Universidade Tecnológica Federal do Paraná – UTFPR
Bacharelado em Ciência da Computação

BCC33B – Arquitetura e Organização de Computadores

Prof. Rogério A. Gonçalves
rogerioag@utfpr.edu.br

Aula 005

Aula de Hoje:

Assembly: Formato Geral de Instruções

Arquitetura do Microprocessador MIPS

Arquitetura interna do microprocessador

Registradores

Barramentos

Conjunto de Instruções

Formatos de Instruções

Tipos de Instruções

Modos de Endereçamento

Linguagem *Assembly*

Instruções Lógicas

- **and, or, xor, nor**
  - **and: útil para usar máscara de bits**
    - » Extrair o byte menos significativo de uma word:

```
0xF234012F AND 0x000000FF = 0x0000002F
```

  - **or: útil para combinar bits**
    - » Combinar 0xF2340000 com 0x000012BC:

```
0xF2340000 OR 0x000012BC = 0xF23412BC
```

  - **nor: útil para inverter bits:**
    - » A NOR $0 = NOT A
- **andi, ori, xori**
  - **O imediato de 16-bit é zero-extended (*não* sign-extended)**

Instruções Lógicas

Source Registers

$s1	1111	1111	1111	1111	0000	0000	0000	0000
$s2	0100	0110	1010	0001	1111	0000	1011	0111

Assembly Code	Result								
`and $s3, $s1, $s2`	$s3	0100	0110	1010	0001	0000	0000	0000	0000
`or  $s4, $s1, $s2`	$s4	1111	1111	1111	1111	1111	0000	1011	0111
`xor $s5, $s1, $s2`	$s5	1011	1001	0101	1110	1111	0000	1011	0111
`nor $s6, $s1, $s2`	$s6	0000	0000	0000	0000	0000	1111	0100	1000

Instruções Lógicas

Source Values

$s1	0000	0000	0000	0000	0000	0000	1111	1111
imm	0000	0000	0000	0000	1111	1010	0011	0100

zero-extended

Assembly Code	Result								
`andi $s2, $s1, 0xFA34`	$s2	0000	0000	0000	0000	0000	0000	0011	0100
`ori $s3, $s1, 0xFA34`	$s3	0000	0000	0000	0000	1111	1010	1111	1111
`xori $s4, $s1, 0xFA34`	$s4	0000	0000	0000	0000	1111	1010	1100	1011

Instruções de Deslocamento (Shift)

- **`sll`: shift left logical**
  - Exemplo: **`sll $t0, $t1, 5  # $t0 <= $t1 << 5`**
- **`srl`: shift right logical**
  - Exemplo : **`srl $t0, $t1, 5  # $t0 <= $t1 >> 5`**
- **`sra`: shift right arithmetic**
  - Exemplo : **`sra $t0, $t1, 5  # $t0 <= $t1 >>> 5`**

Variable shift instructions:

- **`sllv`: shift left logical variable**
  - Exemplo : **`sll $t0, $t1, $t2 # $t0 <= $t1 << $t2`**
- **`srlv`: shift right logical variable**
  - Exemplo : **`srl $t0, $t1, $t2 # $t0 <= $t1 >> $t2`**
- **`srav`: shift right arithmetic variable**
  - Exemplo **: `sra $t0, $t1, $t2 # $t0 <= $t1 >>> $t2`**

Instruções de Deslocamento (Shift)

Assembly Code / Field Values

Assembly Code	op	rs	rt	rd	shamt	funct
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
`sll $t0, $s1, 2`	0	0	17	8	2	0
`srl $s2, $s1, 2`	0	0	17	18	2	2
`sra $s3, $s1, 2`	0	0	17	19	2	3

Machine Code

op	rs	rt	rd	shamt	funct	
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
000000	00000	10001	01000	00010	000000	(0x00114080)
000000	00000	10001	10010	00010	000010	(0x00119082)
000000	00000	10001	10011	00010	000011	(0x00119883)

Gerando Constantes

- **Constantes de 16-bit usando `addi`:**

High-level code

```
// int is a 32-bit signed word
int a = 0x4f3c;
```

MIPS assembly code

```
# $s0 = a
addi $s0, $0, 0x4f3c
```

- **Constantes de 32-bit usando *load upper immediate* (`lui`) e `ori`:**

  (`lui` carrega o imediato de 16-bit na metade mais significativa do registrador seta a menos significativa com 0.)

High-level code

```
int a = 0xFEDC8765;
```

MIPS assembly code

```
# $s0 = a
lui $s0, 0xFEDC
ori $s0, $s0, 0x8765
```

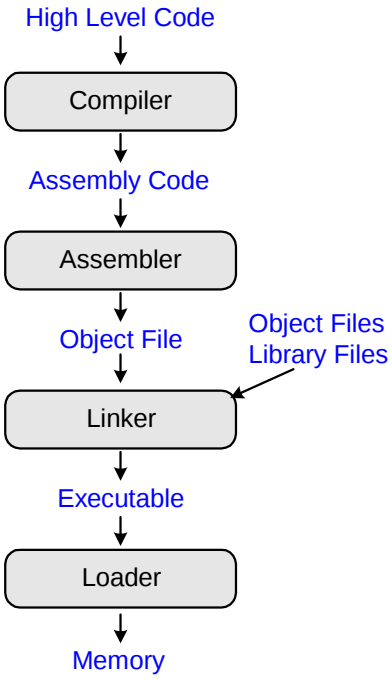
Multiplicação e Divisão

- **Registradores especiais: `lo`, `hi`**
- **Multiplicação 32 × 32 bit, resultado de 64 bit**
  - **`mult $s0, $s1`**
  - **Resultado em `hi`, `lo`**
- **Divisão 32-bit, quociente de 32-bit, resto de 32-bit**
  - **`div $s0, $s1`**
  - **Quociente em `lo`**
  - **Resto em `hi`**

O que Deve ser Armazenado na Memória

- **Instruções**
  - **(também chamado: *text*)**
- **Dado**
  - **Global/estático:** alocado antes de começar a execução
  - **Dinâmico:** alocado pelo programa em execução
- **Qual o tamanho da memória?**
  - **No máximo 2^{32} = 4 gigabytes (4 GB)**
  - **A partir do endereço 0x00000000 ao 0xFFFFFFFF**

