

cin.ufpe.br



UNIVERSIDADE FEDERAL DE PERNAMBUCO

Infra-estrutura Hardware

Infra-estrutura de Hardware

# AULA SIMULADOR RISC-V / ASSEMBLY RISC-V

# Roteiro da Aula

---



- Revisão
- Simulador Venus
- Lista

# Representação da Informação



Programa em  
Linguagem de alto  
nível (e.g., C)

*Compilador*

```
temp = v[k];  
v[k] = v[k+1];  
v[k+1] = temp;
```

Programa em linguagem  
assembly (e.g., MIPS)

*Montador*

```
lw $to,    0($2)  
lw $t1,    4($2)  
sw $t1,    0($2)  
sw $t0,    4($2)
```

Programa em  
linguagem de  
Máquina (MIPS)

Hardware

```
1000 1100 0100 1000 0000 0000 0000 0000  
1000 1100 0100 1001 0000 0000 0000 0100  
1010 1100 0100 1001 0000 0000 0000 0000  
1010 1100 0100 1000 0000 0000 0000 0100
```



# QUAIS INSTRUÇÕES QUE UM PROCESSADOR EXECUTA?

# Conjunto de Instruções (ISA)



- O repertório de instruções de um computador
- Diferentes computadores têm diferentes conjuntos de instruções
  - Mas com muitos aspectos em comum
- Os primeiros computadores tinham conjuntos de instruções muito simples
  - Implementação simplificada
- Muitos computadores modernos também possuem conjuntos de instruções simples



# O ISA do RISC-V



- Usado como o exemplo ao longo do livro...
- Desenvolvido na UC Berkeley como ISA aberto
- Agora gerenciado pela Fundação RISC-V (riscv.org)
- Típico de muitos ISAs modernos
- ISAs similares têm uma grande parcela do mercado principal incorporado
  - Aplicações em eletrônicos de consumo, equipamentos de rede / armazenamento, câmeras, impressoras, ...



# Operandos em Registradores



- Instruções aritméticas usam operandos de registro
- MIPS possui 32 registradores de 32 bits cada
  - Os dados de 32 bits são chamados de "palavra"
- RISC-V possui 32 registradores de 64 bits cada
  - Use para dados com acesso frequente
  - Os dados de 64 bits são chamados de "palavra dupla"
    - 32 registradores de uso geral de 64 bits: x0 a x30





# RISC-V Registradores

---



- x0: o valor constante 0
- x1: endereço de retorno
- x2: apontador de pilha
- x3: ponteiro global
- x4: ponteiro de linha
- x5 - x7, x28 - x31: temporários
- x8: ponteiro do quadro
- x9, x18 - x27: registros salvos
- x10 - x11: argumentos de função / resultados
- x12 - x17: argumentos de função



# Operandos na Memória - Endianess

- MIPS/ RISC V
  - Inteiros com 32 bits
  - Memória endereçada por byte
  - Big Endian (MIPS) Little Endian (RISC V – x86)

$b_3$	$b_2$	$b_1$	$b_0$
-------	-------	-------	-------

$b_3$	$b_2$	$b_1$	$b_0$
-------	-------	-------	-------

10	$b_3$
11	$b_2$
12	$b_1$
13	$b_0$
14	
15	
16	
17	

10	$b_0$
11	$b_1$
12	$b_2$
13	$b_3$
14	
15	
16	
17	

Infra-estrutura Hardware



UNIVERSIDADE FEDERAL  
DE PERNAMBUCO

ufpe.br

# Operandos na Memória

---



- As únicas instruções que acessam a memória são as instruções de Load e Store



# Operandos Constantes Imediatos

- Valores constantes especificados na instrução  
`addi x22, x22, 4`
- Implemente o comum de forma eficiente
  - Constantes “pequenas” são comuns
  - Operandos imediatos evitam instrução de load



# Representando Instruções



- As instruções são codificadas em binário
  - Código de máquina
- Instruções RISC-V
  - Codificadas como instrução de 32 bits
  - Pequeno número de formatos e de códigos de operação (opcode),
  - números de registradores,...
- Regularidade!



