

## Linguagem de máquina do MIPS

- O formato dos campos das instruções estão no documento: MIPS Reference Data Card.pdf

- O código assembly para conferência no MARS: exemplo5\_pag85\_c\_tabela.asm

# Assembly para a instrução while (save[i] == k) i += 1;

```
.data                                # 0x10010000
save: .word 0, 0, 1, 1              # vetor save[]={0, 0, 1, 1};

.text                                # 0x00400000
.globl main                          # declara que main é um símbolo global

main: add $s3, $zero, $zero          # registrador $s3 recebe 0 + 0 (i = 0)
      add $s5, $zero, $zero          # registrador $s5 recebe 0 + 0 (k = 0)
Loop: sll $t1, $s3, 2                # registrador temporário $t1 = 4 * i
      la  $s6, save                  # carrega o endereço do rotulo save (endereço base array) para o registrador $s6
      add $t1, $t1, $s6              # $t1 = endereço de save[i]
      lw  $t0, 0($t1)                # registrador temporário $t0 = save[i]
      bne $t0, $s5, Exit             # vá para Exite se save[i] <> k
      addi $s3, $s3, 1               # i = i + 1
      j Loop                          # vá para Loop
Exit: nop
```

R Op (6 bits)	Rs (5 bits)	Rt (5 bits)	Rd (5 bits)	Shamt (5 bits)	Funct (6 bits)
I Op	Rs	Rt	Endereço (16 bits)		
J Op	Endereço (26 bits)				
0x00400000 main: add \$s3,\$zero,\$zero (add \$19,\$0,\$0)			R[rd] = R[rs] + R[rt] (0/20 hex)		
0x00	0	0	19	0	(0x20)
0 0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 0 0 1 1	0 0 0 0 0	1 0 0 0 0 0
0	0	0	9	8	2 0
0x00400004 add \$s5, \$zero, \$zero (add \$21,\$0,\$0)			R[rt] = R[rs] + SignExtImm (8 hex)		
0x00	0	0	21	0	(0x20)
0 0 0 0 0 0	0 0 0 0 0	0 0 0 0 0	1 0 1 0 1	0 0 0 0 0	1 0 0 0 0 0
0	0	0	A	8	2 0
0x00400008 loop: sll \$t1, \$s3, 2 (sll \$9,\$19,0x00000002)			R[rd] = R[rt] << shamt (0/00 hex)		
0x0	0	19	9	2	(0x00)
0 0 0 0 0 0	0 0 0 0 0	1 0 0 1 1	0 1 0 0 1	0 0 0 1 0	0 0 0 0 0 0
0	0	1	4	8	8 0
0x0040000c la \$s6,save (lui \$1,0x00001001)			R[rt] = {imm, 16'b0} (F hex)		
0x0F	0	1			0x1001
0 0 1 1 1 1	0 0 0 0 0	0 0 0 0 1	0 0 0 1 0	0 0 0 0 0	0 0 0 0 0 1
3	C	0	1	0	0 1
0x00400010 ori \$22,\$1,0x00000000			R[rt] = R[rs]   ZeroExtImm (D hex)		
0x0D	1	22			0x0000
0 0 1 1 0 1	0 0 0 0 1	1 0 1 1 0	0 0 0 0 0	0 0 0 0 0	0 0 0 0 0 0
3	4	3	6	0	0 0