Executando um Programa



Exemplo: Programa em C

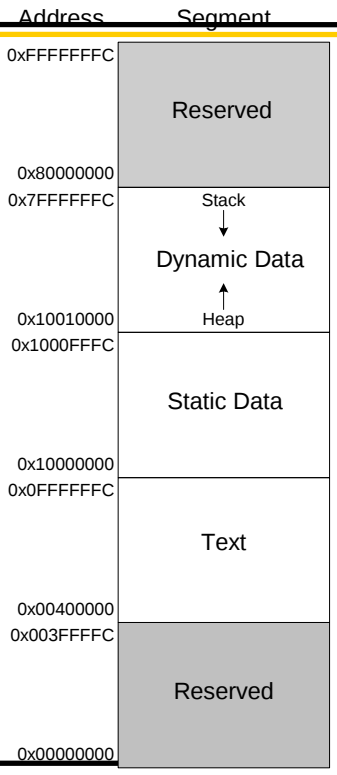
```
int f, g, y;  // global variables

int main(void) {
   f = 2;
   g = 3;
   y = sum(f, g);

   return y;
}

int sum(int a, int b) {
   return (a + b);
}
```

Exemplo: Programa em Assembly

```c
int f, g, y;  // global variables

int main(void) {
  f = 2;
  g = 3;
  y = sum(f, g);

  return y;
}

int sum(int a, int b) {
  return (a + b);
}
```

```
.data
f:  .word   0x00000000
g:  .word   0x00000000
y:  .word   0x00000000
.text
main:
  addi $sp, $sp, -4   # stack frame
  sw   $ra, 0($sp)    # store $ra
  addi $a0, $0, 2     # $a0 = 2
  sw   $a0, f         # f = 2
  addi $a1, $0, 3     # $a1 = 3
  sw   $a1, g         # g = 3
  jal  sum            # call sum
  sw   $v0, y         # y = sum()
  lw   $ra, 0($sp)    # restore $ra
  addi $sp, $sp, 4    # restore $sp
  jr   $ra            # return to OS
sum:
  add  $v0, $a0, $a1  # $v0 = a + b
  jr   $ra            # return
```

Mapa de Memória MIPS



Exemplo: Tabela de Símbolos

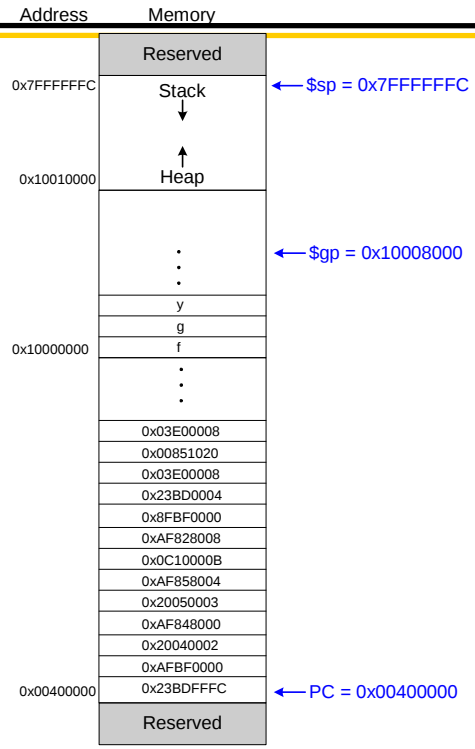
Symbol	Address
f	0x10000000
g	0x10000004
y	0x10000008
main	0x00400000
sum	0x0040002C

Exemplo: Programa Executável

Executable file header	Text Size	Data Size
	0x34 (52 bytes)	0xC (12 bytes)
Text segment	**Address**	**Instruction**
	0x00400000	0x23BDFFFC
	0x00400004	0xAFBF0000
	0x00400008	0x20040002
	0x0040000C	0xAF848000
	0x00400010	0x20050003
	0x00400014	0xAF858004
	0x00400018	0x0C10000B
	0x0040001C	0xAF828008
	0x00400020	0x8FBF0000
	0x00400024	0x23BD0004
	0x00400028	0x03E00008
	0x0040002C	0x00851020
	0x00400030	0x03E0008
Data segment	**Address**	**Data**
	0x10000000	f
	0x10000004	g
	0x10000008	y

```
addi $sp, $sp, -4
sw   $ra, 0 ($sp)
addi $a0, $0, 2
sw   $a0, 0x8000 ($gp)
addi $a1, $0, 3
sw   $a1, 0x8004 ($gp)
jal  0x0040002C
sw   $v0, 0x8008 ($gp)
lw   $ra, 0 ($sp)
addi $sp, $sp, -4
jr   $ra
add  $v0, $a0, $a1
jr   $ra
```

Exemplo: Programa na Memória



Pseudo Instruções

Pseudoinstruction	MIPS Instructions
`li $s0, 0x1234AA77`	`lui $s0, 0x1234` `ori $s0, 0xAA77`
`mul $s0, $s1, $s2`	`mult $s1, $s2` `mflo $s0`
`clear $t0`	`add $t0, $0, $0`
`move $s1, $s2`	`add $s2, $s1, $0`
`nop`	`sll $0, $0, 0`

Pseudo Instruções

Nome	Sintaxe	Tradução Inst. Reais	Significado
Move	move $rt,$rs	add $rt,$rs,$zero	R[rt]=R[rs]
Clear	clear $rt	add $rt,$zero,$zero	R[rt]=0
Not	not $rt, $rs	nor $rt, $rs, $zero	R[rt]=~R[rs]
Load Address	la $rd, LabelAddr	lui $rd, LabelAddr[31:16]; ori $rd,$rd, LabelAddr[15:0]	$rd = Label Address
Load Immediate	li $rd, IMMED[31:0]	lui $rd, IMMED[31:16]; ori $rd,$rd, IMMED[15:0]	$rd = 32 bit Immediate value
Branch unconditionally	b Label	beq $zero,$zero,Label	PC=Label
Branch and link	bal Label	bgezal $zero,Label	R[31]=PC+8; PC=Label
Branch if greater than	bgt $rs,$rt,Label	slt $at,$rt,$rs; bne $at, $zero,Label	if(R[rs]>R[rt]) PC=Label
Branch if less than	blt $rs,$rt,Label	slt $at,$rs,$rt; bne $at,$zero,Label	if(R[rs]<R[rt]) PC=Label
Branch if greater than or equal	bge $rs,$rt,Label	slt $at,$rs,$rt; beq $at,$zero,Label	if(R[rs]>=R[rt]) PC=Label
Branch if less than or equal	ble $rs,$rt,Label	slt $at,$rt,$rs; beq $at,$zero,Label	if(R[rs]<=R[rt]) PC=Label
Branch if greater than unsigned	bgtu $rs,$rt,Label		if(R[rs]>R[rt]) PC=Label
Branch if greater than zero	bgtz $rs,Label		if(R[rs]>0) PC=Label
Branch if equal to zero	beqz $rs,Label		if(R[rs]==0) PC=Label
Multiplies and returns only first 32 bits	mul $d, $s, $t	mult $s, $t; mflo $d	$d = $s * $t
Divides and returns quotient	div $d, $s, $t	div $s, $t; mflo $d	$d = $s / $t
Divides and returns remainder	rem $d, $s, $t	div $s, $t; mfhi $d	$d = $s % $t

