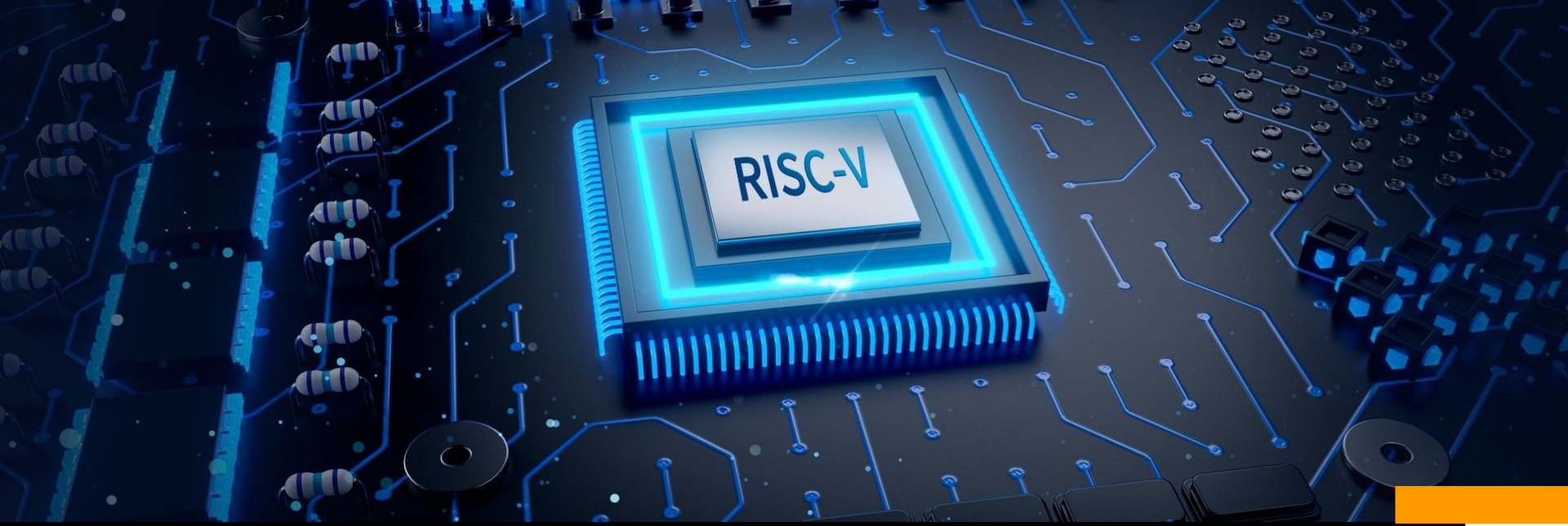


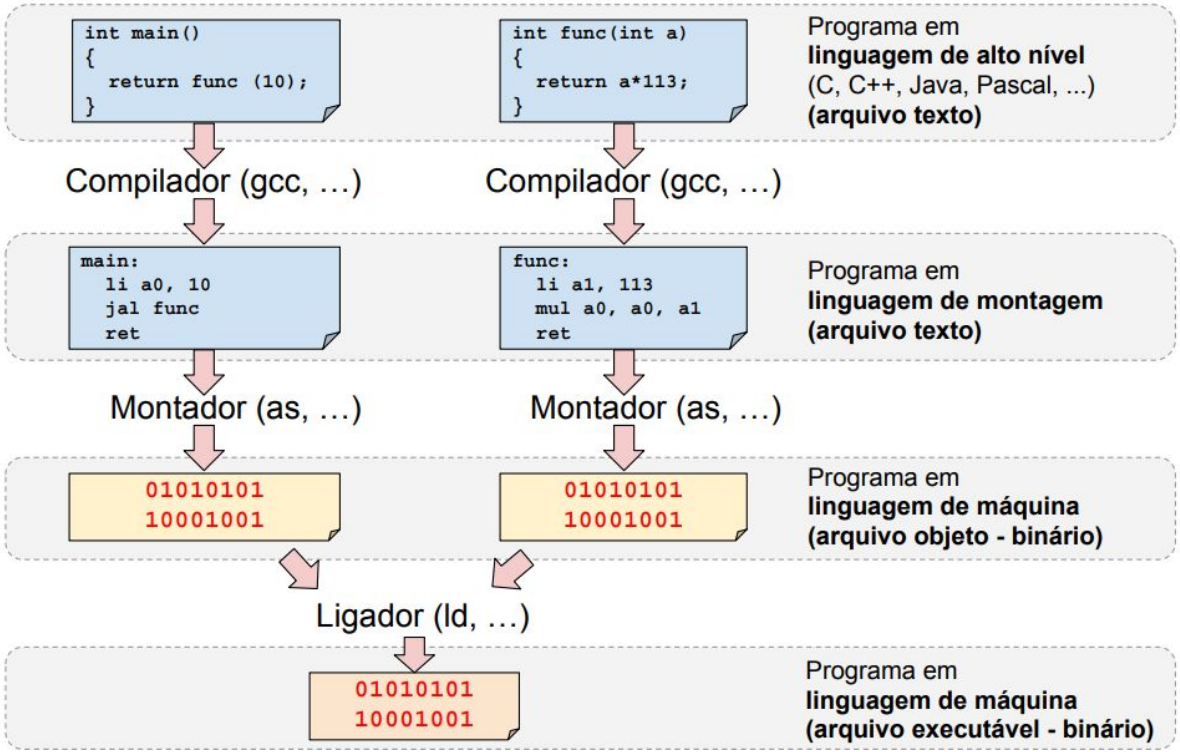
MC404AE - Organização Básica de Computadores e Ling. Montagem

Linguagem de montagem para a arquitetura RV32I

Prof. Allan M. de Souza

UNICAMP

Recapitulando



Agenda

- Sintaxe da linguagem de montagem
- Comentários
- Instruções de montagem
- Valores imediatos e símbolos
- Rótulos
- Contador de localização
- Diretivas de montagem

Sintaxe da Linguagem de Montagem

Sintaxe da Linguagem de Montagem

Programas em linguagem de montagem possuem:

- Rótulos;
- Instruções de montagem;
- Diretivas de montagem;
- Comentários.

Sintaxe da Linguagem de Montagem

Comentários no código.
(Descartados pelo montador)

```
# Comentários após
# o símbolo "cerquilha"
```

Rótulos.
(Anotações de lugar ou endereços)

```
laco:
senao:
varx:
```

Instruções de montagem.
(Instruções do programa)

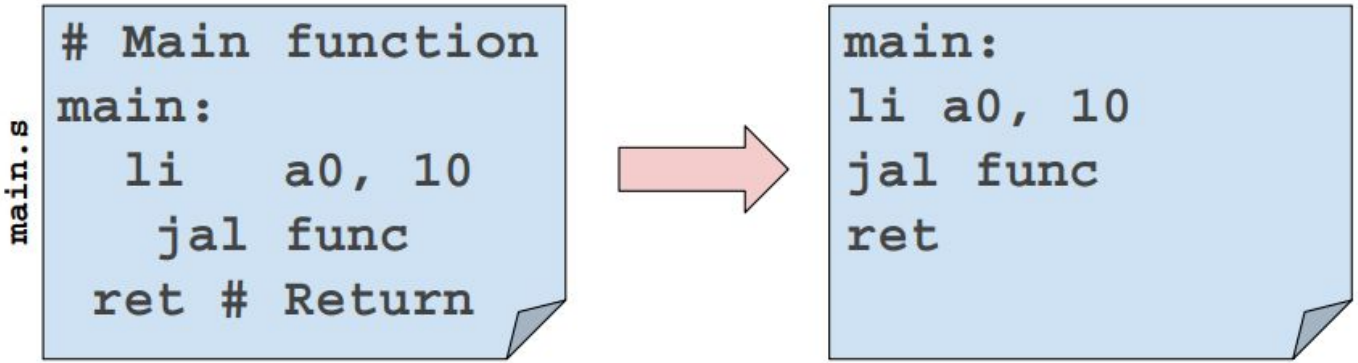
```
add a0, a1, a2
ble a1, a2, then
```

Diretivas de montagem
(Comandos para o montador)

```
.word 0x100
.byte 0xe
.section .data
```

Sintaxe da Linguagem de Montagem

O montador da GNU usa um pré-processador para remover os comentários e espaços em branco extras.



