

MATA48 – ARQUITETURA DE COMPUTADORES
PROF. MARCOS E. BARRETO

NOTAS DE AULA – ARQUITETURA MIPS (32 BITS)

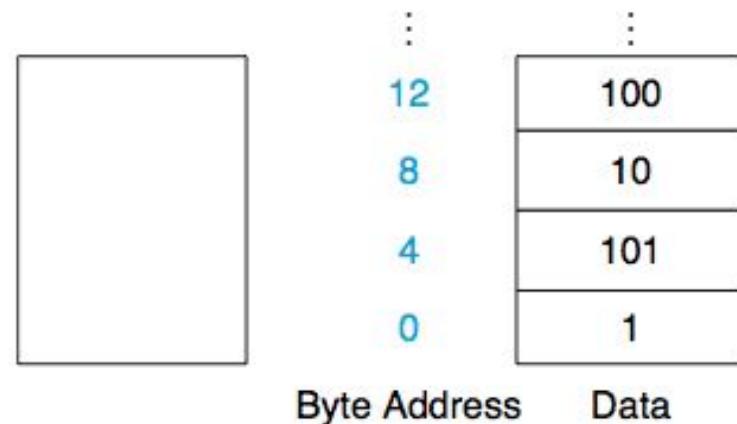
MIPS operands

Name	Example	Comments
32 registers	\$s0-\$s7, \$t0-\$t9, \$zero, \$a0-\$a3, \$v0-\$v1, \$gp, \$fp, \$sp, \$ra, \$at	Fast locations for data. In MIPS, data must be in registers to perform arithmetic, register \$zero always equals 0, and register \$at is reserved by the assembler to handle large constants.
2^{30} memory words	Memory[0], Memory[4], ..., Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, arrays, and spilled registers.

MIPS assembly language

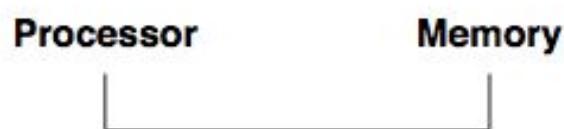
Category	Instruction	Example	Meaning	Comments
Arithmetic	add	add \$s1,\$s2,\$s3	$\$s1 = \$s2 + \$s3$	Three register operands
	subtract	sub \$s1,\$s2,\$s3	$\$s1 = \$s2 - \$s3$	Three register operands
	add immediate	addi \$s1,\$s2,20	$\$s1 = \$s2 + 20$	Used to add constants
Data transfer	load word	lw \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Word from memory to register
	store word	sw \$s1,20(\$s2)	$\text{Memory}[\$s2 + 20] = \$s1$	Word from register to memory
	load half	lh \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Halfword memory to register
	load half unsigned	lhu \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Halfword memory to register
	store half	sh \$s1,20(\$s2)	$\text{Memory}[\$s2 + 20] = \$s1$	Halfword register to memory
	load byte	lb \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Byte from memory to register
	load byte unsigned	lbu \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Byte from memory to register
	store byte	sb \$s1,20(\$s2)	$\text{Memory}[\$s2 + 20] = \$s1$	Byte from register to memory
	load linked word	ll \$s1,20(\$s2)	$\$s1 = \text{Memory}[\$s2 + 20]$	Load word as 1st half of atomic swap
	store condition. word	sc \$s1,20(\$s2)	$\text{Memory}[\$s2 + 20] = \$s1; \$s1 = 0 \text{ or } 1$	Store word as 2nd half of atomic swap
Logical	load upper immed.	lui \$s1,20	$\$s1 = 20 * 2^{16}$	Loads constant in upper 16 bits
	and	and \$s1,\$s2,\$s3	$\$s1 = \$s2 \& \$s3$	Three reg. operands; bit-by-bit AND
	or	or \$s1,\$s2,\$s3	$\$s1 = \$s2 \$s3$	Three reg. operands; bit-by-bit OR
	nor	nor \$s1,\$s2,\$s3	$\$s1 = \sim (\$s2 \$s3)$	Three reg. operands; bit-by-bit NOR
	and immediate	andi \$s1,\$s2,20	$\$s1 = \$s2 \& 20$	Bit-by-bit AND reg with constant
	or immediate	ori \$s1,\$s2,20	$\$s1 = \$s2 20$	Bit-by-bit OR reg with constant
	shift left logical	sll \$s1,\$s2,10	$\$s1 = \$s2 << 10$	Shift left by constant
	shift right logical	srl \$s1,\$s2,10	$\$s1 = \$s2 >> 10$	Shift right by constant
Conditional branch	branch on equal	beq \$s1,\$s2,25	if ($\$s1 == \$s2$) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	bne \$s1,\$s2,25	if ($\$s1 != \$s2$) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	slt \$s1,\$s2,\$s3	if ($\$s2 < \$s3$) $\$s1 = 1$; else $\$s1 = 0$	Compare less than; for beq, bne
	set on less than unsigned	sltu \$s1,\$s2,\$s3	if ($\$s2 < \$s3$) $\$s1 = 1$; else $\$s1 = 0$	Compare less than unsigned
	set less than immediate	slti \$s1,\$s2,20	if ($\$s2 < 20$) $\$s1 = 1$; else $\$s1 = 0$	Compare less than constant
	set less than immediate unsigned	sltiu \$s1,\$s2,20	if ($\$s2 < 20$) $\$s1 = 1$; else $\$s1 = 0$	Compare less than constant unsigned
Unconditional jump	jump	j 2500	go to 10000	Jump to target address
	jump register	jr \$ra	go to \$ra	For switch, procedure return
	jump and link	jal 2500	$\$ra = \text{PC} + 4$; go to 10000	For procedure call