Instruções

- **Soma e subtração**
  - **Signed: add, addi, sub**
    - » Executa a mesma operação que a versão unsigned
    - » Porém o processador gera exceção se overflow
  - **Unsigned: addu, addiu, subu**
    - » O processador não gera exceção se overflow
    - » Nota: `addiu` sign-extends o imediato
- **Multiplicação e Divisão**
  - » Signed: `mult`, `div`
  - » Unsigned: `multu`, `divu`
- **Set Less Than**
  - » Signed: `slt`, `slti`
  - » Unsigned: `sltu`, `sltiu`

Nota: `sltiu` sign-extends o imediato antes da comparação

Instruções

- **Loads**
  - **Signed:**
    - » Sign-extends para criar o valor de 32-bit
    - » Load halfword: `lh`
    - » Load byte: `lb`
  - **Unsigned:** **`addu`, `addiu`, `subu`**
    - » Zero-extends para criar o valor de 32-bit
    - » Load halfword unsigned: `lhu`
    - » Load byte: `lbu`

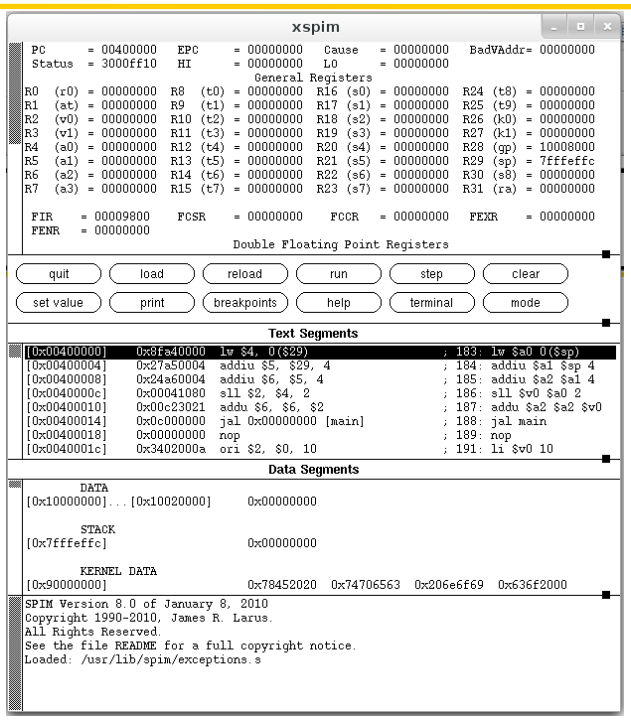
Ponto-Flutuante

- **Floating-point coprocessor (Coprocessor 1)**
- **32 registradores de 32-bit (`$f0` - `$f31`)**
- **Valores precisão dupla são mantidos em dois *floating point registers***
  - **e.g., `$f0` e `$f1`, `$f2` e `$f3`, etc.**
  - **Assim, os registradores de dupla precisão floating point são: `$f0`, `$f2`, `$f4`, etc.**

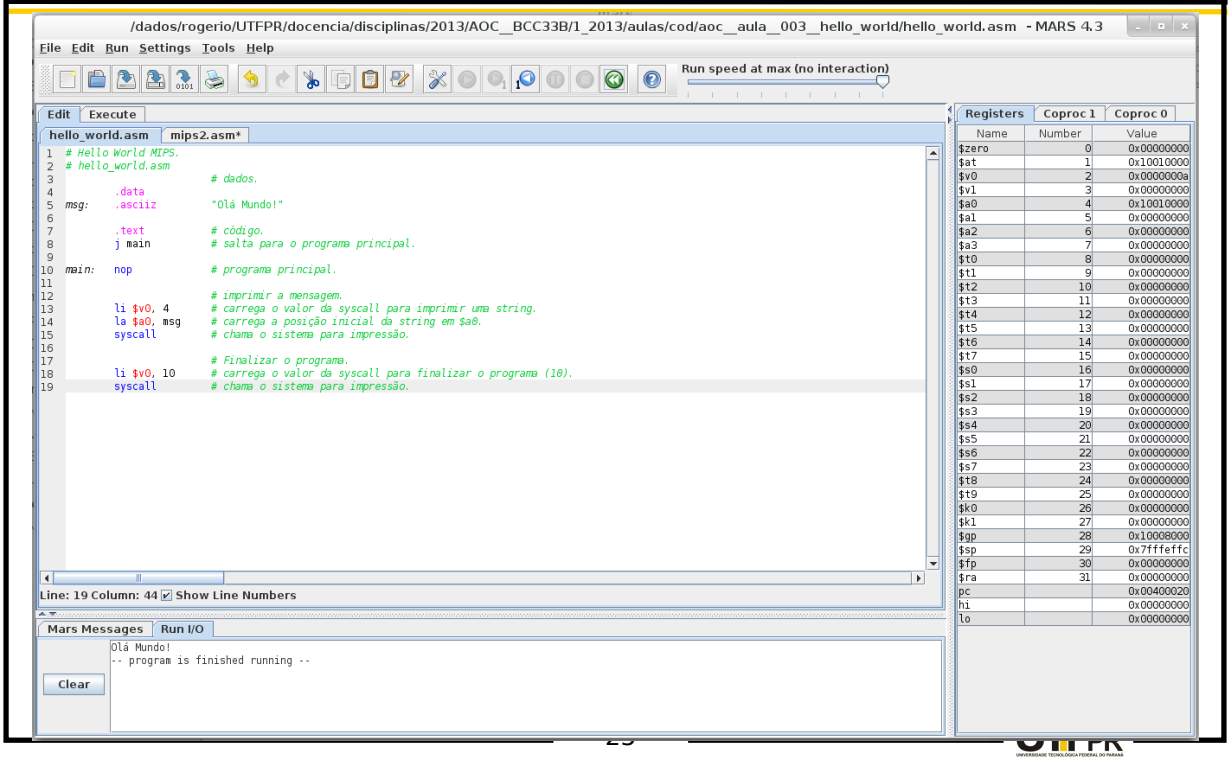
Ponto-Flutuante

Name	Register Number	Usage
$fv0 - $fv1	0, 2	return values
$ft0 - $ft3	4, 6, 8, 10	temporary variables
$fa0 - $fa1	12, 14	procedure arguments
$ft4 - $ft8	16, 18	temporary variables
$fs0 - $fs5	20, 22, 24, 26, 28, 30	saved variables