Sintaxe da linguagem de montagem

Após a remoção dos comentários, a sintaxe da linguagem de montagem pode ser resumida com a seguinte expressão regular:

```
PROGRAM -> LINES
LINES   -> LINE ['\n' LINES]
LINE    -> [<label>] [<instruction>] |
           [<label>] [<directive]
```

Sintaxe da linguagem de montagem

Exemplos de linhas válidas e inválidas:

```
LINE    -> [<label>] [<instruction>] |
           [<label>] [<directive]
```

Linhas válidas

```
sub a1, a2, a3
x: .word 10 # Variável x
y:
.byte 2

# Comentário sozinho
then: add a0, a1, a2 # soma
```

Linhas inválidas

```
x: y:
add a0, a1, a2 .word 10
.byte 2 .word 10

add a0, a1, a2 then:

add a0, a1, a2 sub a1, a2, a3
```

Comentários

Comentários são anotações no código.

- São descartados pelo pré-processador do montador.
- 2 tipos: Comentários de linha e multi-linha

Comentários

Comentários de linha (GNU Assembler).

- Delimitado por um caractere de comentário de linha ○ # no caso do RV32I
- Tudo entre a primeira ocorrência de '#' e o fim da linha é considerado comentário.
- Exemplo (comentários destacados em vermelho)

```
sub a1, a2, a3 # subtrai x de y
### Variável x ###
x: .word 10
# add a0, a1, a2 # soma z e y
```

Comentários

Comentários de linha (GNU Assembler).

- Delimitado pelos pares de texto /* e */ ○ Similar a comentários na linguagem C
- Exemplo (comentários destacados em vermelho)

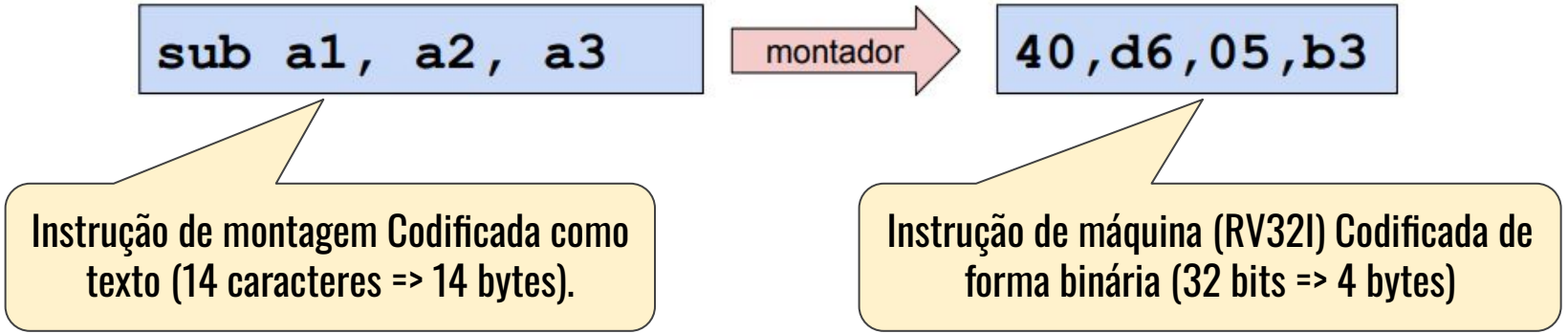
```
x:  .word 10 /* Variável x */

/* Rotina trunc42:
   Entrada: valor a ser truncado em a0
   Retorno: se a0 >= 42 então: 42,
            senão: a0
*/
trunc42:
```

Instruções de Montagem

Instruções de montagem

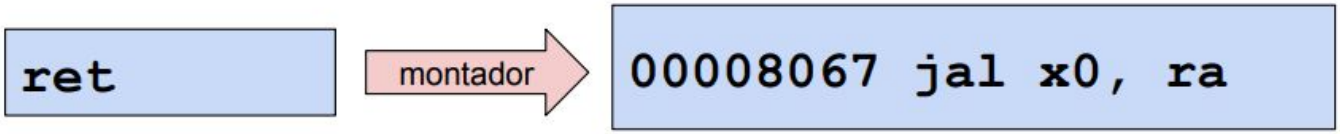
- Instruções de montagem são as instruções do programa
  - São codificadas como texto
- Instruções de montagem são traduzidas para instruções de máquina pelo montador
  - Exemplo:



Instruções de montagem

Instruções de montagem são traduzidas para instruções de máquina pelo montador

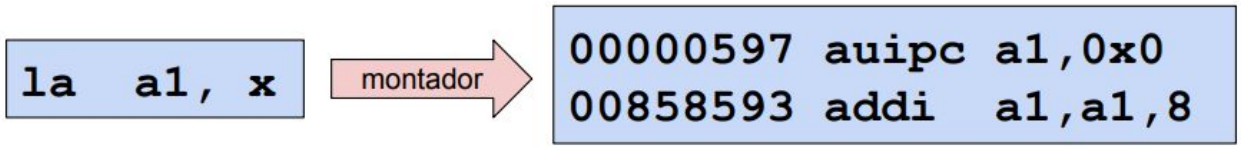
- Geralmente uma instrução de montagem é traduzida para uma instrução de máquina.
- Pseudo-instruções são instruções em linguagem de montagem que não existem na linguagem de máquina.
  - São traduzidas pelo montador para uma ou mais instruções de máquina.



Instruções de montagem

Instruções de montagem são traduzidas para instruções de máquina pelo montador

- Geralmente uma instrução de montagem é traduzida para uma instrução de máquina.
- Pseudo-instruções são instruções em linguagem de montagem que não existem na linguagem de máquina.
  - São traduzidas pelo montador para uma ou mais instruções de máquina.