FIGURE 2.1 MIPS assembly language revealed in this chapter. This information is also found in Column 1 of the MIPS Reference Data Card at the front of this book.



Lembrar que os endereços de memória devem ser múltiplos de 4 (bytes).

Ex: $A[12] = h + A[8]$



Considerando o mapeamento registrador \leq memória

la \$s3, A # \$s3 tem endereço de A[0]
 la \$s2, h # \$s2 tem endereço de h

$$A[12] = h + A[8];$$

```
lw    $t0,32($s3)  # Temporary reg $t0 gets A[8]
add  $t0,$s2,$t0  # Temporary reg $t0 gets h + A[8]
sw    $t0,48($s3)  # Stores h + A[8] back into A[12]
```

add \$t0,\$s1,\$s2

Exemplo de instrução no formato R

0	17	18	8	0	32
000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

MIPS fields are given names to make them easier to discuss:

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

Here is the meaning of each name of the fields in MIPS instructions:

- *op*: Basic operation of the instruction, traditionally called the **opcode**.
- *rs*: The first register source operand.
- *rt*: The second register source operand.
- *rd*: The register destination operand. It gets the result of the operation.
- *shamt*: Shift amount. (Section 2.6 explains shift instructions and this term; it will not be used until then, and hence the field contains zero in this section.)
- *funct*: Function. This field, often called the *function code*, selects the specific variant of the operation in the op field.

Exemplo de instrução no formato I

lw \$t0,32(\$s3) # Temporary reg \$t0 gets A[8]

op	rs	rt	constant or address
6 bits	5 bits	5 bits	16 bits

```
beq register1, register2, L1  
bne register1, register2, L1
```

Exemplo de instrução no formato I

```
bne $s0,$s1,Exit # go to Exit if $s0 ≠ $s1
```

is assembled into this instruction, leaving only 16 bits for the branch address:

5	16	17	Exit
6 bits	5 bits	5 bits	16 bits

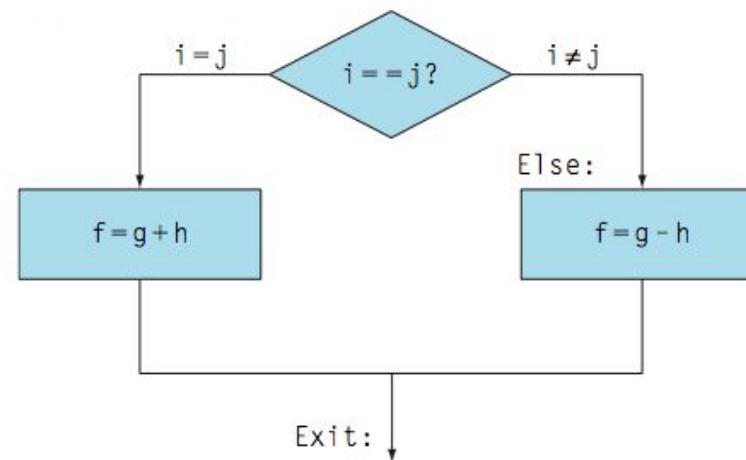


FIGURE 2.9 Illustration of the options in the *if* statement above. The left box corresponds to the *then* part of the *if* statement, and the right box corresponds to the *else* part.

`A[300] = h + A[300];`

is compiled into

```
lw    $t0,1200($t1) # Temporary reg $t0 gets A[300]
add   $t0,$s2,$t0   # Temporary reg $t0 gets h + A[300]
sw    $t0,1200($t1) # Stores h + A[300] back into A[300]
```

What is the MIPS machine language code for these three instructions?