Instrução	Descrição
nop	No operation
ld reg, desl(reg_base)	reg. = mem (reg_base+desl)
sd reg, desl(reg_base)	Mem(reg_base+desl) = reg
lw reg, desl(reg_base)	reg. (31:0)= mem [reg_base+desl](31:0) , extensão do sinal
sw reg, desl(reg_base)	Mem[reg_base+desl](31:0) = reg(31:0)
lh reg, desl(reg_base)	reg. (15:0)= mem [reg_base+desl](15:0) , extensão do sinal
sh reg, desl(reg_base)	Mem[reg_base+desl](15:0) = reg(15:0)
lbu reg, desl(reg_base)	reg. (7:0)= mem [reg_base+desl](7:0) , completa com zeros
sb (reg, desl(reg_base))	Mem[reg_base+desl](7:0) = reg(7:0)
add regi, regj, regk	Regi. <- Regj. + Regk
sub regi, regj, regk	Regi. <- Regj. - Regk
and regi, regj, regk	Regi. <- Regj. and Regk
addi regi, regj, cte	Regi = Regj + cte
srli regd, regs, n	Desloca regs para direita n vezes sem preservar sinal, armazena valor deslocado em regd
srai regd, regs, n	Desloca regs para dir. n vezes preservando o sinal, armazena valor deslocado em regd.
slli regd, regs, n	Desloca regs para esquerda n vezes, armazena valor deslocado em regd.
slt regi, regj, regk	Regi = 1 se (regj) < (regk) caso contrário regi= 0
slti regi, regj, immed	Regi = 1 se (regj) < extensão sinal de immed, caso contrário regi= 0
beq regi, regj, desl	PC = PC + desl*1 se regi = regj
bne regi, regj, desl	PC = PC + desl *1 se regi <> regj
bge regi, regj, desl	PC = PC + desl*1 se regi >=regj
blt regi, regj, desl	PC = PC + desl*1 se regi < regj
jal ra, end	ra = pc, pc = end, se Ra=0 não guarda PC
jalr ra, desl(reg-dst)	Ra=Pc Pc= reg-dst+desl, se Ra=0 não guarda PC
break	Para a execução do programa

# Simulador Venus



- Código fonte: <https://github.com/kvakil/venus>



# Simulador Venus



- Código fonte: <https://github.com/kvakil/venus>
- Disponível para uso online: <http://www.kvakil.me/venus/>





# Simulador Venus



- Suporta o padrão RV32IM.
  - <https://riscv.org/specifications/>
  - <https://rv8.io/isa.html>



# Simulador Venus



- Suporta o padrão RV32IM.
  - [https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISC\\_VGreenCardv8-20151013.pdf](https://www.cl.cam.ac.uk/teaching/1617/ECAD+Arch/files/docs/RISC_VGreenCardv8-20151013.pdf)

<b>Loads</b>	Load Byte	I	LB	rd,rs1,imm		
	Load Halfword	I	LH	rd,rs1,imm		
	Load Word	I	LW	rd,rs1,imm	L{D Q}	rd,rs1,imm
	Load Byte Unsigned	I	LBU	rd,rs1,imm		
	Load Half Unsigned	I	LHU	rd,rs1,imm	L{W D}U	rd,rs1,imm
<b>Stores</b>	Store Byte	S	SB	rs1,rs2,imm		
	Store Halfword	S	SH	rs1,rs2,imm		
	Store Word	S	SW	rs1,rs2,imm	S{D Q}	rs1,rs2,imm
<b>Shifts</b>	Shift Left	R	SLL	rd,rs1,rs2	SLL{W D}	rd,rs1,rs2
	Shift Left Immediate	I	SLLI	rd,rs1,shamt	SLLI{W D}	rd,rs1,shamt
	Shift Right	R	SRL	rd,rs1,rs2	SRL{W D}	rd,rs1,rs2
	Shift Right Immediate	I	SRLI	rd,rs1,shamt	SRLI{W D}	rd,rs1,shamt
	Shift Right Arithmetic	R	SRA	rd,rs1,rs2	SRA{W D}	rd,rs1,rs2
	Shift Right Arith Imm	I	SRAI	rd,rs1,shamt	SRAI{W D}	rd,rs1,shamt
<b>Arithmetic</b>	ADD	R	ADD	rd,rs1,rs2	ADD{W D}	rd,rs1,rs2
	ADD Immediate	I	ADDI	rd,rs1,imm	ADDI{W D}	rd,rs1,imm
	SUBtract	R	SUB	rd,rs1,rs2	SUB{W D}	rd,rs1,rs2
	Load Upper Imm	U	LUI	rd,imm		
	Add Upper Imm to PC	U	AUIPC	rd,imm		
<b>Logical</b>	XOR	R	XOR	rd,rs1,rs2		
	XOR Immediate	I	XORI	rd,rs1,imm		
	OR	R	OR	rd,rs1,rs2		
	OR Immediate	I	ORI	rd,rs1,imm		
	AND	R	AND	rd,rs1,rs2		
	AND Immediate	I	ANDI	rd,rs1,imm		
<b>Optional Compress</b>						
	<b>Category</b>		<b>Name</b>		<b>Fmt</b>	
<b>Loads</b>	Load Word				CL	
	Load Word SP				CI	
	Load Double				CL	
	Load Double SP				CI	
	Load Quad				CL	
	Load Quad SP				CI	