Instruções de montagem

- Sintaxe de instruções de montagem RV32I
- Mnemônico + parâmetros (operandos)
- Mnemônico identifica a operação
- Ex: add => soma

Instruções de montagem

- Parâmetros das instruções de montagem RV32I:
  - lab: Símbolos (p.ex: nomes de rótulos)
  - imm: Constantes numéricas
  - rs1, rs2, rd: Registradores
  - Nome oficial (x0-x31) ou apelidos

RV32IM registers (prefix x) and their aliases

x0 zero	x1 ra	x2 sp	x3 gp	x4 tp	x5 t0	x6 t1	x7 t2	x8 s0	x9 s1	x10 a0	x11 a1	x12 a2	x13 a3	x14 a4	x15 a5
x16 a7	x17 a8	x18 s2	x19 s3	x20 s4	x21 s5	x22 s6	x23 s7	x24 s8	x25 s9	x26 s10	x27 s11	x28 t3	x29 t4	x30 t5	x31 t6

Main control status registers

CSRs:	mtvec	mepc	mcause	mtval	mstatus	mscratch
Fields of mstatus:	mie	mpie	mip			

Instruções de montagem

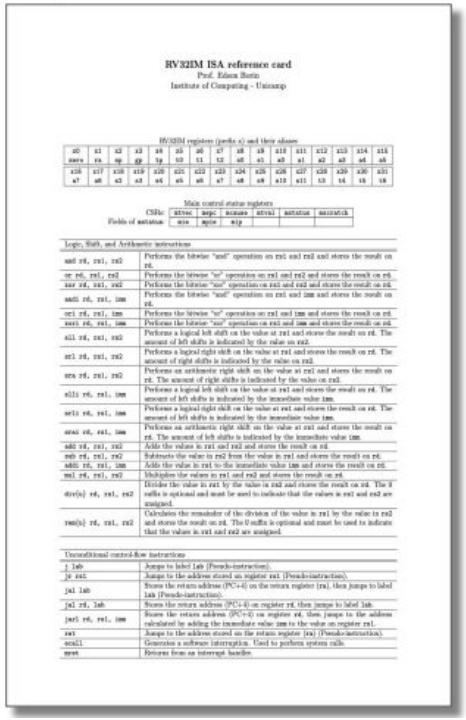
- Parâmetros das instruções de montagem RV32I:
  - lab: Símbolos (p.ex: nomes de rótulos)
  - imm: Constantes numéricas
  - rs1, rs2, rd: Registradores
  - Nome oficial (x0-x31) ou apelidos

Exemplos de instruções de montagem

```
sub  a1, a2, a3
addi a0, a1, 12
la   a0, x
ret
```

Instruções RV32IM

Cartão de referência com resumo das principais instruções



RV32IM ISA reference card

Prof. Edson Borin

Institute of Computing - Unicamp

RV32IM registers (prefix x) and their aliases

x0	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10	x11	x12	x13	x14	x15
zero	ra	sp	gp	tp	t0	t1	t2	s0	s1	a0	a1	a2	a3	a4	a5
x16	x17	x18	x19	x20	x21	x22	x23	x24	x25	x26	x27	x28	x29	x30	x31
a6	a7	s2	s3	s4	s5	s6	s7	s8	s9	s10	s11	t3	t4	t5	t6

Main control status registers

CSRs:	mtvec	mepc	mcause	mtval	mstatus	mscratch
Fields of mstatus:	mie	mpie	mip			

Logic, Shift, and Arithmetic instructions

and rd, rs1, rs2	Performs the bitwise "and" operation on rs1 and rs2 and stores the result on rd.
or rd, rs1, rs2	Performs the bitwise "or" operation on rs1 and rs2 and stores the result on rd.
xor rd, rs1, rs2	Performs the bitwise "xor" operation on rs1 and rs2 and stores the result on rd.
andi rd, rs1, imm	Performs the bitwise "and" operation on rs1 and imm and stores the result on rd.
ori rd, rs1, imm	Performs the bitwise "or" operation on rs1 and imm and stores the result on rd.
xori rd, rs1, imm	Performs the bitwise "xor" operation on rs2 and imm and stores the result on rd.
sll rd, rs1, rs2	Performs a logical left shift on the value at rs1 and stores the result on rd. The amount of left shifts is indicated by the value on rs2.
srl rd, rs1, rs2	Performs a logical right shift on the value at rs1 and stores the result on rd. The amount of right shifts is indicated by the value on rs2.
sra rd, rs1, rs2	Performs an arithmetic right shift on the value at rs1 and stores the result on rd. The amount of right shifts is indicated by the value on rs2.
slli rd, rs1, imm	Performs a logical left shift on the value at rs1 and stores the result on rd. The amount of left shifts is indicated by the immediate value imm.
srli rd, rs1, imm	Performs a logical right shift on the value at rs1 and stores the result on rd. The amount of left shifts is indicated by the immediate value imm.
srai rd, rs1, imm	Performs an arithmetic right shift on the value at rs1 and stores the result on rd. The amount of left shifts is indicated by the immediate value imm.
add rd, rs1, rs2	Adds the values in rs1 and rs2 and stores the result on rd.
sub rd, rs1, rs2	Subtracts the value in rs2 from the value in rs1 and stores the result on rd.
addi rd, rs1, imm	Adds the value in rs1 to the immediate value imm and stores the result on rd.
mul rd, rs1, rs2	Multiplies the values in rs1 and rs2 and stores the result on rd.
div[u] rd, rs1, rs2	Divides the value in rs1 by the value in rs2 and stores the result on rd. The U suffix is optional and must be used to indicate that the values in rs1 and rs2 are unsigned.
rem[u] rd, rs1, rs2	Calculates the remainder of the division of the value in rs1 by the value in rs2 and stores the result on rd. The U suffix is optional and must be used to indicate that the values in rs1 and rs2 are unsigned.

Unconditional control-flow instructions

j lab	Jumps to label lab (Pseudo-instruction).
jr rs1	Jumps to the address stored on register rs1 (Pseudo-instruction).
jal lab	Stores the return address (PC+4) on the return register (ra), then jumps to label lab (Pseudo-instruction).
jal rd, lab	Stores the return address (PC+4) on register rd, then jumps to label lab.
jarl rd, rs1, imm	Stores the return address (PC+4) on register rd, then jumps to the address calculated by adding the immediate value imm to the value on register rs1.
ret	Jumps to the address stored on the return register (ra) (Pseudo-instruction).
ecall	Generates a software interruption. Used to perform system calls.
mret	Returns from an interrupt handler.

Conditional set and control-flow instructions

slt rd, rs1, rs2	Sets rd with 1 if the signed value in rs1 is less than the signed value in rs2, otherwise, sets it with 0.
slti rd, rs1, imm	Sets rd with 1 if the signed value in rs1 is less than the sign-extended immediate value imm, otherwise, sets it with 0.
sltu rd, rs1, rs2	Sets rd with 1 if the unsigned value in rs1 is less than the unsigned value in rs2, otherwise, sets it with 0.
sltui rd, rs1, imm	Sets rd with 1 if the unsigned value in rs1 is less than the unsigned immediate value imm, otherwise, sets it with 0.
seqz rd, rs1	Sets rd with 1 if the value in rs1 is equal to zero, otherwise, sets it with 0 (Pseudo-instruction).
snez rd, rs1	Sets rd with 1 if the value in rs1 is not equal to zero, otherwise, sets it with 0 (Pseudo-instruction).
sltz rd, rs1	Sets rd with 1 if the signed value in rs1 is less than zero, otherwise, sets it with 0 (Pseudo-instruction).
sgtz rd, rs1	Sets rd with 1 if the signed value in rs1 is greater than zero, otherwise, sets it with 0 (Pseudo-instruction).
beq rs1, rs2, lab	Jumps to label lab if the value in rs1 is equal to the value in rs2.
bne rs1, rs2, lab	Jumps to label lab if the value in rs1 is different from the value in rs2.
beqz rs1, lab	Jumps to label lab if the value in rs1 is equal to zero (Pseudo-instruction).
bnez rs1, lab	Jumps to label lab if the value in rs1 is not equal to zero (Pseudo-instruction).
blt rs1, rs2, lab	Jumps to label lab if the signed value in rs1 is smaller than the signed value in rs2.
bltu rs1, rs2, lab	Jumps to label lab if the unsigned value in rs1 is smaller than the unsigned value in rs2.
bge rs1, rs2, lab	Jumps to label lab if the signed value in rs1 is greater or equal to the signed value in rs2.
bgeu rs1, rs2, lab	Jumps to label lab if the unsigned value in rs1 is greater or equal to the unsigned value in rs2.

