ISA MIPS

Tabelas importantes

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

Name	Register number	Usage	Preserved on call?
$zero	0	The constant value 0	n.a.
$v0–$v1	2–3	Values for results and expression evaluation	no
$a0–$a3	4–7	Arguments	no
$t0–$t7	8–15	Temporaries	no
$s0–$s7	16–23	Saved	yes
$t8–$t9	24–25	More temporaries	no
$gp	28	Global pointer	yes
$sp	29	Stack pointer	yes
$fp	30	Frame pointer	yes
$ra	31	Return address	yes

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions are 32 bits long
R-format	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	op	rs	rt	address/immediate			Transfer, branch, imm. format
J-format	op	target address					Jump instruction format

Figuras extraídas do livro "Organização e Projeto de Computadores: A interface Hardware/Software" de David A. Patterson e John L. Hennessy

op(31:26)								
28–26 31–29	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	R-format	Bltz/gez	jump	jump & link	branch eq	branch ne	blez	bgtz
1(001)	add immediate	addiu	set less than imm.	set less than imm. unsigned	andi	ori	xori	load upper immediate
2(010)	TLB	FlPt						
3(011)								
4(100)	load byte	load half	lwl	load word	load byte unsigned	load half unsigned	lwr	
5(101)	store byte	store half	swl	store word			swr	
6(110)	load linked word	lwcl						
7(111)	store cond. word	swcl						

op(31:26)=010000 (TLB), rs(25:21)								
23–21 25–24	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(00)	mfc0		cfc0		mtc0		ctc0	
1(01)								
2(10)								
3(11)								

op(31:26)=000000 (R-format), funct(5:0)								
2–0 5–3	0(000)	1(001)	2(010)	3(011)	4(100)	5(101)	6(110)	7(111)
0(000)	shift left logical		shift right logical	sra	sllv		srlv	srav
1(001)	jump register	jalr			syscall	break		
2(010)	mfhi	mthi	mflo	mtlo				
3(011)	mult	multu	div	divu				
4(100)	add	addu	subtract	subu	and	or	xor	not or (nor)
5(101)			set l.t.	set l.t. unsigned				
6(110)								
7(111)								

Figuras extraídas do livro "Organização e Projeto de Computadores: A interface Hardware/Software" de David A. Patterson e John L. Hennessy

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)	$s1 = Memory[$s2 + 20]	Word from memory to register
	store word	sw $s1,20($s2)	Memory[$s2 + 20] = $s1	Word from register to memory
	load half	lh $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	load half unsigned	lhu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Halfword memory to register
	store half	sh $s1,20($s2)	Memory[$s2 + 20] = $s1	Halfword register to memory
	load byte	lb $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	load byte unsigned	lbu $s1,20($s2)	$s1 = Memory[$s2 + 20]	Byte from memory to register
	store byte	sb $s1,20($s2)	Memory[$s2 + 20] = $s1	Byte from register to memory
	load linked word	ll $s1,20($s2)	$s1 = Memory[$s2 + 20]	Load word as 1st half of atomic swap
	store condition. word	sc $s1,20($s2)	Memory[$s2+20]=$s1;$s1=0 or 1	Store word as 2nd half of atomic swap
	load upper immed.	lui $s1,20	$s1 = 20 * 2^{16}	Loads constant in upper 16 bits
Logical	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	$s1 = ~ ($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 = PC + 4; go to 10000	For procedure call

Figuras extraídas do livro "Organização e Projeto de Computadores: A interface Hardware/Software" de David A. Patterson e John L. Hennessy