# Exemplo de Programa em Linguagem de Montagem



- Escreva o seguinte código na aba **Editor**

[www.kvakil.me/venus/](http://www.kvakil.me/venus/)

Editor

Simulator

```
1 addi a0, zero, 5
2 addi a1, zero, 10
3 addi a2, zero, 5
4 add a3,a0,a1
5 sub a3,a3,a2
6
```

```
addi a0, zero, 5
addi a1, zero, 10
addi a2, zero, 5
add a3,a0,a1
sub a3,a3,a2
```



# Exemplo de Programa em Linguagem de Montagem



- Vá para aba **Simulator** para executar o código

Editor

Simulator

Run

Step

Prev

Reset

Dump

Machine Code	Basic Code	Original Code
0x00500513	addi x10 x0 5	addi a0, zero, 5
0x00a00593	addi x11 x0 10	addi a1, zero, 10
0x00500613	addi x12 x0 5	addi a2, zero, 5
0x00b506b3	add x13 x10 x11	add a3,a0,a1
0x40c686b3	sub x13 x13 x12	sub a3,a3,a2

console output

RegistersMemory

zero	0
ra (x1)	0
sp (x2)	2147483632
gp (x3)	268435456
tp (x4)	0
t0 (x5)	0
t1 (x6)	0
t2 (x7)	0
s0 (x8)	0
s1 (x9)	0
a0 (x10)	0

Display Settings

Decimal

# Exemplo de Programa em Linguagem de Montagem



- **Original code** seria o código em uma escrita com diretivas de alto nível. (ex: zero)
- **Basic code** seria o código com as diretivas traduzidas para assembly.

Editor

Simulator

Run

Step

Prev

Reset

Dump

Machine Code	Basic Code	Original Code
0x00500513	addi x10 x0 5	addi a0, zero, 5
0x00a00593	addi x11 x0 10	addi a1, zero, 10
0x00500613	addi x12 x0 5	addi a2, zero, 5
0x00b506b3	add x13 x10 x11	add a3,a0,a1
0x40c686b3	sub x13 x13 x12	sub a3,a3,a2

console output

RegistersMemory

zero	0
ra (x1)	0
sp (x2)	2147483632
gp (x3)	268435456
tp (x4)	0
t0 (x5)	0
t1 (x6)	0
t2 (x7)	0
s0 (x8)	0
s1 (x9)	0
a0 (x10)	0

Display Settings

Decimal

Infra-estrutura Hardware

# Exemplo de Programa em Linguagem de Montagem



- Visualize as informações armazenadas nos **registradores** do RISC V

Editor

Simulator

Run

Step

Prev

Reset

Dump

Machine Code	Basic Code	Original Code
0x00500513	addi x10 x0 5	addi a0, zero, 5
0x00a00593	addi x11 x0 10	addi a1, zero, 10
0x00500613	addi x12 x0 5	addi a2, zero, 5
0x00b506b3	add x13 x10 x11	add a3,a0,a1
0x40c686b3	sub x13 x13 x12	sub a3,a3,a2

console output

Registers

Memory

zero

0

ra (x1)

0

sp (x2)

2147483632

gp (x3)