Data movement instructions

mv rd, rs	Copies the value from register rs into register rd (Pseudo-instruction).
li rd, imm	Loads the immediate value imm into register rd (Pseudo-instruction).
la rd, rot	Loads the label address rot into register rd (Pseudo-instruction).
lw rd, imm(rs1)	Loads a 32-bit **signed** or **unsigned word** from memory into register rd. The memory address is calculated by adding the immediate value imm to the value in rs1.
lh rd, imm(rs1)	Loads a 16-bit **signed halfword** from memory into register rd. The memory address is calculated by adding the immediate value imm to the value in rs1.
lhu rd, imm(rs1)	Loads a 16-bit **unsigned halfword** from memory into register rd. The memory address is calculated by adding the immediate value imm to the value in rs1.
lb rd, imm(rs1)	Loads a 8-bit **signed** byte from memory into register rd. The memory address is calculated by adding the immediate value imm to the value in rs1.
lbu rd, imm(rs1)	Loads a 8-bit **unsigned** byte from memory into register rd. The memory address is calculated by adding the immediate value imm to the value in rs1.
sw rs1, imm(rs2)	Stores the 32-bit value at register rs1 into memory. The memory address is calculated by adding the immediate value imm to the value in rs2.
sh rs1, imm(rs2)	Stores the 16 least significant bits from register rs1 into memory. The memory address is calculated by adding the immediate value imm to the value in rs2.
sb rs1, imm(rs2)	Stores the 8 least significant bits from register rs1 into memory. The memory address is calculated by adding the immediate value imm to the value in rs2.
l{w|h|hu|b|bu} rd, lab	For each one of the lw, lh, lhu, lb, and lbu machine instructions there is a pseudo-instruction that performs the same operation, but the memory address is calculated based on a label (lab) (Pseudo-instruction).
s{w|h|b} rd, lab	For each one of the sw, sh, and sb machine instructions there is a pseudo-instruction that performs the same operation, but the memory address is calculated based on a label (lab) (Pseudo-instruction).

Valores Imediatos e Símbolos

Valores Imediatos e Símbolos

- Valores imediatos são valores numéricos.
- São codificados na própria instrução quando usados como parâmetro de instruções de montagem
- Exemplos

```
li a0, 10      # carrega dez em a0
li a0, -10     # carrega menos dez em a0
li a1, 0xa     # carrega dez em a1
li a2, 0b1010  # carrega dez em a2
li a3, 012     # carrega dez em a3
li a4, '0'     # carrega quarenta e oito em a4
li a5, 'a'     # carrega noventa e sete em a5
li a5, -'a'    # carrega menos noventa e sete em a5
```

Valores imediatos e símbolos

- Valores imediatos são valores numéricos.
- Podem ser usados em diretivas também

```
.set temp, 100
.word 10
.byte 'a'
```

Valores imediatos e símbolos

Símbolos são "nomes" que são associados a valores numéricos.

- A tabela de símbolos é a estrutura de dados que mapeia os nomes dos símbolos nos valores.
- O montador transforma rótulos em símbolos e os armazena na tabela de símbolos.
- O símbolo criado é associado a um endereço que representa a posição do rótulo no programa

Valores imediatos e símbolos

Nomes podem ser usados como parâmetro em algumas diretivas e instruções de montagem.

```
x:  .word 10 # Rótulo x: define o símbolo x
.set temp, 100 # Diretiva .set define um símbolo

la a0, x     # carrega endereço de x em a0
li a1, temp # carrega a constante temp em a1

y:  .word x   # Inicia o conteúdo da variável y
              # com o endereço da variável x
```

Rótulos

Rótulos

Rótulos são marcadores no código que serão convertidos em endereços pelo montador.

- O montador da GNU para RV32I aceita dois tipos de rótulos:
- Rótulos simbólicos; e
- Rótulos numéricos

Rótulos

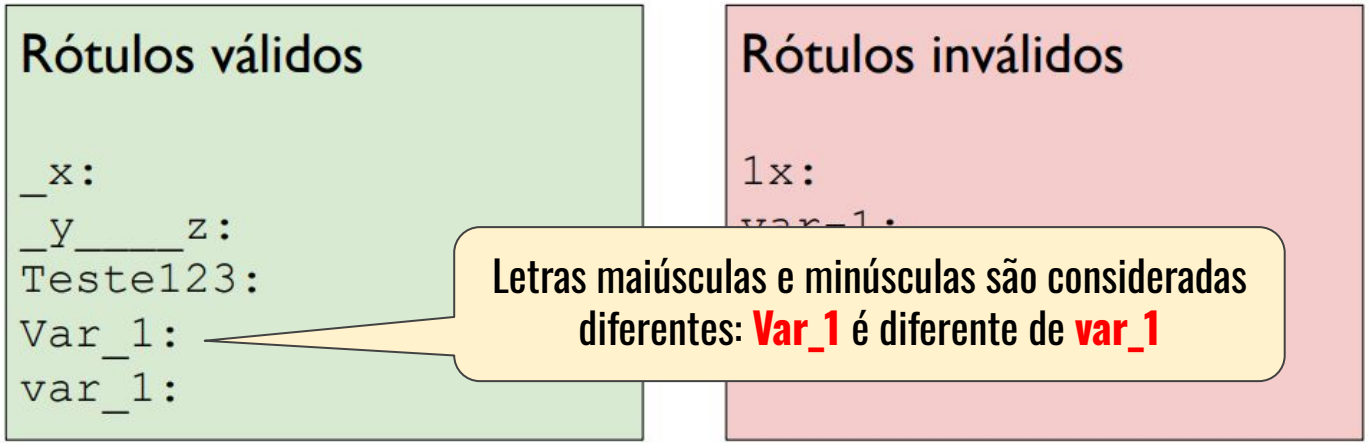
- **Rótulos simbólicos** são convertidos para símbolos e adicionados na tabela de símbolo.
  - Usados geralmente para anotar a posição (endereço) de variáveis globais e rotinas do código.

