Arquitetura de Computadores
Aulas Práticas 2017/2018

4. Representação Binária de Instruções MIPS R2000

Formato, operandos e registos das instruções assembly MIPS R2000:

Name	Fields						Comments
Field size	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
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

MIPS operands

Name	Example	Comments
32 registers	$s0–$s7, $t0–$t9, $zero, $a0–$a3, $v0–$v1, $gp, $fp, $sp, $ra	Fast locations for data. In MIPS, data must be in registers to perform arithmetic. MIPS register $zero always equals 0. $gp (28) is the global pointer, $sp (29) is the stack pointer, $fp (30) is the frame pointer, and $ra (31) is the return address.
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, such as those saved on procedure calls.

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

Subconjunto das instruções assembly MIPS R2000:

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
Data transfer	load word	`lw $s1,100($s2)`	`$s1 = Memory[$s2 + 100]`	Data from memory to register
	store word	`sw $s1,100($s2)`	`Memory[$s2 + 100] = $s1`	Data from register to memory
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 $s1,$s2,100`	`$s1 = $s2 & 100`	Bit-by-bit AND reg with constant
	or immediate	`ori $s1,$s2,100`	`$s1 = $s2 | 100`	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,L`	if (`$s1 == $s2`) go to `L`	Equal test and branch
	branch on not equal	`bne $s1,$s2,L`	if (`$s1 != $s2`) go to `L`	Not equal test and branch
	set on less than	`slt $s1,$s2,$s3`	if (`$s2 < $s3`) `$s1 = 1`; else `$s1 = 0`	Compare less than; used with `beq`, `bne`
	set on less than immediate	`slt $s1,$s2,100`	if (`$s2 < 100`) `$s1 = 1`; else `$s1 = 0`	Compare less than immediate; used with `beq`, `bne`
Unconditional jump	jump	`j L`	`go to L`	Jump to target address
	`jump register`	`jr $ra`	go to `$ra`	For procedure return
	jump and link	`jal L`	`$ra = PC + 4; go to L`	For procedure call

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`
`lw`	I	35	18	17	100			`lw $s1,100($s2)`
`sw`	I	43	18	17	100			`sw $s1,100($s2)`
`and`	R	0	18	19	17	0	36	`and $s1,$s2,$s3`
`or`	R	0	18	19	17	0	37	`or $s1,$s2,$s3`
`nor`	R	0	18	19	17	0	39	`nor $s1,$s2,$s3`
`andi`	I	12	18	17	100			`andi $s1,$s2,100`
`ori`	I	13	18	17	100			`ori $s1,$s2,100`
`sll`	R	0	0	18	17	10	0	`sll $s1,$s2,10`
`srl`	R	0	0	18	17	10	2	`srl $s1,$s2,10`
`beq`	I	4	17	18	25			`beq $s1,$s2,100`
`bne`	I	5	17	18	25			`bne $s1,$s2,100`
`slt`	R	0	18	19	17	0	42	`slt $s1,$s2,$s3`
`j`	J	2	2500					`j 10000` (see Section 2.9)
`jr`	R	0	31	0	0	0	8	`jr $ra`
`jal`	J	3	2500					`jal 10000` (see Section 2.9)
Field size		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	All MIPS instructions 32 bits
R-format	R	op	rs	rt	rd	shamt	funct	Arithmetic instruction format
I-format	I	op	rs	rt	address			Data transfer, branch format

1. Que instruções assembly MIPS R2000 representam as seguintes sequências de 32 bits?

- `00000010010100111000100000100000`
- `10001110010100010000000001100100`
- `00010010000110010000000000011001`
- `00000011111000000000000000001000`
- `00001100000000000000100111000100`

2. Traduza os fragmentos de código seguintes para sequências de instruções MIPS R2000:

- ```
  8616          add $t0, $s1, $s2
  8620          add $t1, $s3, $s4
  8624          sub $s0, $t0, $t1
  ```

- ```
  8616          lw  $t0, 32($s3)
  8620          add $t0, $s2, $t0
  8624          sw  $t0, 48($s3)
  ```

- ```
  8616          bne $s3, $s4, else
  8620          add $s0, $s1, $s2
  8624          j   exit
  8628  else:   sub $s0, $s1, $s2
  8632  exit:   ...
  ```

- ```
  3664  loop:   add $t1, $s3, $s3
  3668          add $t1, $t1, $t1
  3672          add $t1, $t1, $s6
  3676          lw  $t0, 0($t1)
  3680          bne $t0, $s5, exit
  3684          add $s3, $s3, $s4
  3688          j   loop
  3692  exit:   ...
  ```

- ```
  3664  proc:   addiu $sp, $sp, -12
  3668          sw    $t1, 8($sp)
  3672          sw    $t0, 4($sp)
  3676          sw    $s0, 0($sp)
  3680          add   $t0, $a0, $a1
  3684          add   $t1, $a2, $a3
  3688          sub   $s0, $t0, $t1
  3692          add   $v0, $s0, $zero
  ```
  ```

```
3696            lw    $s0, 0($sp)
3700            lw    $t0, 4($sp)
3704            lw    $t1, 8($sp)
3708            addiu $sp, $sp, 12
3712            jr $ra
```