



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:

$$0xF234012F \text{ AND } 0x000000FF = 0x0000002F$$

- **or : útil para combinar bits**

- » Combinar 0xF2340000 com 0x000012BC:

$$0xF2340000 \text{ OR } 0x000012BC = 0xF23412BC$$

- **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

```
and $s3, $s1, $s2  
or  $s4, $s1, $s2  
xor $s5, $s1, $s2  
nor $s6, $s1, $s2
```

Result

\$s3	0100	0110	1010	0001	0000	0000	0000	0000
\$s4	1111	1111	1111	1111	1111	0000	1011	0111
\$s5	1011	1001	0101	1110	1111	0000	1011	0111
\$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

```
andi $s2, $s1, 0xFA34  
ori  $s3, $s1, 0xFA34  
xori $s4, $s1, 0xFA34
```

Result

\$s2	0000	0000	0000	0000	0000	0000	0011	0100
\$s3	0000	0000	0000	0000	1111	1010	1111	1111
\$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

	op	rs	rt	rd	shamt	funct
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
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Machine Code

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

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*)

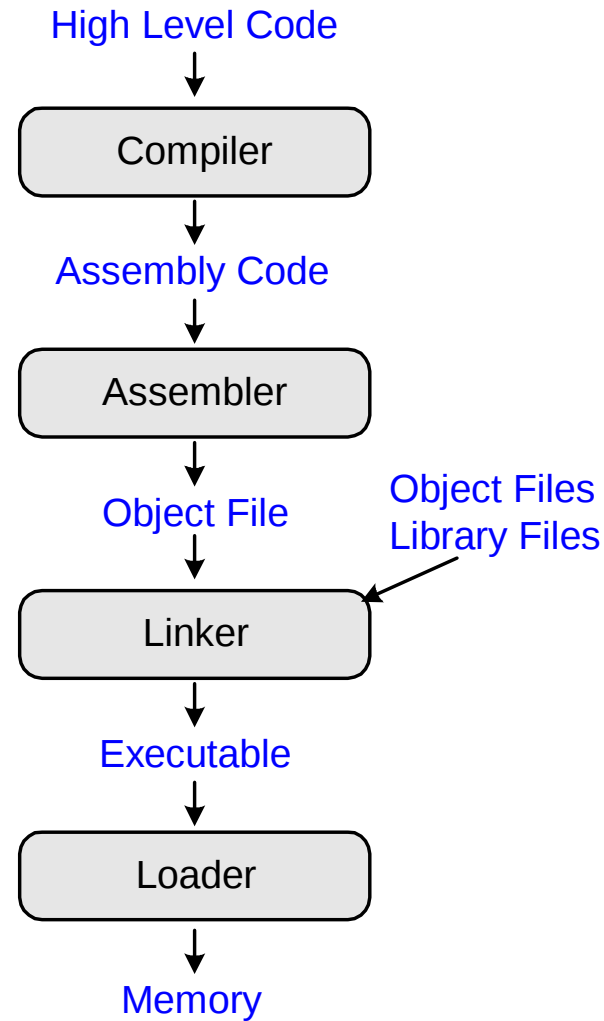
- **Dado**

- **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

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

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

    return y;
}

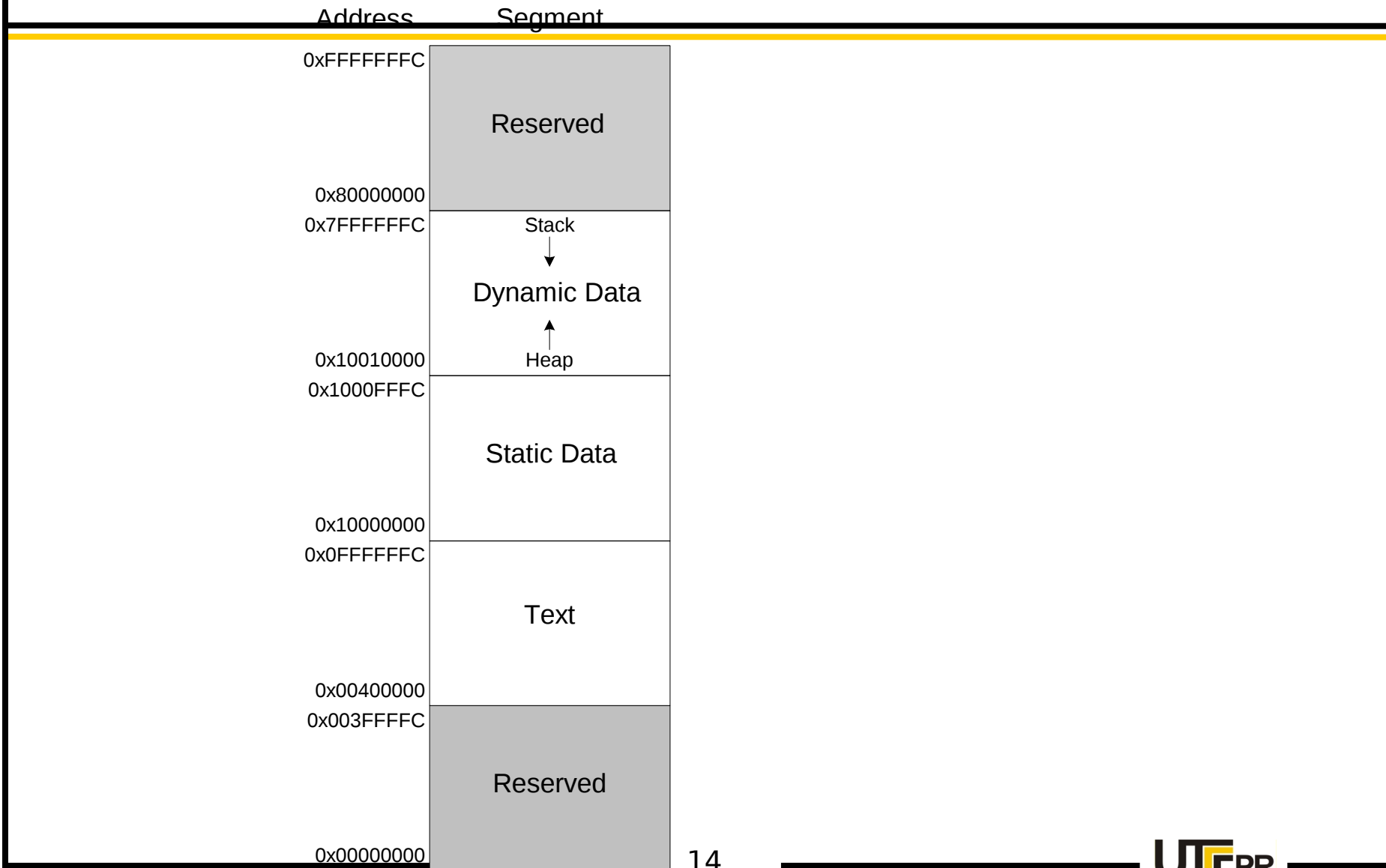
int sum(int a, int b) {
    return (a + b);
}
```