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



100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

MIPS machine language

Name	Format	Example						Comments
add	R	0	18	19	17	0	32	add \$s1,\$s2,\$s3
sub	R	0	18	19	17	0	34	sub \$s1,\$s2,\$s3
addi	I	8	18	17	100			addi \$s1,\$s2,100
lw	I	35	18	17	100			lw \$s1,100(\$s2)
sw	I	43	18	17	100			sw \$s1,100(\$s2)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer format

FIGURE 2.6 MIPS architecture revealed through Section 2.5. The two MIPS instruction formats so far are R and I. The first 16 bits are the same: both contain an *op* field, giving the base operation; an *rs* field, giving one of the sources; and the *rt* field, which specifies the other source operand, except for load word, where it specifies the destination register. R-format divides the last 16 bits into an *rd* field, specifying the destination register; the *shamt* field, which Section 2.6 explains; and the *funct* field, which specifies the specific operation of R-format instructions. I-format combines the last 16 bits into a single *address* field.

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

FIGURE 2.5 MIPS instruction encoding. In the table above, “reg” means a register number between 0 and 31, “address” means a 16-bit address, and “n.a.” (not applicable) means this field does not appear in this format. Note that add and sub instructions have the same value in the op field; the hardware uses the funct field to decide the variant of the operation: add (32) or subtract (34).

```

Loop: sll    $t1,$s3,2  # Temp reg $t1 = 4 * i
      add $t1,$t1,$s6  # $t1 = address of save[i]
      lw   $t0,0($t1)   # Temp reg $t0 = save[i]
      bne $t0,$s5, Exit # go to Exit if save[i] ≠ k
      addi $s3,$s3,1    # i = i + 1
      j    Loop          # go to Loop
Exit:

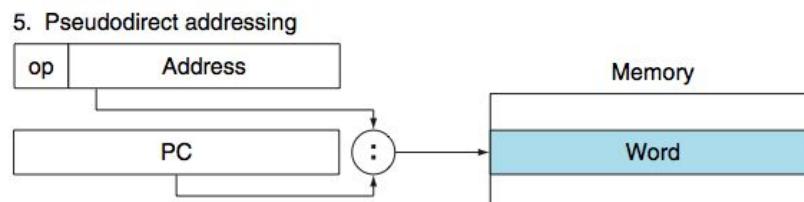
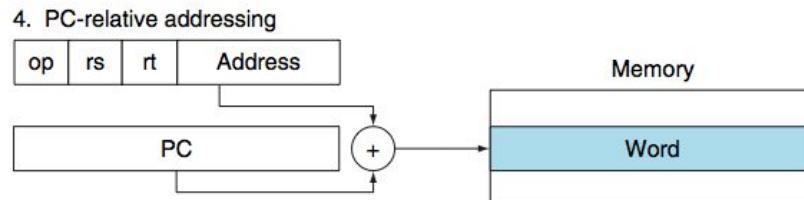
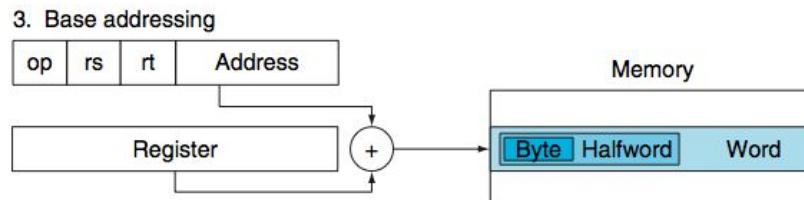
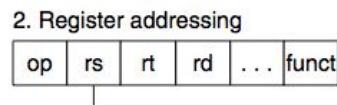
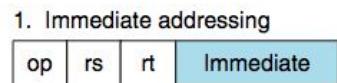
```

Exemplo de instrução no formato J

If we assume we place the loop starting at location 80000 in memory, what is the MIPS machine code for this loop?

The assembled instructions and their addresses are:

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



MODOS DE ENDEREÇAMENTO DE MEMÓRIA

1. *Immediate addressing*, where the operand is a constant within the instruction itself
2. *Register addressing*, where the operand is a register
3. *Base or displacement addressing*, where the operand is at the memory location whose address is the sum of a register and a constant in the instruction
4. *PC-relative addressing*, where the branch address is the sum of the PC and a constant in the instruction
5. *Pseudodirect addressing*, where the jump address is the 26 bits of the instruction concatenated with the upper bits of the PC

FIGURE 2.18 Illustration of the five MIPS addressing modes. The operands are shaded in color. The operand of mode 3 is in memory, whereas the operand for mode 2 is a register. Note that versions of load and store access bytes, halfwords, or words. For mode 1, the operand is 16 bits of the instruction itself. Modes 4 and 5 address instructions in memory, with mode 4 adding a 16-bit address shifted left 2 bits to the PC and mode 5 concatenating a 26-bit address shifted left 2 bits with the 4 upper bits of the PC.