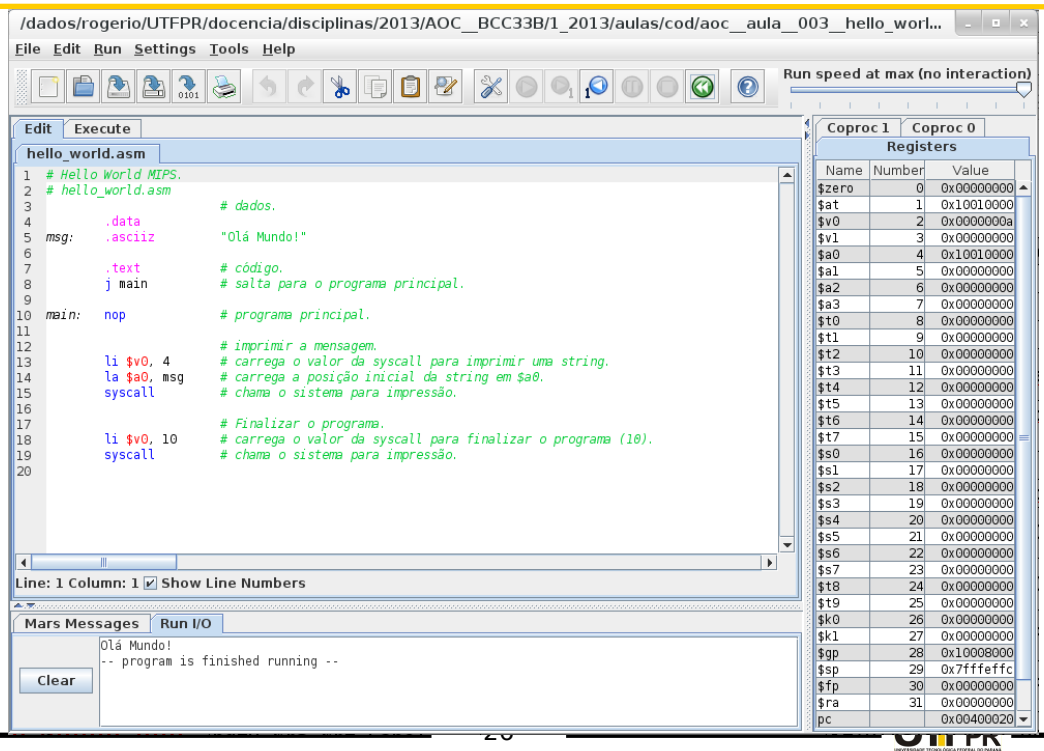
Simulador SPIM



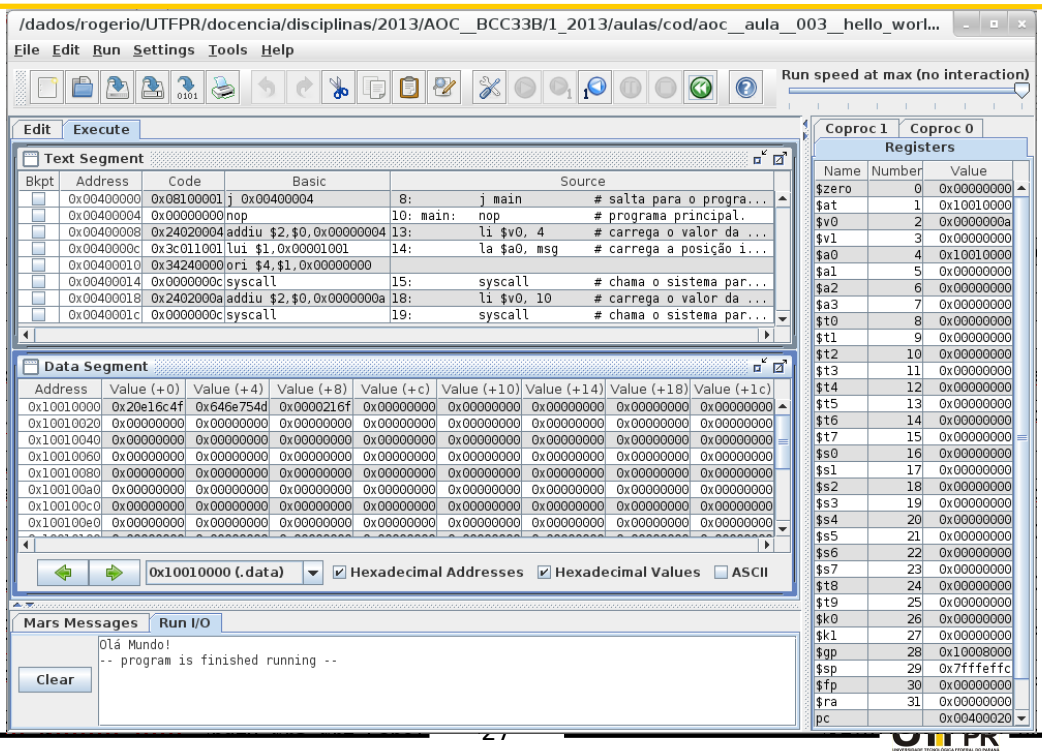
Simulador MARS



Hello World no MARS



Hello World no MARS



Diversão para Casa

- **Download do Simulador MARS**
- **Testar as instruções da aula**
- **Verificar alterações dos conteúdos dos registradores e memória**

Exercícios

- **Implemente os programas em A*ssembly* do MIPS para resolver as expressões:**

  a) a = b + c

  b) f = (g + h) – (i + j)

Exercícios

Supondo que h seja associado com o registrador *$s2* e o endereço base do *Array A* armazenado em *$s3*. Qual o código MIPS para o comando A[12] = h + A[8]?

Exercícios

Suponha que h seja associado com o registrador $s2 e o endereço base do array A armazenado em $s3. Qual o código MIPS para o comando A[12] = h + A[8];?

Solução

```
lw    $t0,32($s3)  #  $t0 recebe A[8]
add   $t0,$s2,$t0  #  $t0 recebe h + A[8]
sw    $t0,48($s3)  # armazena o resultado em A[12]
```

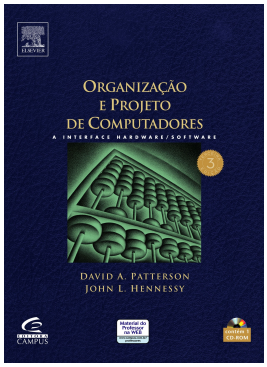

Exercícios

- **Qual o código MIPS para carregar uma constante de 32 bits no registrador $s0 ?**
  0000 0000 0011 1101 0000 1001 0000 0000

Solução

```
lui   $s0,61          # 61_10 = 0000 0000 0011 1101_2
ori   $s0,$s0,2304    # 2304_10 = 0000 1001 0000 0000_2

0000 0000 0011 1101 0000 1001 0000 0000_2
```

Leitura Recomendada



Capítulo 2

PATTERSON, David A.; HENNESSY, John L. ***Organização e projeto de computadores: a interface hardware/software.*** Rio de Janeiro, RJ: Elsevier, 2005. 484 p. ISBN 9788535215212.