268435456

tp (x4)

0

t0 (x5)

0

t1 (x6)

0

t2 (x7)

0

s0 (x8)

0

s1 (x9)

0

a0 (x10)

0

Display Settings

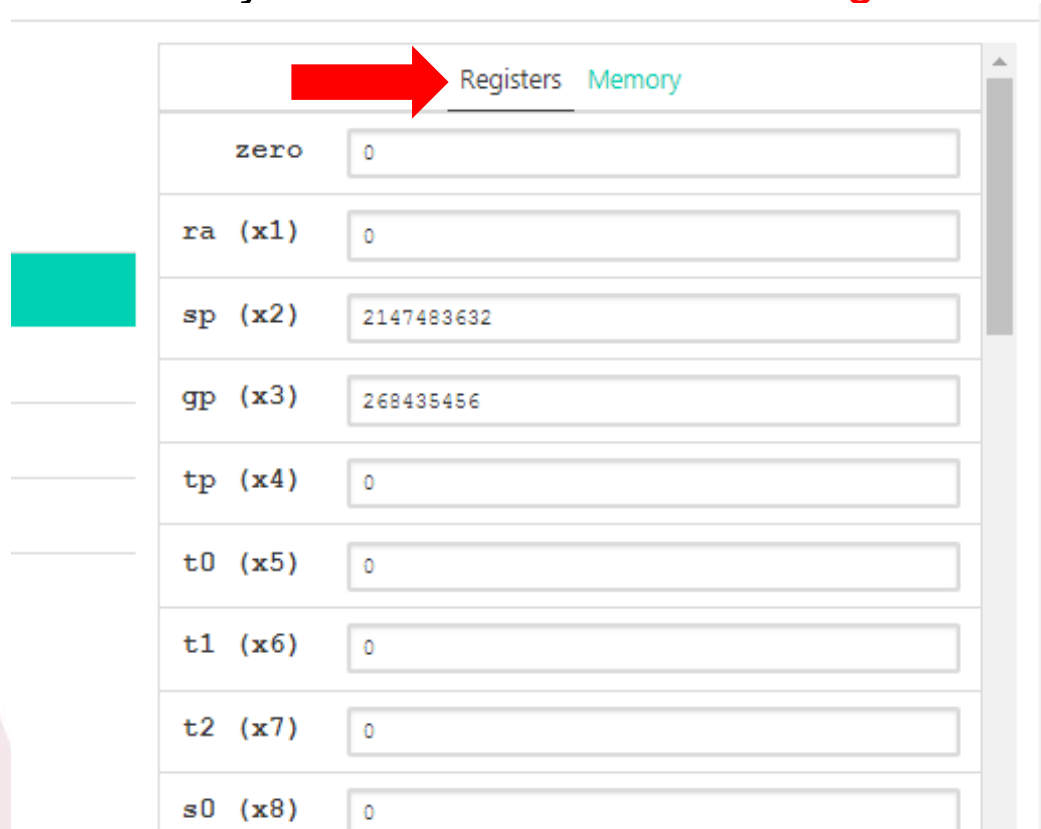
Decimal

Infra-estrutura Hardware

# Exemplo de Programa em Linguagem de Montagem



- Visualize as informações armazenadas nos **registradores** do RISC V



A screenshot of a RISC-V register file interface. A red arrow points to the "Registers" tab, which is selected. The interface shows a list of registers with their names and current values. A teal square is positioned to the left of the "sp (x2)" register.

Register Name	Value
zero	0
ra (x1)	0
sp (x2)	2147483632
gp (x3)	268435456
tp (x4)	0
t0 (x5)	0
t1 (x6)	0
t2 (x7)	0
s0 (x8)	0



# Exemplo de Programa em Linguagem de Montagem



- Defina a base numérica em que valores serão visualizados

<code>s0 (x8)</code>	<input type="text" value="0"/>
<code>s1 (x9)</code>	<input type="text" value="0"/>
<code>a0 (x10)</code>	<input type="text" value="0"/>

