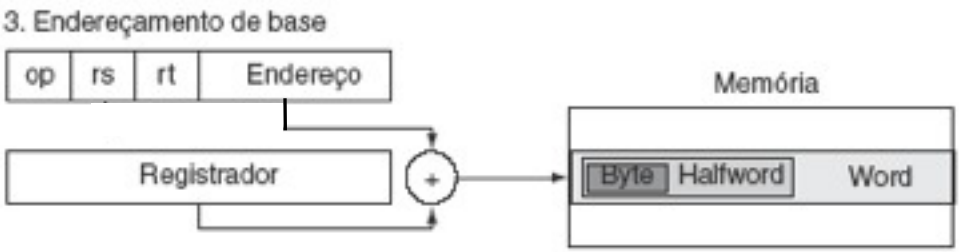
1. Endereçamento imediato

Ex.:
addi $t0,$t1,Imm

2. Endereçamento em registrador

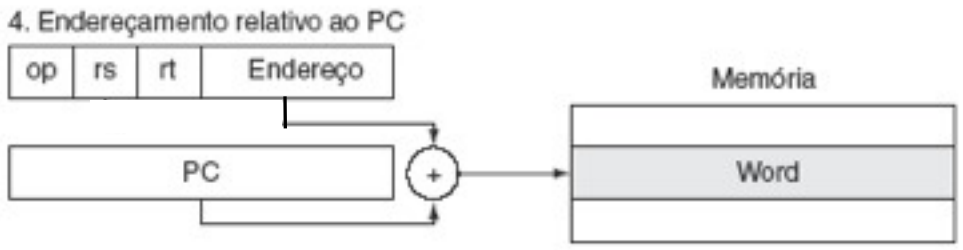
add $t0,$t1,$t2
jr $t0

3. Endereçamento de base



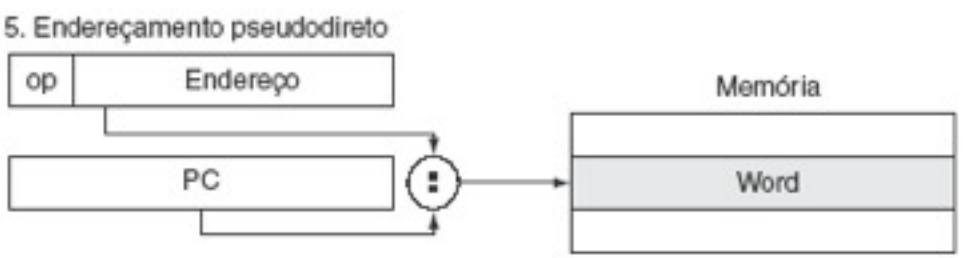
lw $t0,Imm($t1)
lhu $t0,Imm($t1)
lbu $t0,Imm($t1)

4. Endereçamento relativo ao PC



beq $t0,$t1,Label

5. Endereçamento pseudodireto



j Label
jal Label

Endereços em desvios

- **Exemplo**: while(save[i]==k) i++;

```
Loop:  sll $t1,$s3,2
       add $t1,$t1,$s6
       lw $t0,0($t1)
       bne $t0,$s5, Exit
       addi $s3,$s3,1
       j Loop
Exit:
```

80000	0	0	19	9	2	0
80004	0	9	22	9	0	32
80008	35	9	8	0		
80012	5	8	21	2		
80016	8	19	19	1		
80020	2	20000				
80024	...					

O que aconteceria se imm=-1?

Linguagem Assembly *vs.* linguagem de máquina

- O assembly fornece uma representação simbólica conveniente
  - muito mais fácil do que escrever números binários
  - por exemplo, destino primeiro
  - Pode-se usar *Labels* ao invés de endereços numéricos
- A linguagem de máquina é realidade subjacente
  - por exemplo, o destino não é mais o primeiro
  - Labels são convertidos em números apropriados
- O assembly pode fornecer “pseudo-instruções”
  - por exemplo, “move $t0, $t1” existe apenas no assembly
  - podendo ser implementada usando “add $t0,$t1,$zero”
- Ao considerar o desempenho, você deve contar as instruções reais