Resumo da Aula de Hoje

<u>Tópicos mais importantes:</u>

Linguagem Assembly

Microprocessador MIPS

Registradores

Formatos de Instruções

Tipos de Instruções

Modos de Endereçamento

Universidade Tecnológica Federal do Paraná – UTFPR
Bacharelado em Ciência da Computação

BCC33B – Arquitetura e Organização de Computadores

Prof. Rogério A. Gonçalves
rogerioag@utfpr.edu.br

Aula 005

Aula de Hoje:

Assembly: Formato Geral de Instruções

Arquitetura do Microprocessador MIPS

Arquitetura interna do microprocessador

Registradores

Barramentos

Conjunto de Instruções

Formatos de Instruções

Tipos de Instruções

Modos de Endereçamento

Linguagem *Assembly*

Instruções Lógicas

- **and, or, xor, nor**
  - **and: útil para usar máscara de bits**
    - » Extrair o byte menos significativo de uma word:

```
0xF234012F AND 0x000000FF = 0x0000002F
```

  - **or: útil para combinar bits**
    - » Combinar 0xF2340000 com 0x000012BC:

```
0xF2340000 OR 0x000012BC = 0xF23412BC
```

  - **nor: útil para inverter bits:**
    - » A NOR $0 = NOT A
- **andi, ori, xori**
  - **O imediato de 16-bit é zero-extended (*não* sign-extended)**

Instruções Lógicas

Source Registers

$s1	1111	1111	1111	1111	0000	0000	0000	0000
$s2	0100	0110	1010	0001	1111	0000	1011	0111

Assembly Code	Result								
`and $s3, $s1, $s2`	$s3	0100	0110	1010	0001	0000	0000	0000	0000
`or  $s4, $s1, $s2`	$s4	1111	1111	1111	1111	1111	0000	1011	0111
`xor $s5, $s1, $s2`	$s5	1011	1001	0101	1110	1111	0000	1011	0111
`nor $s6, $s1, $s2`	$s6	0000	0000	0000	0000	0000	1111	0100	1000

Instruções Lógicas

Source Values

$s1	0000	0000	0000	0000	0000	0000	1111	1111
imm	0000	0000	0000	0000	1111	1010	0011	0100

zero-extended

Assembly Code	Result								
`andi $s2, $s1, 0xFA34`	$s2	0000	0000	0000	0000	0000	0000	0011	0100
`ori $s3, $s1, 0xFA34`	$s3	0000	0000	0000	0000	1111	1010	1111	1111
`xori $s4, $s1, 0xFA34`	$s4	0000	0000	0000	0000	1111	1010	1100	1011

Instruções de Deslocamento (Shift)

- **`sll`: shift left logical**
  - Exemplo: **`sll $t0, $t1, 5  # $t0 <= $t1 << 5`**
- **`srl`: shift right logical**
  - Exemplo : **`srl $t0, $t1, 5  # $t0 <= $t1 >> 5`**
- **`sra`: shift right arithmetic**
  - Exemplo : **`sra $t0, $t1, 5  # $t0 <= $t1 >>> 5`**

Variable shift instructions:

- **`sllv`: shift left logical variable**
  - Exemplo : **`sll $t0, $t1, $t2 # $t0 <= $t1 << $t2`**
- **`srlv`: shift right logical variable**
  - Exemplo : **`srl $t0, $t1, $t2 # $t0 <= $t1 >> $t2`**
- **`srav`: shift right arithmetic variable**
  - Exemplo **: `sra $t0, $t1, $t2 # $t0 <= $t1 >>> $t2`**

Instruções de Deslocamento (Shift)

Assembly Code / Field Values

Assembly Code	op	rs	rt	rd	shamt	funct
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
`sll $t0, $s1, 2`	0	0	17	8	2	0
`srl $s2, $s1, 2`	0	0	17	18	2	2
`sra $s3, $s1, 2`	0	0	17	19	2	3

Machine Code

op	rs	rt	rd	shamt	funct	
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	
000000	00000	10001	01000	00010	000000	(0x00114080)
000000	00000	10001	10010	00010	000010	(0x00119082)
000000	00000	10001	10011	00010	000011	(0x00119883)

Gerando Constantes

- **Constantes de 16-bit usando `addi`:**

High-level code

```
// int is a 32-bit signed word
int a = 0x4f3c;
```

MIPS assembly code

```
# $s0 = a
addi $s0, $0, 0x4f3c
```

- **Constantes de 32-bit usando *load upper immediate* (`lui`) e `ori`:**

  (`lui` carrega o imediato de 16-bit na metade mais significativa do registrador seta a menos significativa com 0.)

High-level code

```
int a = 0xFEDC8765;
```

MIPS assembly code

```
# $s0 = a
lui $s0, 0xFEDC
ori $s0, $s0, 0x8765
```

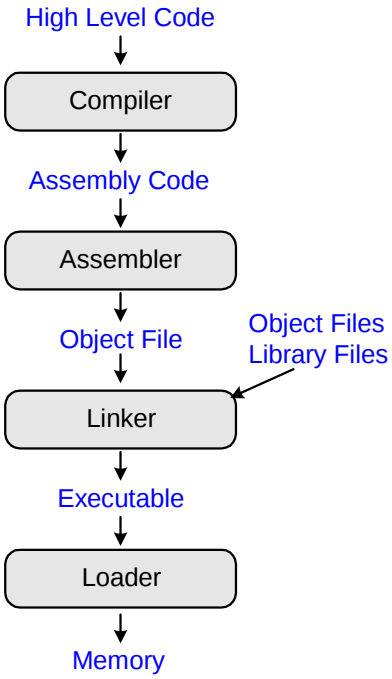
Multiplicação e Divisão

- **Registradores especiais: `lo`, `hi`**
- **Multiplicação 32 × 32 bit, resultado de 64 bit**
  - **`mult $s0, $s1`**
  - **Resultado em `hi`, `lo`**
- **Divisão 32-bit, quociente de 32-bit, resto de 32-bit**
  - **`div $s0, $s1`**
  - **Quociente em `lo`**
  - **Resto em `hi`**

O que Deve ser Armazenado na Memória

- **Instruções**
  - **(também chamado: *text*)**
- **Dado**
  - **Global/estático:** alocado antes de começar a execução
  - **Dinâmico:** alocado pelo programa em execução
- **Qual o tamanho da memória?**
  - **No máximo 2^{32} = 4 gigabytes (4 GB)**
  - **A partir do endereço 0x00000000 ao 0xFFFFFFFF**