Display Settings






# Exemplo de Programa em Linguagem de Montagem



- Visualize as informações armazenadas na **memória** do RISC V



	Registers	Memory			
Address	+0	+1	+2	+3	
0x00000018	00	00	00	00	
0x00000014	00	00	00	00	
0x00000010	b3	86	c6	40	
0x0000000c	b3	06	b5	00	
0x00000008	13	06	50	00	
0x00000004	93	05	a0	00	
0x00000000	13	05	50	00	
-----	--	--	--	--	
-----	--	--	--	--	



# Exemplo de Programa em Linguagem de Montagem



- Visualize as informações armazenadas na **memória** do RISC V

Address	Registers Memory			
	+0	+1	+2	+3
0x00000018	00	00	00	00
0x00000014	00	00	00	00
0x00000010	b3	86	c6	40
0x0000000c	b3	06	b5	00
0x00000008	13	06	50	00
0x00000004	93	05	a0	00
0x00000000	13	05	50	00
-----	--	--	--	--
-----	--	--	--	--

Deslocamento de Bytes



# Exemplo de Programa em Linguagem de Montagem



- Vá direto para a região de memória específica

	0x0ffffffec	0	0	0
	0x0ffffffe8	0	0	0

**Jump to**

-- choose -- ▾

Up

Down

**Display Settings**

Text  
Data  
Heap  
Stack



# Exemplo de Programa em Linguagem de Montagem



- O comando **run** executará todas as instruções sem parar



Run

Step

Prev

Reset

Dump

Machine Code	Basic Code	Original Code
0x00500513	addi x10 x0 5	addi a0, zero, 5
0x00a00593	addi x11 x0 10	addi a1, zero, 10
0x00500613	addi x12 x0 5	addi a2, zero, 5
0x00b506b3	add x13 x10 x11	add a3,a0,a1
0x40c686b3	sub x13 x13 x12	sub a3,a3,a2

console output

Infra-estrutura Hardware

Registers	
zero	0
ra (x1)	0
sp (x2)	2147483632
gp (x3)	268435456
tp (x4)	0
t0 (x5)	0
t1 (x6)	0
t2 (x7)	0
s0 (x8)	0
s1 (x9)	0
a0 (x10)	0
Display Settings	
Decimal	

# Debugging



- Se alguma instrução tiver **breakpoint** o comando run parará nessa instrução
- Ao clicar na instrução você adicionará o breakpoint.

Machine Code	Basic Code	Original Code
0x00500513	addi x10 x0 5	addi a0, zero, 5
0x00a00593	addi x11 x0 10	addi a1, zero, 10
0x00500613	addi x12 x0 5	addi a2, zero, 5
0x00b506b3	add x13 x10 x11	add a3,a0,a1
0x40c686b3	sub x13 x13 x12	sub a3,a3,a2



# Debugging



- O comando **step** executa instrução por instrução

Run

Step

Prev

Reset

Dump

Machine Code	Basic Code	Original Code
0x00500513	addi x10 x0 5	addi a0, zero, 5
0x00a00593	addi x11 x0 10	addi a1, zero, 10
0x00500613	addi x12 x0 5	addi a2, zero, 5
0x00b506b3	add x13 x10 x11	add a3,a0,a1
0x40c686b3	sub x13 x13 x12	sub a3,a3,a2

console output

Registers

Memory

zero

0x00000000

ra (x1)

0x00000000

sp (x2)

0x7fffffff0

gp (x3)

0x10000000

tp (x4)

0x00000000

t0 (x5)

0x00000000

t1 (x6)

0x00000000

t2 (x7)

0x00000000

s0 (x8)

0x00000000

s1 (x9)

0x00000000

a0

0x00000005

(x10)



# Debugging



- Valores dos registradores são atualizados de acordo com a instrução

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0x00500513	addi x10 x0 5	addi a0, zero, 5
0x00a00593	addi x11 x0 10	addi a1, zero, 10
0x00500613	addi x12 x0 5	addi a2, zero, 5
0x00b506b3	add x13 x10 x11	add a3,a0,a1
0x40c686b3	sub x13 x13 x12	sub a3,a3,a2

a0  
(x10) 0x00000005

Registers Memory

Register	Value
zero	0x00000000
ra (x1)	0x00000000
sp (x2)	0x7fffffff0
gp (x3)	0x10000000
tp (x4)	0x00000000
t0 (x5)	0x00000000
t1 (x6)	0x00000000
t2 (x7)	0x00000000
s0 (x8)	0x00000000
s1 (x9)	0x00000000
a0 (x10)	0x00000005