```
.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

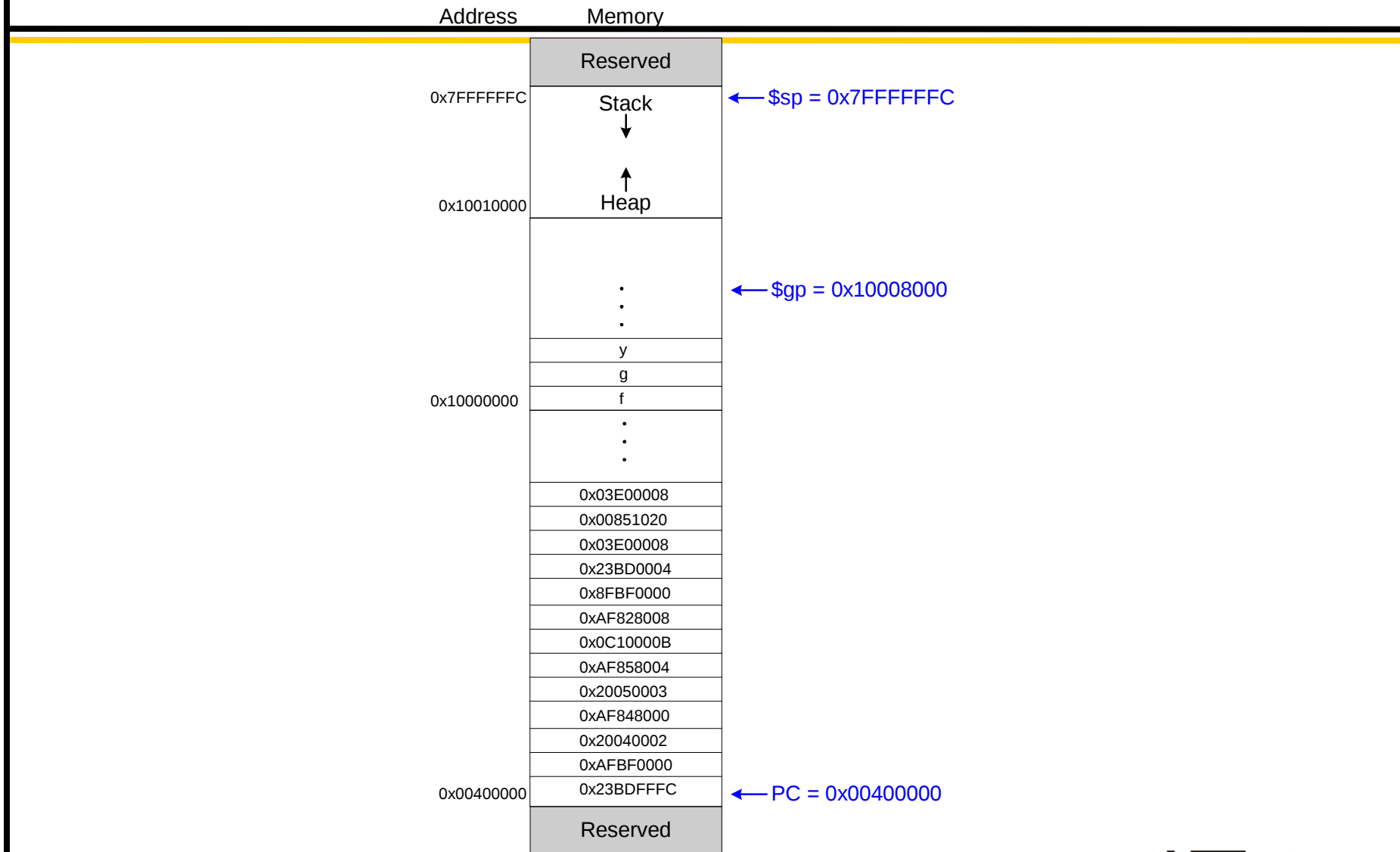
Exemplo: Programa Executável

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
<code>li \$s0, 0x1234AA77</code>	<code>lui \$s0, 0x1234</code> <code>ori \$s0, 0xAA77</code>
<code>mul \$s0, \$s1, \$s2</code>	<code>mult \$s1, \$s2</code> <code>mflo \$s0</code>
<code>clear \$t0</code>	<code>add \$t0, \$0, \$0</code>
<code>move \$s1, \$s2</code>	<code>add \$s2, \$s1, \$0</code>
<code>nop</code>	<code>sll \$0, \$0, 0</code>