- A sintaxe de um rótulo simbólico é uma palavra com letras, dígitos numéricos e underscore '_' terminada com o caractere “:”

- Não pode começar com dígito numérico - (similar a nome de variáveis em C)

Rótulos

- A sintaxe de um rótulo simbólico é uma palavra com letras, dígitos numéricos e underscore '_' terminada com o caractere “:”
  - Não pode começar com dígito numérico(similar a nome de variáveis em C)



Rótulos

- Rótulos numéricos são rótulos locais úteis para referenciar trechos de código próximos.
  - Eles são referenciados de forma relativa e um rótulo numérico pode ter o mesmo nome que outros rótulos numéricos no mesmo arquivo.
- A sintaxe de um rótulo numérico é um dígito numérico seguido do caractere “:”

Exemplos de rótulos numéricos

```
1:
2:
1:
8:
```

Rótulos

Referências para rótulos numéricos devem incluir o dígito que identifica o rótulo e um sufixo que indica se é uma referência para o próximo rótulo numérico (f: forward) ou para o anterior (b: backward)

Exemplos de referências para rótulos numéricos

```
1:
beq a0, zero, 1f # Retorna da função
beq a0, a1,   1b # Salta para trás
1:
ret
```

Rótulos

Exemplo com rótulos simbólicos e numéricos

```
# Pow function -- computes a^b
# Inputs: a0=a, a1=b
# Output: a0=a^b
pow:
  mv   a2, a0      # Saves a0 in a2
  li   a0, 1       # Sets a0 to 1
1:
  beqz a1, 1f      # If a1 = 0 then done
  mul  a0, a0, a2  # Else, multiply
  addi a1, a1, -1  # Decrements the counter
  j    1b          # Repeat
1:
  ret
```

Contador de Localização

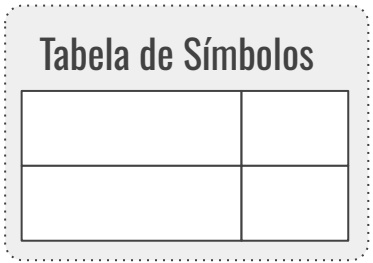
Contador de Localização

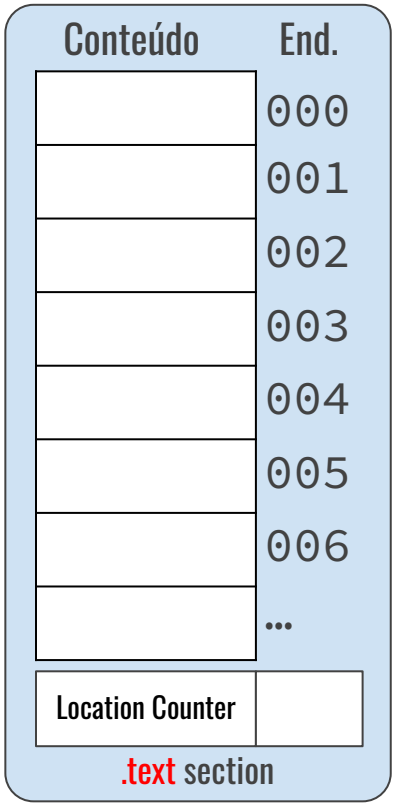
- O contador de localização (location counter) é um contador interno do montador que auxilia no processo de atribuir endereços às instruções e símbolos.
- Ele contém o endereço da próxima posição livre de memória, ou seja, a posição onde será montado o próximo elemento do programa.
- Cada seção do programa tem seu próprio contador de localização e todos são iniciados com zero no início do processo de montagem.

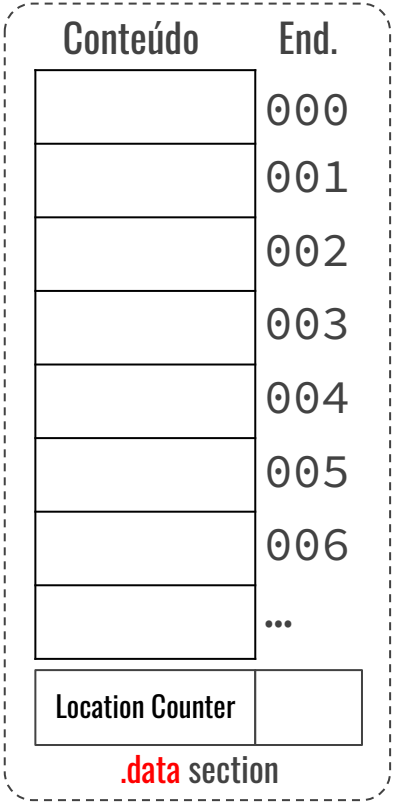
Contador de localização

```
sum42:
   addi a0, a0, 42
   ret
```

Tabela de Símbolos

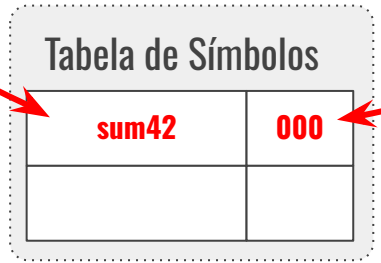


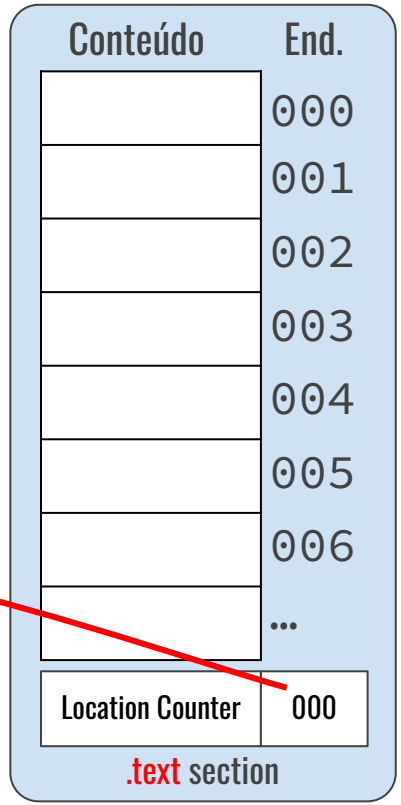


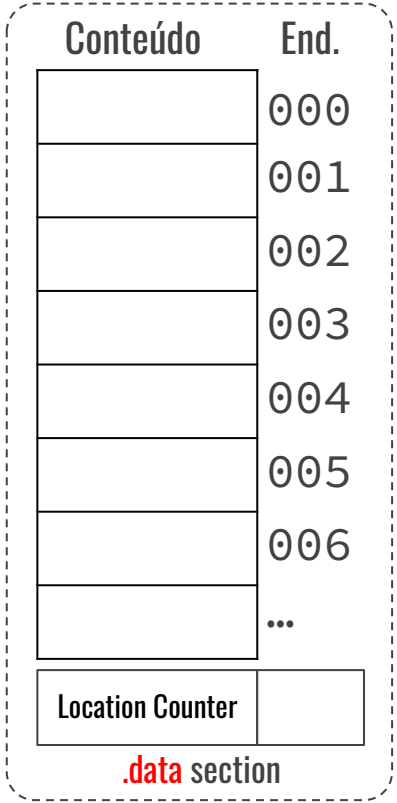


Contador de localização

```
sum42:
    addi a0, a0, 42
    ret
```







Contador de localização

0x02a50513
Instrução de máquina

```
sum42:
    addi a0, a0, 42
    ret
```

Tabela de Símbolos

sum42	000

Conteúdo	End.
00010011	000
00000101	001
10100101	002
00000010	003
	004
	005
	006
	...