# Debugging



- É possível voltar cada passo

Editor Simulator

Run Step Prev Reset Dump

Machine Code	Basic Code	Original Code
0x00500513	addi x10 x0 5	addi a0, zero, 5
0x00a00593	addi x11 x0 10	addi a1, zero, 10
0x00500613	addi x12 x0 5	addi a2, zero, 5
0x00b506b3	add x13 x10 x11	add a3,a0,a1
0x40c686b3	sub x13 x13 x12	sub a3,a3,a2

Registers Memory

zero	0
ra (x1)	0
sp (x2)	2147483632
gp (x3)	268435456
tp (x4)	0
t0 (x5)	0
t1 (x6)	0





# Debugging



- Ou voltar para o começo

Editor

Simulator

Run

Step

Prev

Reset

Dump

Machine Code	Basic Code	Original Code
0x00500513	addi x10 x0 5	addi x0, zero, 5
0x00a00593	addi x11 x0 10	addi x1, zero, 10
0x00500613	addi x12 x0 5	addi a2, zero, 5
0x00b506b3	add x13 x10 x11	add a3,a0,a1
0x40c686b3	sub x13 x13 x12	sub a3,a3,a2

Registers

Memory

zero	0
ra (x1)	0
sp (x2)	2147483632
gp (x3)	268435456
tp (x4)	0
t0 (x5)	0
t1 (x6)	0



# Console output



- Visualize os dados no console

[Run](#) [Step](#) [Prev](#) [Reset](#) [Dump](#)

Machine Code	Basic Code	Original Code
0x00100513	addi x10 x0 1	addi a0 x0 1 # print_int ecall
0x02a00593	addi x11 x0 42	addi a1 x0 42 # integer 42
0x00000073	ecall	ecall


42



# Console output



- Para visualizar:
  - Carregue a ID (1 para inteiro e 4 para string) no registrador a0 e carregue o argumento em a1 – a7.



ID ( a0 )	Name	Description
1	print_int	prints integer in a1
4	print_string	prints the null-terminated string whose address is in a1
9	sbrk	allocates a1 bytes on the heap, returns pointer to start in a0
10	exit	ends the program
11	print_character	prints ASCII character in a1
17	exit2	ends the program with return code in a1



# Console output

---



- Para visualizar:
  - Em seguida, chame a instrução `ecall` (environment call)



# Console output

---



- Exemplo para testar:
  - Escreva este código e em seguida clique em **run**

```
addi a0 x0 1      # print_int ecall
addi a1 x0 42     # integer 42
ecall
```



# Console output



- Visualize os dados no console
  - Deverá aparecer o valor 42 no console

Machine Code	Basic Code	Original Code
0x00100513	addi x10 x0 1	addi a0 x0 1 # print_int ecall
0x02a00593	addi x11 x0 42	addi a1 x0 42 # integer 42
0x00000073	ecall	ecall

42



# Console output



- Para visualizar:
  - Mas detalhes sobre environment call:
    - <https://github.com/kvakil/venus/wiki/Environmental-Calls>

ID ( a0 )	Name	Description
1	print_int	prints integer in a1
4	print_string	prints the null-terminated string whose address is in a1
9	sbrk	allocates a1 bytes on the heap, returns pointer to start in a0
10	exit	ends the program
11	print_character	prints ASCII character in a1
17	exit2	ends the program with return code in a1



# Outro exemplo...carregando dado da memória e loop

Escreva o seguinte código no espaço Editor

.data

a: .word 4

.text

lw a1,a

addi a0, zero, 0

loop:

addi a0,a0,1

bne a0,a1,loop

```
1 .data
2 a: .word 4
3 .text
4 lw a1,a
5 addi a0, zero, 0
6 loop:
7 addi a0,a0,1
8 bne a0,a1,loop
```

Editor

Simulator





# Exemplo de Programa em Linguagem de Montagem



- Diretivas suportadas
  - <https://github.com/kvakil/venus/wiki/Assembler-Directives>

Directive	Effect
<code>.data</code>	Store subsequent items in the [[static segment
<code>.text</code>	Store subsequent instructions in the [[text segment
<code>.byte</code>	Store listed values as 8-bit bytes.
<code>.ascii</code>	Store subsequent string in the data segment and add null-terminator.
<code>.word</code>	Store listed values as unaligned 32-bit words.
<code>.globl</code>	Makes the given label global.
<code>.float</code>	Reserved.
<code>.double</code>	Reserved.
<code>.align</code>	Reserved.