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

```

xspim
PC      = 00400000  EPC    = 00000000  Cause  = 00000000  BadVAddr= 00000000
Status  = 3000fff0  HI     = 00000000  LO      = 00000000

General Registers
R0 (r0) = 00000000  R8 (t0) = 00000000  R16 (s0) = 00000000  R24 (t8) = 00000000
R1 (a0) = 00000000  R9 (t1) = 00000000  R17 (s1) = 00000000  R25 (t9) = 00000000
R2 (v0) = 00000000  R10 (t2) = 00000000  R18 (s2) = 00000000  R26 (k0) = 00000000
R3 (v1) = 00000000  R11 (t3) = 00000000  R19 (s3) = 00000000  R27 (k1) = 00000000
R4 (a0) = 00000000  R12 (t4) = 00000000  R20 (s4) = 00000000  R28 (gp) = 10008000
R5 (a1) = 00000000  R13 (t5) = 00000000  R21 (s5) = 00000000  R29 (sp) = 7ffffeffc
R6 (a2) = 00000000  R14 (t6) = 00000000  R22 (s6) = 00000000  R30 (s8) = 00000000
R7 (a3) = 00000000  R15 (t7) = 00000000  R23 (s7) = 00000000  R31 (ra) = 00000000

FIR      = 00009800  FCSR    = 00000000  FCCR     = 00000000  FEXR     = 00000000
FENR     = 00000000

Double Floating Point Registers

quit      load      reload      run      step      clear
set value  print     breakpoints  help     terminal  mode

Text Segments
[0x00400000] 0x8fa40000 lw $4, 0($29) ; 183: lw $a0 0($sp)
[0x00400004] 0x27a50004 addiu $5, $29, 4 ; 184: addiu $a1 $sp 4
[0x00400008] 0x24a60004 addiu $6, $5, 4 ; 185: addiu $a2 $a1 4
[0x0040000c] 0x00041080 sll $2, $4, 2 ; 186: sll $v0 $a0 2
[0x00400010] 0x00c23021 addu $6, $6, $2 ; 187: addu $a2 $a2 $v0
[0x00400014] 0x0c000000 jal 0x00000000 [main] ; 188: jal main
[0x00400018] 0x00000000 nop ; 189: nop
[0x0040001c] 0x3402000a ori $2, $0, 10 ; 191: li $v0 10

Data Segments
DATA
[0x10000000]...[0x10020000] 0x00000000

STACK
[0x7ffffeffc] 0x00000000

KERNEL DATA
[0x90000000] 0x78452020 0x74706563 0x206e6f69 0x636f2000

SPIM Version 8.0 of January 8, 2010
Copyright 1990-2010, James R. Larus.
All Rights Reserved.
See the file README for a full copyright notice.
Loaded: /usr/lib/spim/exceptions.s
  
```


Simulador MARS

/dados/rogerio/UTFPR/docencia/disciplinas/2013/AOC_BCC33B/1_2013/aulas/cod/aoc_aula_003_hello_world/hello_world.asm - MARS 4.3

File Edit Run Settings Tools Help

Run speed at max (no interaction)

hello_world.asm mips2.asm*

```
1 # Hello World MIPS.
2 # hello_world.asm
3
4      .data          # dados.
5 msg:  .asciiz       "Olá Mundo!"
6
7      .text          # código.
8      j main         # salta para o programa principal.
9
10     main: nop       # programa principal.
11
12     # imprimir a mensagem.
13     li $v0, 4        # carrega o valor da syscall para imprimir uma string.
14     la $a0, msg      # carrega a posição inicial da string em $a0.
15     syscall          # chama o sistema para impressão.
16
17     # Finalizar o programa.
18     li $v0, 10       # carrega o valor da syscall para finalizar o programa (10).
19     syscall          # chama o sistema para impressão.
```

Line: 19 Column: 44 ☒ Show Line Numbers

Mars Messages Run I/O

Olá Mundo!
-- program is finished running --

Clear

Registers		Coproc 1	Coproc 0
Name	Number	Value	
\$zero	0	0x00000000	
\$at	1	0x10010000	
\$v0	2	0x0000000a	
\$v1	3	0x00000000	
\$a0	4	0x10010000	
\$a1	5	0x00000000	
\$a2	6	0x00000000	
\$a3	7	0x00000000	
\$t0	8	0x00000000	
\$t1	9	0x00000000	
\$t2	10	0x00000000	
\$t3	11	0x00000000	
\$t4	12	0x00000000	
\$t5	13	0x00000000	
\$t6	14	0x00000000	
\$t7	15	0x00000000	
\$s0	16	0x00000000	
\$s1	17	0x00000000	
\$s2	18	0x00000000	
\$s3	19	0x00000000	
\$s4	20	0x00000000	
\$s5	21	0x00000000	
\$s6	22	0x00000000	
\$s7	23	0x00000000	
\$t8	24	0x00000000	
\$t9	25	0x00000000	
\$k0	26	0x00000000	
\$k1	27	0x00000000	
\$gp	28	0x10008000	
\$sp	29	0x7ffffc00	
\$fp	30	0x00000000	
\$ra	31	0x00000000	
pc		0x00400020	
hi		0x00000000	
lo		0x00000000	

Hello World no MARS

/dados/rogerio/UTFPR/docencia/disciplinas/2013/AOC_BCC33B/1_2013/aulas/cod/aoc_aula_003_hello_worl...