Executando um Programa



Exemplo: Programa em C

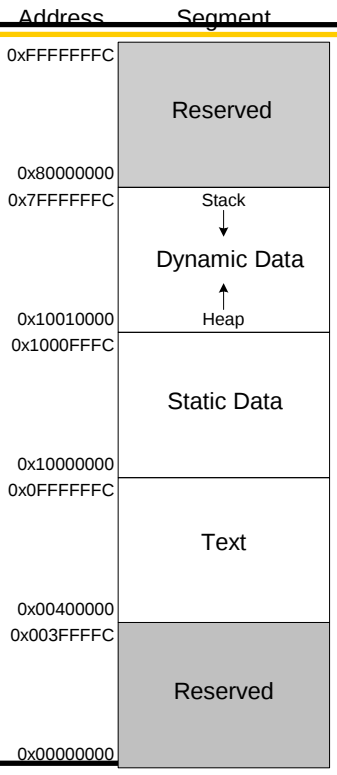
```
int f, g, y;  // global variables

int main(void) {
   f = 2;
   g = 3;
   y = sum(f, g);

   return y;
}

int sum(int a, int b) {
   return (a + b);
}
```

Exemplo: Programa em Assembly

```c
int f, g, y;  // global variables

int main(void) {
  f = 2;
  g = 3;
  y = sum(f, g);

  return y;
}

int sum(int a, int b) {
  return (a + b);
}
```

```
.data
f:  .word   0x00000000
g:  .word   0x00000000
y:  .word   0x00000000
.text
main:
  addi $sp, $sp, -4   # stack frame
  sw   $ra, 0($sp)    # store $ra
  addi $a0, $0, 2     # $a0 = 2
  sw   $a0, f         # f = 2
  addi $a1, $0, 3     # $a1 = 3
  sw   $a1, g         # g = 3
  jal  sum            # call sum
  sw   $v0, y         # y = sum()
  lw   $ra, 0($sp)    # restore $ra
  addi $sp, $sp, 4    # restore $sp
  jr   $ra            # return to OS
sum:
  add  $v0, $a0, $a1  # $v0 = a + b
  jr   $ra            # return
```

Mapa de Memória MIPS



Exemplo: Tabela de Símbolos

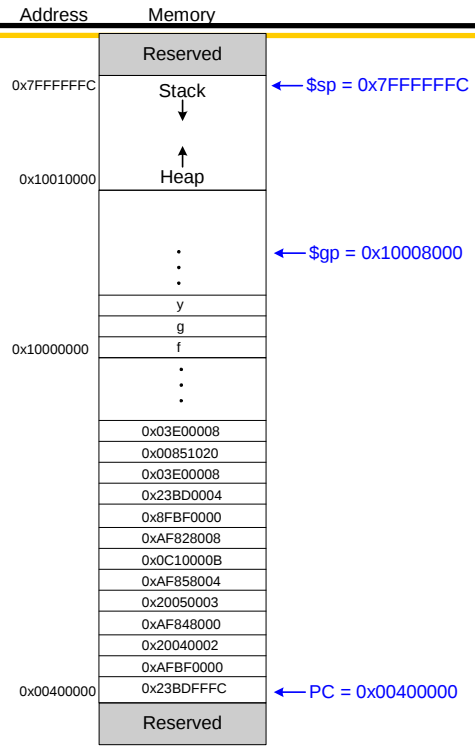
Symbol	Address
f	0x10000000
g	0x10000004
y	0x10000008
main	0x00400000
sum	0x0040002C

Exemplo: Programa Executável

Executable file header	Text Size	Data Size
	0x34 (52 bytes)	0xC (12 bytes)
Text segment	**Address**	**Instruction**
	0x00400000	0x23BDFFFC
	0x00400004	0xAFBF0000
	0x00400008	0x20040002
	0x0040000C	0xAF848000
	0x00400010	0x20050003
	0x00400014	0xAF858004
	0x00400018	0x0C10000B
	0x0040001C	0xAF828008
	0x00400020	0x8FBF0000
	0x00400024	0x23BD0004
	0x00400028	0x03E00008
	0x0040002C	0x00851020
	0x00400030	0x03E0008
Data segment	**Address**	**Data**
	0x10000000	f
	0x10000004	g
	0x10000008	y

```
addi $sp, $sp, -4
sw   $ra, 0 ($sp)
addi $a0, $0, 2
sw   $a0, 0x8000 ($gp)
addi $a1, $0, 3
sw   $a1, 0x8004 ($gp)
jal  0x0040002C
sw   $v0, 0x8008 ($gp)
lw   $ra, 0 ($sp)
addi $sp, $sp, -4
jr   $ra
add  $v0, $a0, $a1
jr   $ra
```

Exemplo: Programa na Memória



Pseudo Instruções

Pseudoinstruction	MIPS Instructions
`li $s0, 0x1234AA77`	`lui $s0, 0x1234` `ori $s0, 0xAA77`
`mul $s0, $s1, $s2`	`mult $s1, $s2` `mflo $s0`
`clear $t0`	`add $t0, $0, $0`
`move $s1, $s2`	`add $s2, $s1, $0`
`nop`	`sll $0, $0, 0`

Pseudo Instruções

Nome	Sintaxe	Tradução Inst. Reais	Significado
Move	move $rt,$rs	add $rt,$rs,$zero	R[rt]=R[rs]
Clear	clear $rt	add $rt,$zero,$zero	R[rt]=0
Not	not $rt, $rs	nor $rt, $rs, $zero	R[rt]=~R[rs]
Load Address	la $rd, LabelAddr	lui $rd, LabelAddr[31:16]; ori $rd,$rd, LabelAddr[15:0]	$rd = Label Address
Load Immediate	li $rd, IMMED[31:0]	lui $rd, IMMED[31:16]; ori $rd,$rd, IMMED[15:0]	$rd = 32 bit Immediate value
Branch unconditionally	b Label	beq $zero,$zero,Label	PC=Label
Branch and link	bal Label	bgezal $zero,Label	R[31]=PC+8; PC=Label
Branch if greater than	bgt $rs,$rt,Label	slt $at,$rt,$rs; bne $at, $zero,Label	if(R[rs]>R[rt]) PC=Label
Branch if less than	blt $rs,$rt,Label	slt $at,$rs,$rt; bne $at,$zero,Label	if(R[rs]<R[rt]) PC=Label
Branch if greater than or equal	bge $rs,$rt,Label	slt $at,$rs,$rt; beq $at,$zero,Label	if(R[rs]>=R[rt]) PC=Label
Branch if less than or equal	ble $rs,$rt,Label	slt $at,$rt,$rs; beq $at,$zero,Label	if(R[rs]<=R[rt]) PC=Label
Branch if greater than unsigned	bgtu $rs,$rt,Label		if(R[rs]>R[rt]) PC=Label
Branch if greater than zero	bgtz $rs,Label		if(R[rs]>0) PC=Label
Branch if equal to zero	beqz $rs,Label		if(R[rs]==0) PC=Label
Multiplies and returns only first 32 bits	mul $d, $s, $t	mult $s, $t; mflo $d	$d = $s * $t
Divides and returns quotient	div $d, $s, $t	div $s, $t; mflo $d	$d = $s / $t
Divides and returns remainder	rem $d, $s, $t	div $s, $t; mfhi $d	$d = $s % $t