# Outro exemplo...carregando dado da memória e loop



A memória de dados começa  
no endereço 0x10000000  
Que é o mesmo apontado  
pelo registrador gp

	0x1000001c	0	0	0	0
	0x10000018	0	0	0	0
	0x10000014	0	0	0	0
	0x10000010	0	0	0	0
	0x1000000c	0	0	0	0
	0x10000008	0	0	0	0
	0x10000004	0	0	0	0
	0x10000000	4	0	0	0
gp					

Jump to -- choose -- Up Down



# Outro exemplo...carregando dado da memória e loop



Instrução lw a1,a terminou  
registrador x11 (ou a1) agora está com o valor 4

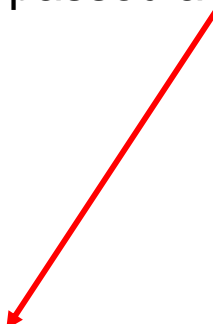
	0x00000000
a0 (x10)	0x00000000
a1 (x11)	0x00000004
a2 (x12)	0x00000000



# Outro exemplo...carregando dado da memória e loop



Após executar o run o registrador x10 (ou a0) passou a ter o valor 4



a0 (x10)	0x00000004
a1 (x11)	0x00000004



# Outros tipos de dados

---



- Para declarar uma string:
  - Nome: .asciiz “*meu nome é..*”
  - Exemplo:
    - msg:.asciiz"1-2-3-4-5"
    - lb x5, 0(gp) #colocando o valor que esta na memoria



