#5 : MIPS Programming I

Computer Architecture 2021/2022

Ricardo Rocha

Computer Science Department, Faculty of Sciences, University of Porto

Arithmetic Instructions

add $s1, $s2, $s3	$s1 = $s2 + $s3	(add)
addu $s1, $s2, $s3	$s1 = $s2 + $s3	(add unsigned, no overflow)
addi $s1, $s2, 20	$s1 = $s2 + 20	(add immediate, sign-extend)
addiu $s1, $s2, 20	$s1 = $s2 + 20	(add immediate, no overflow)
sub $s1, $s2, $s3	$s1 = $s2 – $s3	(subtract)
mul $s1, $s2, $s3	$s1 = $s2 * $s3	(multiply)

Logical Instructions

and $s1, $s2, $s3	$s1 = $s2 & $s3	(and, bit-by-bit)
andi $s1, $s2, 20	$s1 = $s2 & 20	(and immediate)
or $s1, $s2, $s3	$s1 = $s2 | $s3	(or)
nor $s1, $s2, $s3	$s1 = ~ ($s2 | $s3)	(nor)
sll $s1, $s2, 10	$s1 = $s2 << 10	(shift left logical)
srl $s1, $s2, 10	$s1 = $s2 >> 10	(shift right logical)

Load Instructions

lw $s1, 20($s2)	$s1 = Mem[$s2 + 20]	(load word, from memory)
lh $s1, 20($s2)	$s1 = Mem[$s2 + 20]	(load half word, sign-extend)
lhu $s1, 20($s2)	$s1 = Mem[$s2 + 20]	(load half word, zero-extend)
lb $s1, 20($s2)	$s1 = Mem[$s2 + 20]	(load byte, sign-extend)
lbu $s1, 20($s2)	$s1 = Mem[$s2 + 20]	(load byte, zero-extend)
li $s1, 20	$s1 = 20	(load immediate)
la $s1, L	$s1 = L	(load address)

Store Instructions

sw $s1, 20($s2)	Mem[$s2 + 20] = $s1	(store word, to memory)
sh $s1, 20($s2)	Mem[$s2 + 20] = $s1	(store half word)
sb $s1, 20($s2)	Mem[$s2 + 20] = $s1	(store byte)

Branch Instructions

beq $s1, $s2, 25	if ($s1 == $s2) go to (PC+4+100)	(branch on equal)
beq $s1, $s2, L	if ($s1 == $s2) go to L	(branch on equal)
bne $s1, $s2, L	if ($s1 != $s2) go to L	(branch on not equal)
blt $s1, $s2, L	if ($s1 < $s2) go to L	(branch on less than)
bgt $s1, $s2, L	if ($s1 > $s2) go to L	(branch on greater than)
ble $s1, $s2, L	if ($s1 <= $s2) go to L	(branch on less than or equal)
slt $s1, $s2, $s3	if ($s2 < $s3) $s1 = 1 else $s1 = 0	(set on less than, for use with beq/bne)
slti $s1, $s2, 20	if ($s2 < 20) $s1 = 1 else $s1 = 0	(set on less than immediate)

Jump Instructions

j 2500	go to 10000	(jump to target address)
j L	go to L	(jump to target address)
jal L	$ra = PC+4; go to L	(jump and link, for procedure call)
jr $ra	go to $ra	(jump register, for procedure return)

Pseudo-Instructions

Most assembler instructions represent machine instructions one-to-one. To **simplify programming**, the assembler can also treat common variations of machine instructions as if they were instructions in their own right. Such instructions are called **pseudo-instructions**. The hardware need not implement the pseudo-instructions and register $at (assembler temporary) is reserved for this purpose.

li $s1, 20	→	**addiu $s1, $zero, 20**
move $t0, $t1	→	**addu $t0, $zero, $t1**
blt $s1, $s2, L	→	**slt $at, $s1, $s2**
		bne $at, $zero, L

Program Structure

```
        .data                # data segment (constants and global variables)
_b1:    .byte 1              # byte (8 bits) with value 1
_h1:    .half 10             # half word (16 bits) with value 10
_w1:    .word 100            # word (32 bits) with value 100
_a1:    .byte 1, 2, 3, 4     # array of 4 bytes with values 1, 2, 3 and 4
_a2:    .word 0:100          # array of 100 words with values 0
_s1:    .ascii  “abc\n”      # string not null terminated
_s2:    .asciiz “123”        # string null terminated”
_e1:    .space 100           # leave 100 bytes of space

        .text                # text segment (program instructions)
_main:                       # main procedure
        ...
        li $v0, 10           # load code 10 for system call exit()
        syscall              # exit()
```

System Calls

To request a service, **load the system call code into register $v0** and **arguments into registers $a0–$a3 or $f12** (floating point values).

Return values are put in **register $v0 or $f0** (floating-point results).

Service	System call code	Arguments	Result
print_int	1	$a0 = integer	
print_float	2	$f12 = float	
print_double	3	$f12 = double	
print_string	4	$a0 = string	
read_int	5		integer (in $v0)
read_float	6		float (in $f0)
read_double	7		double (in $f0)
read_string	8	$a0 = buffer, $a1 = length	
sbrk	9	$a0 = amount	address (in $v0)
exit	10		
print_char	11	$a0 = char	
read_char	12		char (in $v0)
open	13	$a0 = filename (string), $a1 = flags, $a2 = mode	file descriptor (in $a0)
read	14	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars read (in $a0)
write	15	$a0 = file descriptor, $a1 = buffer, $a2 = length	num chars written (in $a0)
close	16	$a0 = file descriptor	
exit2	17	$a0 = result	