Location Counter	~~000~~ 004

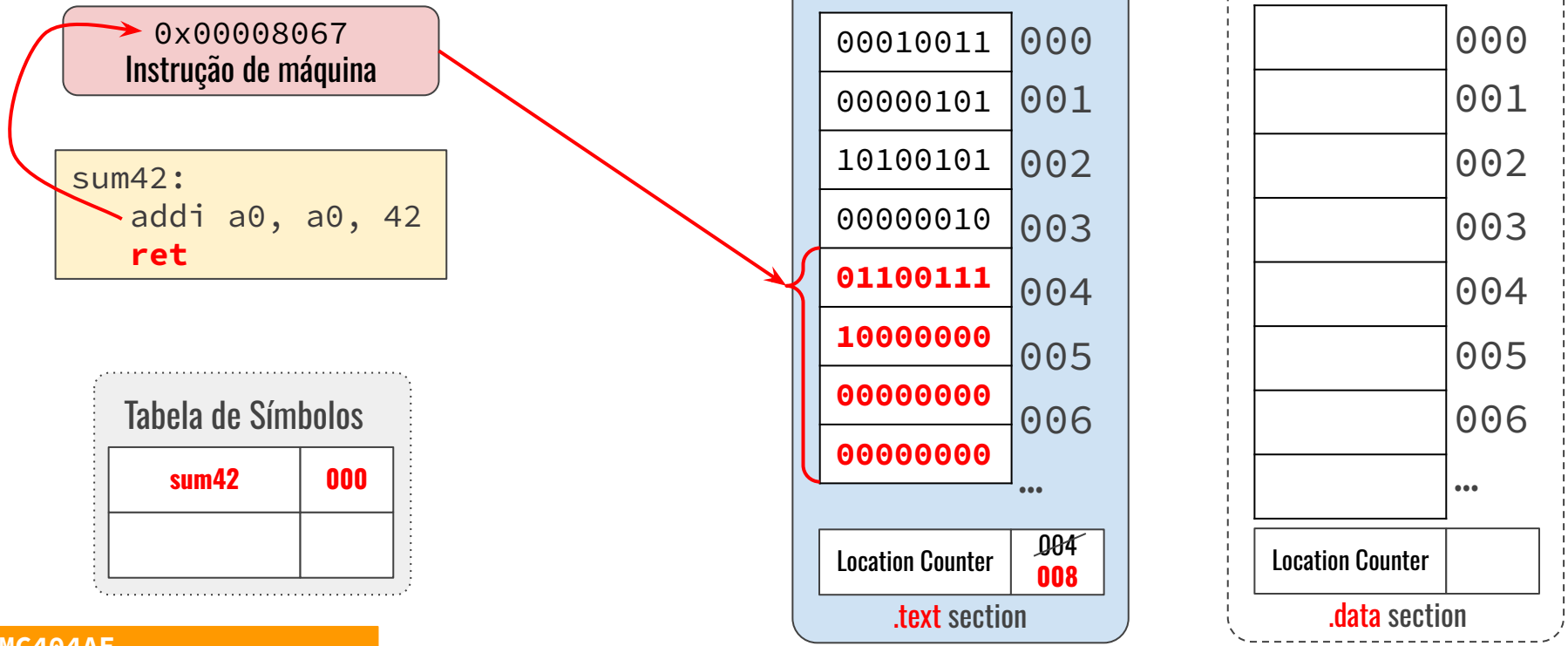
.text section

Conteúdo	End.
	000
	001
	002
	003
	004
	005
	006
	...

Location Counter	

.data section

Contador de localização



Diretivas de Montagem

Diretivas de Montagem

Diretivas de montagem são comandos para controlar o processo de montagem.

- Estes comandos são interpretados pelo montador durante o processo de montagem!

Exemplo:

- A diretiva "**.byte 45**" instrui o montador a colocar um byte com valor 45 no programa.

Diretivas de Montagem

Inserção de valores

Directive	Arguments	Description
.string	string	Emit NULL terminated string
.asciz	string	Emit NULL terminated string (alias for .string)
.ascii	string	Emit string without NULL character
.byte	ex	
.half	e	
.word	e	
.dword	e	

Diretivas: .string e .asciz adicionam uma string codificada em ASCII e terminada em zero no ponto atual de montagem do programa.
Exemplo: .string "oi"

- Adiciona **três bytes** no programa, os bytes com valores 111 e 105, referentes à codificação ASCII das letras "o" e "i", e o byte com valor 0.

Diretivas

Inserção de valores

Directive	Arguments	Description
.string	string	Emit NULL terminated string
.asciz	string	Emit NULL terminated string (alias for .string)
.ascii	string	Emit string without NULL character
.byte	ex	
.half	e	
.word	e	
.dword	e	

Diretivas: .string e .asciz adicionam uma string codificada em ASCII mas não adiciona o byte zero no final

Exemplo: .string "oi"

- Adiciona **dois bytes** no programa, os bytes com valores 111 e 105, referentes à codificação ASCII das letras "o" e "i".

Diretivas de Montagem

Inserção de valores

Directive	Arguments	Description
.string	string	Emit NULL terminated string
.asciz	string	Emit NULL terminated string (alias for .string)
.ascii	string	Emit string without NULL character
.byte	ex	
.half	e	
.word	e	
.dword	e	

Diretiva: .byte adiciona um (ou mais) byte(s) no ponto atual de montagem do programa.

Exemplo: .byte 10, 20, 30

- Adiciona três bytes no programa, com os valores 10, 20 e 30.

Diretivas de Montagem

Inserção de valores

Directive	Arguments	Description
.string	string	Emit NULL terminated string
.asciz	string	Emit NULL terminated string (alias for .string)
.ascii	string	Emit string without NULL character
.byte	expression [, expression]*	Emit one or more 8-bit comma separated words
.half	expressi	
.word	expre	
.dword	expressi	

Diretiva .half, .word e .dword adicionam um (ou mais) valores de 16, 32 e 64 bits, respectivamente, no ponto atual de montagem do programa.

.word 20, 30

- Adiciona dois valores de 32 bits (4 bytes) no programa.

Diretivas de Montagem

Inserção de valores

Directive	Arguments	Description
.string	string	Emit NULL terminated string
.asciz	string	Emit NULL terminated string (alias for .string)
.ascii	string	Emit string without NULL character
.byte	expression [, expression]*	Emit one or more 8-bit comma separated words
.half	expression [, expression]*	Emit one or more 16-bit comma separated words
.word	expression [, expression]*	Emit one or more 32-bit comma separated words
.dword	expression [, expression]*	Emit one or more 64-bit comma separated words

Quando combinadas com rótulos, podem ser usadas para **declarar** e **inicializar** **variáveis globais**

Exemplo

```
x: .word 12 # Variável x iniciada com valor 12 (4 bytes)
y: .byte 12 # Variável y iniciada com valor 12 (1 byte)
msg: .string "MC404" # Variável msg com string "MC404"
```

Diretivas de Montagem

diretiva .section

Programas executáveis são organizados em seções.

- Seção ".text": dedicada ao armazenamento do código do programa (as instruções)
- Seção ".data": dedicada ao armazenamento das variáveis globais inicializadas
- Seção ".bss": dedicada ao armazenamento das variáveis globais não inicializadas
- Seção ".rodata": dedicada ao armazenamento de constantes (ready-only).