# Outros tipos de dados

---

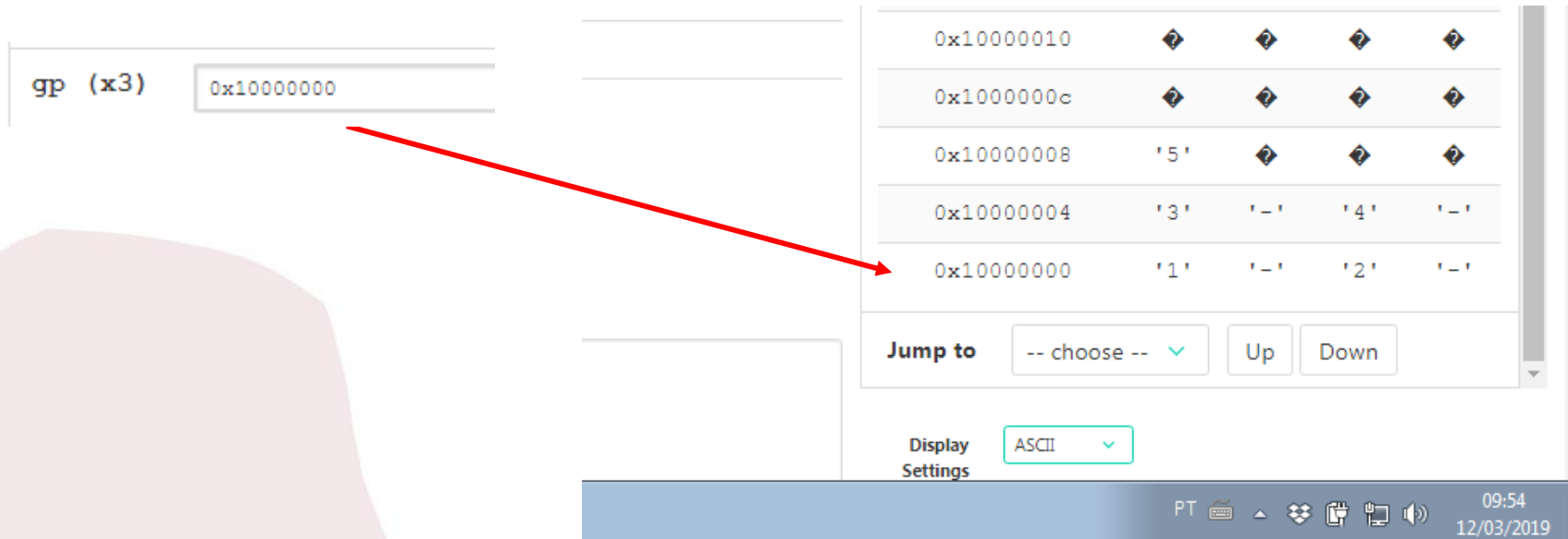


- Escrevam o seguinte código:
- .data
- msg:.asciiz "1-2-3-4-5"
- .text
- lb a2,(0)gp
- lb a3,(1)gp
- lb a4,(2)gp
- lb a5,(3)gp
- lb a6,(4)gp
- lb a7,(5)gp



# Outros tipos de dados

- Observe que o registrador gp aponta para o início da string na memória



gp (x3) 0x10000000

0x10000010	?	?	?	?
0x1000000c	?	?	?	?
0x10000008	'5'	?	?	?
0x10000004	'3'	'-'	'4'	'-'
0x10000000	'1'	'-'	'2'	'-'

Jump to -- choose -- Up Down

Display Settings ASCII

PT 09:54 12/03/2019

# Outros tipos de dados

- Execute run e verifique os valores da string nos registradores

a2 (x12)	'1'
a3 (x13)	'-'
a4 (x14)	'2'
a5 (x15)	'-'
a6 (x16)	'3'
a7 (x17)	'-'



# Outros tipos de dados



- Outra maneira similar

- .data
- msg:.asciiz "1-2-3-4-5"
- .text
- la a1, msg
- lb a2,(0)a1
- lb a3,(1)a1
- lb a4,(2)a1
- lb a5,(3)a1
- lb a6,(4)a1

A instrução la  
Carrega o endereço base de msg



# Outros tipos de dados

---



- É possível inicializar dados na memória usando diretivas como as presentes aqui
- Por exemplo, para declarar uma variável inteira a:
  - a: .word *valor*
  - String nome:
    - Nome: .string “*meu nome é..*”
    - Nome: .asciz “*meu nome é..*”



# Outros tipos de dados

---



- Por exemplo, para declarar uma variável inteira a:
  - a: .word *valor*
  - Exemplo: a: .word 10  
                  b: .word 20



# Outros tipos de dados

---



- Para declarar uma string:
  - Nome: `.string "meu nome é.."`
  - Nome: `.asciz "meu nome é.."`
  - Exemplo:
    - `msg:.string "1-2-3-4-5"`
    - `lb x5, 0(gp) #colocando o valor que esta na memoria`



# Outros tipos de dados

---



- Escrevam o seguinte código:
- .data
- msg:.string "1-2-3-4-5"
- .text
- lb a2,(0)gp
- lb a3,(1)gp
- lb a4,(2)gp
- lb a5,(3)gp
- lb a6,(4)gp
- lb a7,(5)gp



# Outros tipos de dados

Directive	Arguments	Description
<code>.2byte</code>		16-bit comma separated words (unaligned)
<code>.4byte</code>		32-bit comma separated words (unaligned)
<code>.8byte</code>		64-bit comma separated words (unaligned)
<code>.half</code>		16-bit comma separated words (naturally aligned)
<code>.word</code>		32-bit comma separated words (naturally aligned)
<code>.dword</code>		64-bit comma separated words (naturally aligned)
<code>.byte</code>		8-bit comma separated words

<code>.dtprelword</code>		64-bit thread local word
<code>.dtprelword</code>		32-bit thread local word
<code>.sleb128</code>	expression	signed little endian base 128, DWARF
<code>.uleb128</code>	expression	unsigned little endian base 128, DWARF
<code>.asciz</code>	"string"	emit string (alias for <code>.string</code> )
<code>.string</code>	"string"	emit string
<code>.incbin</code>	"filename"	emit the included file as a binary sequence of octets
<code>.zero</code>	integer	zero bytes

# Atividade Prática

---



- Responder questões 1, 2 e 3 da lista disponível no site até às 23:59 de hoje (23/08).
- Questões 4 – 6 até o dia (03/09) às 23:59.