File Edit Run Settings Tools Help

Run speed at max (no interaction)

0101

Edit Execute

hello_world.asm

```
1 # Hello World MIPS.
2 # hello_world.asm
3
4         .data           # dados.
5 msg:    .asciiz         "Olá Mundo!"
6
7         .text           # código.
8         j main          # salta para o programa principal.
9
10 main:   nop             # programa principal.
11
12         # imprimir a mensagem.
13         li $v0, 4        # carrega o valor da syscall para imprimir uma string.
14         la $a0, msg      # carrega a posição inicial da string em $a0.
15         syscall          # chama o sistema para impressão.
16
17         # Finalizar o programa.
18         li $v0, 10       # carrega o valor da syscall para finalizar o programa (10).
19         syscall          # chama o sistema para impressão.
20
```

Line: 1 Column: 1 ☒ Show Line Numbers

Mars Messages Run I/O

Olá Mundo!
-- program is finished running --

Clear

Coproc 1 Coproc 0

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x10010000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7ffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400020

Hello World no MARS

/dados/rogerio/UTFPR/docencia/disciplinas/2013/AOC_BCC33B/1_2013/aulas/cod/aoc_aula_003_hello_worl...

File Edit Run Settings Tools Help

Run speed at max (no interaction)

Edit Execute

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x08100001	j 0x00400004	8: j main # salta para o progra...
	0x00400004	0x00000000	nop	10: main: nop # programa principal.
	0x00400008	0x24020004	addiu \$2,\$0,0x00000004	13: li \$v0, 4 # carrega o valor da ...
	0x0040000c	0x3c011001	lui \$1,0x00001001	14: la \$a0, msg # carrega a posição i...
	0x00400010	0x34240000	ori \$4,\$1,0x00000000	
	0x00400014	0x0000000c	syscall	15: syscall # chama o sistema par...
	0x00400018	0x2402000a	addiu \$2,\$0,0x0000000a	18: li \$v0, 10 # carrega o valor da ...
	0x0040001c	0x0000000c	syscall	19: syscall # chama o sistema par...

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x20e16c4f	0x646e754d	0x0000216f	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010040	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010060	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x10010080	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100a0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100c0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000
0x100100e0	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

0x10010000 (.data) [X] Hexadecimal Addresses [X] Hexadecimal Values [] ASCII

Mars Messages Run I/O

Olá Mundo!
-- program is finished running --

Clear

Coproc 1 Coproc 0

Registers

Name	Number	Value
\$zero	0	0x00000000
\$at	1	0x10010000
\$v0	2	0x0000000a
\$v1	3	0x00000000
\$a0	4	0x10010000
\$a1	5	0x00000000
\$a2	6	0x00000000
\$a3	7	0x00000000
\$t0	8	0x00000000
\$t1	9	0x00000000
\$t2	10	0x00000000
\$t3	11	0x00000000
\$t4	12	0x00000000
\$t5	13	0x00000000
\$t6	14	0x00000000
\$t7	15	0x00000000
\$s0	16	0x00000000
\$s1	17	0x00000000
\$s2	18	0x00000000
\$s3	19	0x00000000
\$s4	20	0x00000000
\$s5	21	0x00000000
\$s6	22	0x00000000
\$s7	23	0x00000000
\$t8	24	0x00000000
\$t9	25	0x00000000
\$k0	26	0x00000000
\$k1	27	0x00000000
\$gp	28	0x10008000
\$sp	29	0x7fffffc
\$fp	30	0x00000000
\$ra	31	0x00000000
pc		0x00400020

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 *Assembly* 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 ?

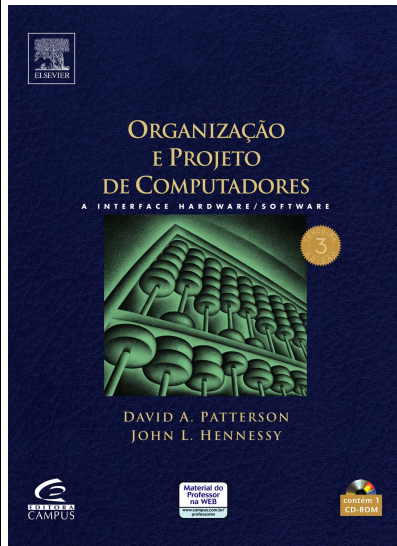
0000 0000 0011 1101 0000 1001 0000 0000

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 0000₂

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

Tópicos mais importantes:

Linguagem Assembly

Microprocessador MIPS

Registradores

Formatos de Instruções

Tipos de Instruções

Modos de Endereçamento