Instruções

- **Soma e subtração**
  - **Signed: add, addi, sub**
    - » Executa a mesma operação que a versão unsigned
    - » Porém o processador gera exceção se overflow
  - **Unsigned: addu, addiu, subu**
    - » O processador não gera exceção se overflow
    - » Nota: `addiu` sign-extends o imediato
- **Multiplicação e Divisão**
  - » Signed: `mult`, `div`
  - » Unsigned: `multu`, `divu`
- **Set Less Than**
  - » Signed: `slt`, `slti`
  - » Unsigned: `sltu`, `sltiu`

Nota: `sltiu` sign-extends o imediato antes da comparação

Instruções

- **Loads**
  - **Signed:**
    - » Sign-extends para criar o valor de 32-bit
    - » Load halfword: `lh`
    - » Load byte: `lb`
  - **Unsigned:** **`addu`, `addiu`, `subu`**
    - » Zero-extends para criar o valor de 32-bit
    - » Load halfword unsigned: `lhu`
    - » Load byte: `lbu`

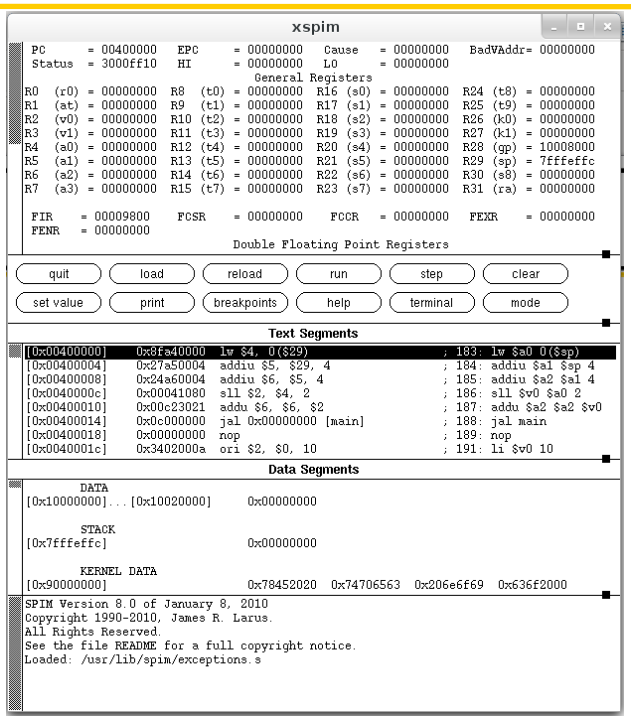
Ponto-Flutuante

- **Floating-point coprocessor (Coprocessor 1)**
- **32 registradores de 32-bit (`$f0` - `$f31`)**
- **Valores precisão dupla são mantidos em dois *floating point registers***
  - **e.g., `$f0` e `$f1`, `$f2` e `$f3`, etc.**
  - **Assim, os registradores de dupla precisão floating point são: `$f0`, `$f2`, `$f4`, etc.**

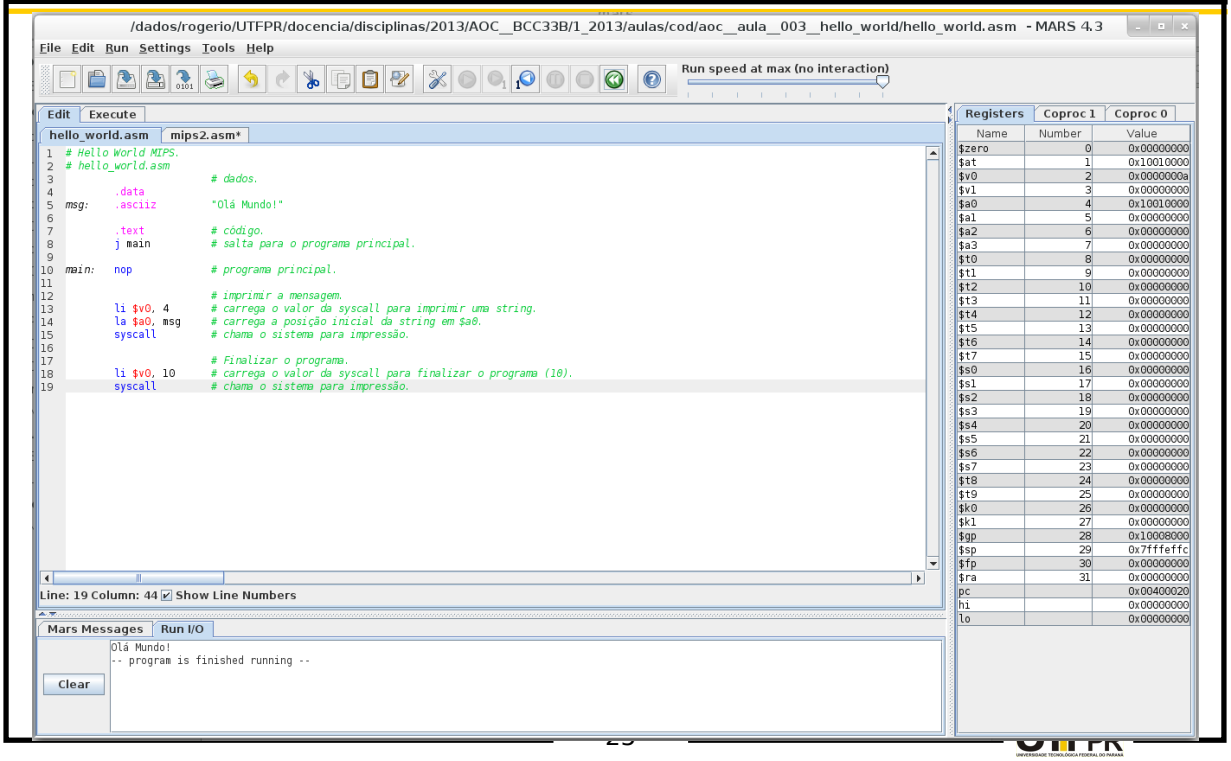
Ponto-Flutuante

Name	Register Number	Usage
$fv0 - $fv1	0, 2	return values
$ft0 - $ft3	4, 6, 8, 10	temporary variables
$fa0 - $fa1	12, 14	procedure arguments
$ft4 - $ft8	16, 18	temporary variables
$fs0 - $fs5	20, 22, 24, 26, 28, 30	saved variables