Diretivas de Montagem

---

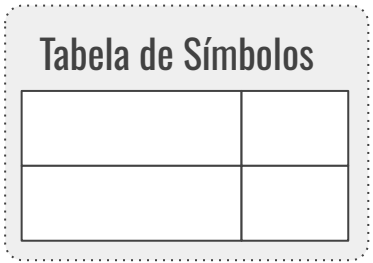
diretiva .section

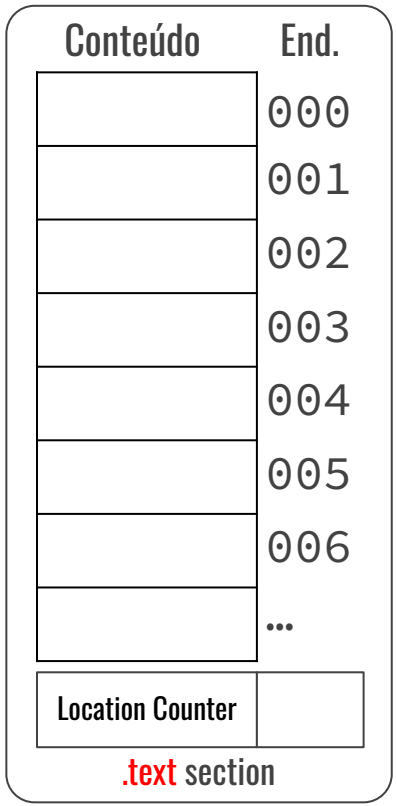
- A diretiva **.section** serve para informar ao montador qual seção deve receber os próximos elementos a serem montados.

Contador de localização

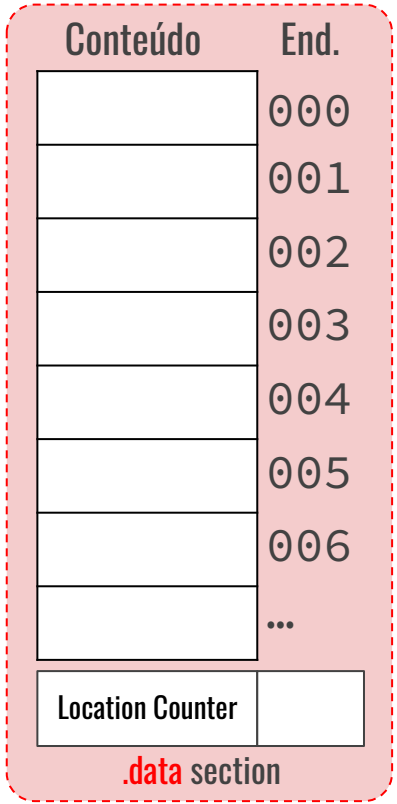
```
.section .data
msg:  .ascii “hello”
x:    .word 10
```

Tabela de Símbolos



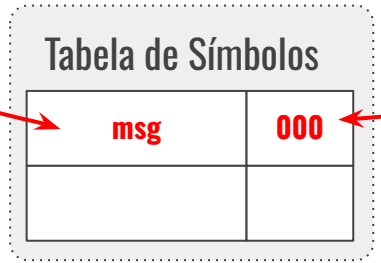


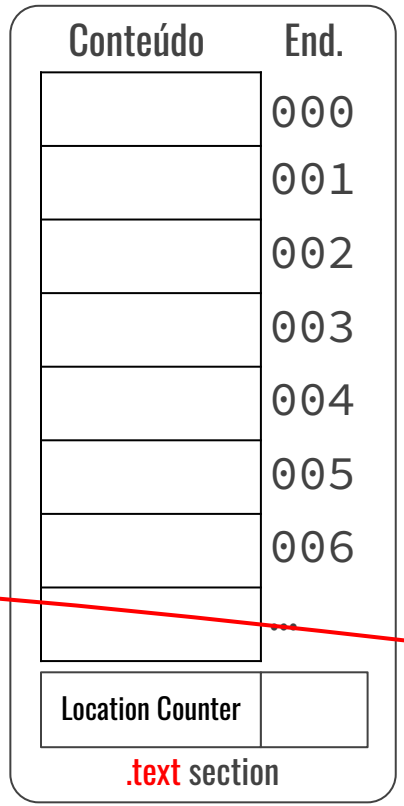
Seção ativa



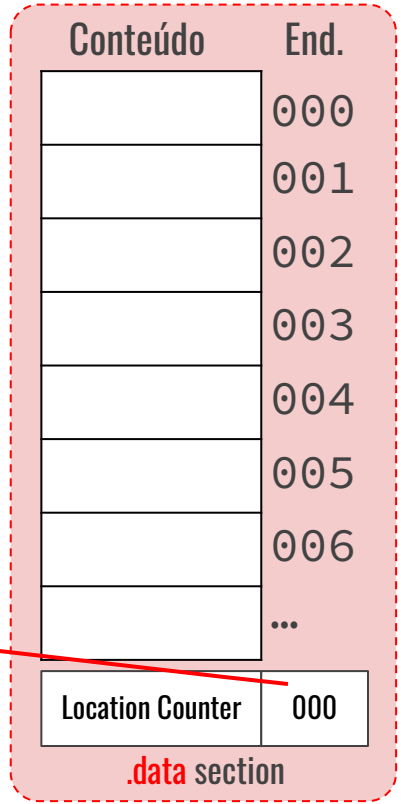
Contador de localização

```
.section .data
msg:  .ascii “hello”
x:    .word 10
```





Seção ativa



Contador de localização

68,65,6c,6c,6f
texto em ASCII

```
.section .data
msg:  .ascii "hello"
x:    .word 10
```

Tabela de Símbolos

msg	000

Conteúdo	End.
	000
	001
	002
	003
	004
	005
	006
	...
Location Counter	

.text section

Seção ativa

Conteúdo	End.
01101000	000
01100101	001
01101100	002
01101100	003
01101111	004
	005
	006
	...
Location Counter	~~000~~ 005

.data section

Diretivas de Montagem

Diretiva .section

```
.section .data # Muda para a seção .data
x: .word 10 # variável global inicializada com o valor 10 (4 bytes)
y: .word 10 # variável global inicializada com o valor 10 (4 bytes)

.section .text # Muda para a seção .text (código)
add_x_y:
    lw a0, x
    lw a1, y
    add a0, a0, a1
    sw a0, z, a1
    ret

.section .rodata # Muda para a seção read only data
msg: .asciz "Assembly rocks!" # constante

.section .bss # Muda para a seção .bss (dados não inicializados)
z: .skip 4  # variável global não inicializada (4 bytes)
```

Diretivas de Montagem

No montador da GNU:

- A diretiva .data é um apelido para a diretiva .section .data
- A diretiva .text é um apelido para a diretiva .section .text
- A diretiva .bss é um apelido para a diretiva .section .bss

Diretivas de Montagem

- Usando as diretivas .data e .text

```
.data
x: .word 10
y: .word 10

.text
add_x_y:
    lw a0, x
    lw a1, y
    add a0, a0, a1
    sw a0, z, a1
    ret
```

=

```
.section .data
x: .word 10
y: .word 10

.section .text
add_x_y:
    lw a0, x
    lw a1, y
    add a0, a0, a1
    sw a0, z, a1
    ret
```

Diretivas de Montagem

As diretivas .set e .equ

- Adicionam símbolos e seus respectivos valores à tabela de símbolos.
- Podem ser usadas para "nomear" constantes.

```
.set max_value, 42

truncates_value_to_max:
  li    t1, max_value
  ble   a0, t1, ok
  mv    a0, t1
ok:
  ret
```

Diretivas de Montagem

As diretivas .set e .equ

```
.set max_value, 42

truncates_value_to_max:
  li    t1, max_value
  ble   a0, t1, ok
  mv    a0, t1
ok:
  ret
```

```
objdump -t max_value.o

max_value.0: file format elf32-littleriscv
...
SYMBOL TABLE:
...
0000002a l *ABS* 00000000 max_value
00000000 l .text 00000000 truncates_value_to_max
0000000c l .text 00000000 ok
```

Diretivas de Montagem

Diretiva .globl

- Os símbolos do programa são classificados como globais ou locais.
  - Locais: apenas visíveis dentro do mesmo arquivo.
  - Globais: visíveis externamente => Usados pelo ligador para resolver referências não definidas. Por padrão os símbolos são locais.