0x00400014 add \$t1,\$t1,\$s6 (add \$9,\$9,\$22)																R[rd] = R[rs] + R[rt] (0/20 hex)																															
0x00						9						22						9												(0x20)																	
0	0	0	0	0	0	0	1	0	0	1	1	0	1	1	0	0	1	0	0	1	0	0	0	0	0	1	0	0	0	0	0																
0						1						3						6						4						8						2						0					
0x00400018 lw \$t0, 0(\$t1) (lw \$8,0x00000000(\$9))																R[rt] = M[R[rs]+SignExtImm] (23 hex)																															
0x23						9						8																		0x0000																	
1	0	0	0	1	1	0	1	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																
8						D						2						8						0						0						0						0					
R[rt] = M[R[rs]+SignExtImm] R[8] = M[R[9]+0x0000.0000]																																															
0x0040001c bne \$t0, \$s5, Exit (bne \$8,\$21,0x00000002)																if(R[rs]!=R[rt]) PC=PC+4+BranchAddr (0x05)																															
0x05						8						21																		0x0002																	
0	0	0	1	0	1	0	1	0	0	0	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																
1						5						1						5						0						0						0						2					
0x0002 extendido sinal de 16 bits para 32 bits = 0x0000.0002 0x0000.0002 << 2 = 0x0000.0008 bytes  PC = (PC+4) +BranchAddr PC = 0x00400020 + 0x00000008 = 0x00400028 (Exit: nop)																																															
0x00400020 addi \$s3, \$s3, 1 (addi \$19,\$19,0x00000001)																R[rt] = R[rs] + SignExtImm (0x08)																															
0x08						19						19																		0x0001																	
0	0	0	0	0	0	1	0	0	1	1	1	0	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1																
0						2						7						3						0						0						0						1					
0x00400024 j Loop (j 0x00400008) end. bytes 0x00400008 / 4 = 0x00100002 words																PC=JumpAddr (0x02)																															
0x02																														0x0100002																	
0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																
0						8						1						0						0						0						0						2					
0x0100002 << 2 = 0x0400008																																															
						0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															
						0						4						0						0						0						0						8					
PC+4= 0x00400028																																															
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0																
0						0						4						0						0						0						2						8					
PC = 4 Bits mais significativos (PC+4= 0x00400028) concatenados 28 bits 0x0400008 = 0x00400008																																															
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0																
0						0						4						0						0						0						0						8					
0x00400028																0x00000000																															

nop																							
0x0						0x0						0x0						0x0					
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0						0						0						0					

Text Segment

Bkpt	Address	Code	Basic	Source
	0x00400000	0x00009820	add \$19,\$0,\$0	9: main: add \$s3, \$zero, \$zero # registrador \$s3 recebe 0 + 0 (i = 0)
	0x00400004	0x0000a820	add \$21,\$0,\$0	10: add \$s5, \$zero, \$zero # registrador \$s5 recebe 0 + 0 (k = 0)
	0x00400008	0x00134880	sll \$9,\$19,0x00000002	17: Loop: sll \$t1, \$s3, 2 # registrador temporário \$t1 = 4 * i
	0x0040000c	0x3c011001	lui \$1,0x00001001	18: la \$s6,save # carrega o endereço do rotulo save (endereço base array) para o registrador \$s6
	0x00400010	0x34360000	ori \$22,\$1,0x00000000	
	0x00400014	0x01364820	add \$9,\$9,\$22	20: add \$t1, \$t1, \$s6 # \$t1 = endereço de save[i]
	0x00400018	0x8d280000	lw \$8,0x00000000(\$9)	23: lw \$t0, 0(\$t1) # registrador temporário \$t0 = save[i]
	0x0040001c	0x15150002	bne \$8,\$21,0x00000002	26: bne \$t0, \$s5, Exit # vá para Exite se save[i] > k
	0x00400020	0x22730001	addi \$19,\$19,0x0000...	27: addi \$s3, \$s3, 1 # i = i + 1
	0x00400024	0x08100002	j 0x00400008	31: j Loop # vá para Loop
	0x00400028	0x00000000	nop	33: Exit: nop

Data Segment

Address	Value (+0)	Value (+4)	Value (+8)	Value (+c)	Value (+10)	Value (+14)	Value (+18)	Value (+1c)
0x10010000	0x00000000	0x00000000	0x00000001	0x00000001	0x00000000	0x00000000	0x00000000	0x00000000
0x10010020	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000	0x00000000

←

→

0x10010000 (.data)

☒ Hexadecimal Addresses☒ Hexadecimal Values☐ ASCII