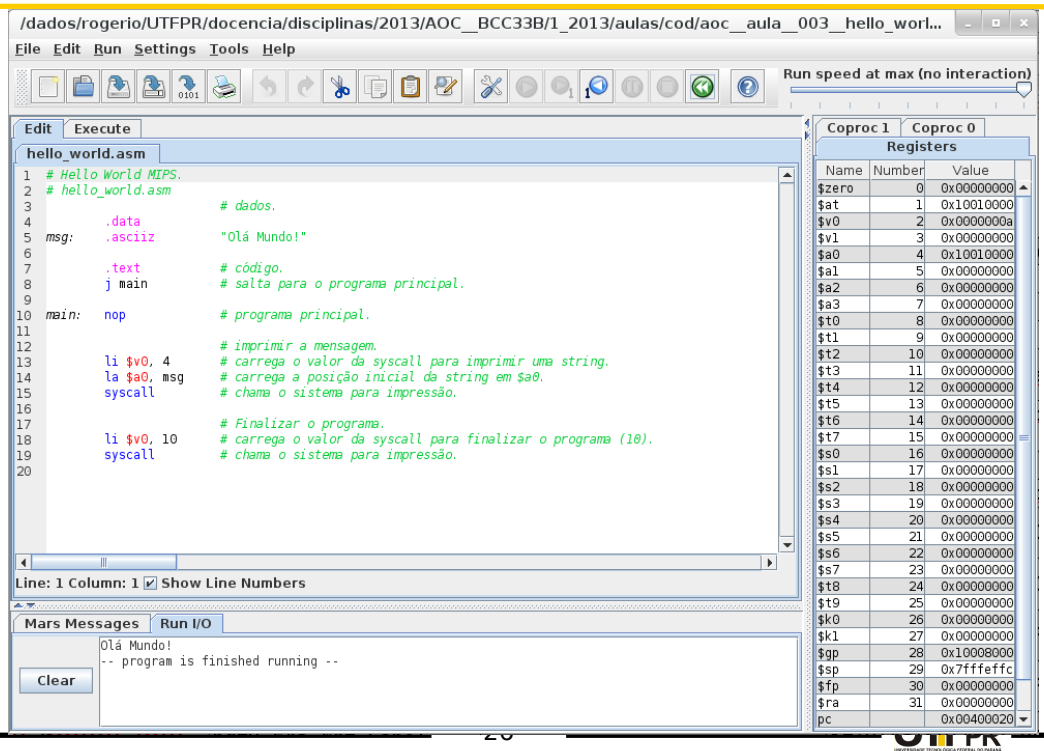
Simulador SPIM



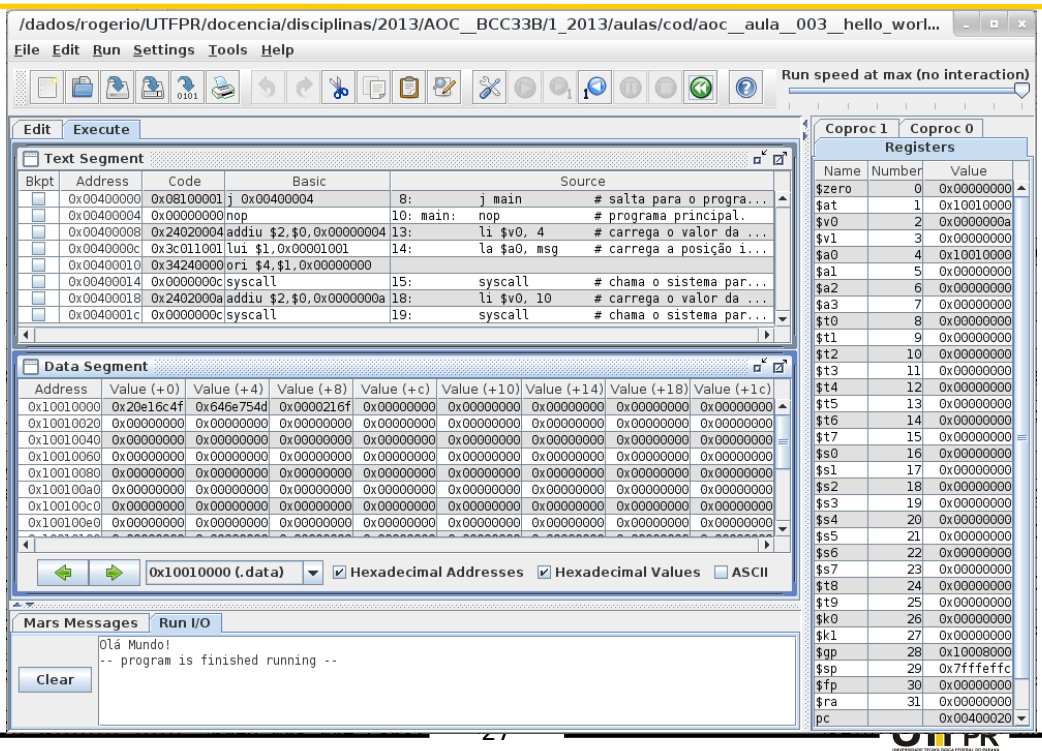
Simulador MARS



Hello World no MARS



Hello World no MARS



Diversão para Casa

- **Download do Simulador MARS**
- **Testar as instruções da aula**
- **Verificar alterações dos conteúdos dos registradores e memória**

Exercícios

- **Implemente os programas em A*ssembly* do MIPS para resolver as expressões:**

  a) a = b + c

  b) f = (g + h) – (i + j)

Exercícios

Supondo que h seja associado com o registrador *$s2* e o endereço base do *Array A* armazenado em *$s3*. Qual o código MIPS para o comando A[12] = h + A[8]?

Exercícios

Suponha que h seja associado com o registrador $s2 e o endereço base do array A armazenado em $s3. Qual o código MIPS para o comando A[12] = h + A[8];?

Solução

```
lw    $t0,32($s3)  #  $t0 recebe A[8]
add   $t0,$s2,$t0  #  $t0 recebe h + A[8]
sw    $t0,48($s3)  # armazena o resultado em A[12]
```

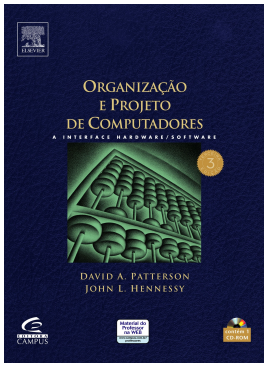

Exercícios

- **Qual o código MIPS para carregar uma constante de 32 bits no registrador $s0 ?**
  0000 0000 0011 1101 0000 1001 0000 0000

Solução

```
lui   $s0,61          # 61_10 = 0000 0000 0011 1101_2
ori   $s0,$s0,2304    # 2304_10 = 0000 1001 0000 0000_2

0000 0000 0011 1101 0000 1001 0000 0000_2
```

Leitura Recomendada



Capítulo 2

PATTERSON, David A.; HENNESSY, John L. ***Organização e projeto de computadores: a interface hardware/software.*** Rio de Janeiro, RJ: Elsevier, 2005. 484 p. ISBN 9788535215212.

Resumo da Aula de Hoje

<u>Tópicos mais importantes:</u>

Linguagem Assembly

Microprocessador MIPS

Registradores

Formatos de Instruções

Tipos de Instruções

Modos de Endereçamento