- A diretiva ".globl nome" transforma o símbolo nome em global.

func.s

```
.globl func
func:
  addi a0, a0, 42
  ret
```

Diretivas de Montagem

Diretiva .align

- Algumas arquiteturas de computador exigem que as instruções (ou dados) maiores do que um byte sejam armazenados na memória em endereços múltiplos do tamanho da instrução (ou do dado)

RV32I

- Instruções: Precisam ser armazenadas em endereços múltiplos de 4
- Dados: O manual recomenda armazenar dados do tipo halfword (2 bytes), word (4 bytes), e double word (8 bytes), em endereços múltiplos de 2, 4 e 8, respectivamente.

Diretivas de Montagem

Diretiva .align

- Instruções não alinhadas são instruções que não estão armazenadas em endereços múltiplos de 4 bytes.

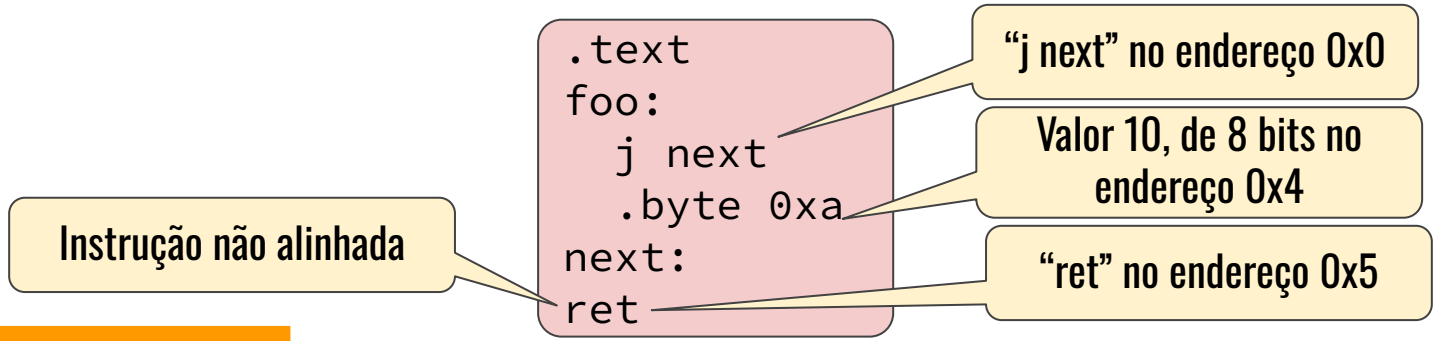
- A CPU não executa corretamente instruções não alinhadas

Diretivas de Montagem

Diretiva .align

- Instruções não alinhadas são instruções que não estão armazenadas em endereços múltiplos de 4 bytes.

- A CPU não executa corretamente instruções não alinhadas



Diretivas de Montagem

Diretiva .align

- O montador não verifica se instruções estão alinhadas ou não
- O programa abaixo é montado [illegible]em mensagens de erro ou **warning**.

```
.text
foo:
  j next
  .byte 0xa
next:
ret
```

Responsabilidade do programador (ou compilador) se certificar que instruções e dados estão alinhados!

Diretivas de Montagem

Diretiva .align

- Forma **incorreta** de se alinhar instruções

```
.text
foo:
  j next
  .byte 0xa
  .skip 3
next: ret
```

Diretivas de Montagem

Diretiva .align

- Forma **correta** de se alinhar instruções

```
.text
foo:
  j next
  .byte 0xa
.align 2
next: ret
```

Diretivas de Montagem

Diretiva .align

- Forma correta de se alinhar instruções: .align 2
- A diretiva **.align N** verifica se o valor no contador de localização é múltiplo de a 2N e:
  - Se não for, avança o contador de localização até que seu valor seja múltiplo de 2N.
  - Se já for, não faz nada.

Diretivas de Montagem

Diretiva .align

- O compilador geralmente insere a diretiva .align 2 antes de cada função do programa para garantir que as instruções do programa estarão alinhadas a endereços múltiplos de 4.

```
.text
.align 2

func1:
  addi a0, a0, 2
  ret

.align 2
func2:
  addi a0, a0, 2
  ret
```

Diretivas de montagem

Diretiva .align – Alinhamento de dados

```
.data
x: .byte 42
y: .word 1969
```

as, ld

```
SYMBOL TABLE:
...
00011054 l  .data  00000000 x
00011055 l  .data  00000000 y
...
```

```
.data
x: .byte 42
.align 2
y: .word 1969
```

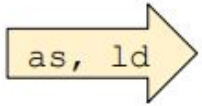
as, ld

```
SYMBOL TABLE:
...
00011054 l  .data  00000000 x
00011058 l  .data  00000000 y
...
```

Diretivas de montagem

Diretiva .align - Alinhamento de dados

```
.data
x: .byte 42
.align 2
.align 2
.align 2
.align 2
y: .word 1969
```




```
SYMBOL TABLE:
...
00011054 l   .data    00000000 x
00011058 l   .data    00000000 y
...
```

Diretivas de montagem

Alocação de espaço na seção .bss

Variáveis globais não inicializadas devem ser adicionadas à seção .bss.

- Durante a carga do programa, o SO aloca espaço na memória, mas não inicializa esta parte da memória
  - OBS: Alguns SOs inicializam esta parte com zero, mas o programador não deve supor que o conteúdo será iniciado.
- O montador não permite adicionar valores (dados ou instruções) nesta seção.

Diretivas de montagem

Alocação de espaço na seção .bss

- O montador não permite adicionar valores (dados ou instruções) na seção.

Precisamos reservar espaço (avançar o contador de localização) sem adicionar valores na seção!

xlib.s

```
.section .bss
x: .word 10
.section .text
get_x:
  la t1, x
  lw a0, (t1)
  ret
```

```
$ as -march=rv32im xlib.s -o xlib.o
xlib.s: Assembler messages:
xlib.s:2: Error: attempt to store non-zero value in section `.bss'
```

Diretivas de montagem

Alocação de espaço na seção .bss

- A diretiva .skip N avança o contador de localização sem adicionar valores

```
.section .bss
x: .skip 4
.section .text
get_x:
  la t1, x
  lw a0, (t1)
ret
```

".skip 4" aloca 4 bytes para a variável x!

Diretivas de montagem

Alocação de espaço na seção .bss

- A diretiva.common name,size,align
- 1) Alinha o ponto atual de montagem da seção .bss em um endereço múltiplo de 2^{align};
- 2) Define um rótulo name na seção .bss; e
- 3) Aloca size bytes na seção .bss

Ela pode ser usada em qualquer ponto do programa e não há necessidade de mudar a seção para .bss (o montador faz isso automaticamente)

Diretivas de montagem

Alocação de espaço na seção .bss

- A diretiva.common name,size,align

```
.text
foo:
  mv a0, a1
  ret
.common vetor1,40,2
.common vetor2,40,2
bar:
  li a0, 42
  ret
```

as, ld, objdump

```
SYMBOL TABLE:
...

00010074 l .text 00000000 foo
0001007c l .text 00000000 bar
...
00011084 g .bss 00000028 vetor2
...
000110ac g .bss 00000028 vetor1
...
```