# **Assembly Operations**

The right column gives the command followed by its arguments. Arguments beginning with r are registers. An imm argument is a number; its subscript is the maximum size in bits. For commands with a result, rd is the destination. Operations marked with \* are pseudoinstructions, which translate into other instructions.

### Arithmetic operations

Add add rd, rs, rt addi rd, rs,  ${\rm imm}_{16}$ Add immediate Subtract sub rd, rs, rt Multiply\* mulo rd, rs, rt Unsigned multiply\* mulou rd, rs, rt Divide s by t\*div rd, rs, rt rem rd, rs, rt Remainder\* Unsigned remainder\* remu rd, rs, rt Absolute value abs rd, rs Negate value\* neg rd, rs

### Bit operations

Count leading ones clo rd, rs Count leading zeros clz rd, rs Bitwise AND and rd, rs, rt Bitwise AND immediate andi rd, rs,  $imm_{16}$ Bitwise NOR nor rd, rs, rt Bitwise NOT not rd, rs Bitwise OR or rd, rs, rt Bitwise OR immediate ori rd, rs,  $imm_{16}$ Bitwise XOR xor rd, rs, rt Bitwise XOR immediate xori rd, rs, imm<sub>16</sub> Shift left (by immediate; fill with 0s) sll rd, rs, imm<sub>5</sub> Shift left (by register value; fill with 0s) sllv rd, rs, rt Shift right (by immediate; fill with 0s) srl rd, rs, imm<sub>5</sub> Shift right (by register value; fill with 0s) srlv rd, rs, rt Shift right arithmetic (by immediate; fill with MSB) sra rd, rs, imm<sub>5</sub> Shift right arithmetic (by register value; fill with MSB) srav rd, rs, rt Circular shift left\* rol rd, rs, rt Circular shift right\* ror rd, rs, rt Load immediate\*  $li rd, imm_{32}$ System call (see table below) syscall

The behavior of syscall is controlled by register v0. The following are useful codes:

- 1 Print int stored in a0
- 4 Print string whose address is stored in \$a0
- 5 Read int into v0
- 8 Read string into buffer at address a0 of length a1
- 10 Exit simulation
- 11 Print character stored in a0
- 12 Read character into v0

## Comparison, branch, and jump instructions

Set instructions place 1 or 0 in rd depending on the condition. For example, seq sets rd to 1 when rs = rt.

Set if equal	seq rd, rs, rt
Set if not equal	sne rd, rs, rt
Set if greater than*	sgt rd, rs, rt
Set if greater than or equal*	sge rd, rs, rt
Set if greater than unsigned*	sgtu rd, rs, rt
Set if greater than or equal unsigned*	sgeu rd, rs, rt
Set if less than	slt rd, rs, rt
Set if less than immediate	slti rt, rs, ${\sf imm}_{16}$
Set if less than or equal*	sle rd, rs, rt
Set if less than unsigned	sltu rd, rs, rt
Set if less than or equal unsigned*	sleu rd, rs, rt
Set if less than unsigned immediate	sltiu rd, rs, $imm_{16}$
Unconditional branch*	b label
Branch if equal	beq rs, rt, label
Branch if not equal	bne rs, rt, label
Branch if greater than*	bgt rs, rt, label
Branch if greater than or equal*	bge rs, rt, label
Branch if greater than or equal unsigned*	bgeu rs, rt, label
Branch if greater than unsigned*	bgtu rs, rt, label
Branch if less than*	blt rs, rt, label
Branch if less than or equal*	ble rs, rt, label
Branch if less than or equal unsigned*	bleu rs, rt, label
Branch if less than unsigned*	bltu rs, rt, label

## Memory instructions

```
"the following goes in the text segment"
                                         .text
"the following goes in the data segment"
                                         .data
"store str as a string"
                                         .ascii str
"store str as a null-terminated string"
                                        .asciiz str
"store these words in memory"
                                        .word w1 w2 ...
"reserve n bytes of space"
                                        .space n
"align on 2^n-byte boundary"
                                        .align n
Load address (not contents) *
                                        la rd, address
Load byte
                                        1b rd, address
Load unsigned byte
                                        lbu rd, address
Store byte
                                        sb rt, address
Load word
                                        lw rd, address
Store word
                                        sw rd, address
```

The normal load instructions for less than a word sign-extend the value; the unsigned versions do not. Addresses for memory instructions can be any of the following forms:

(register)	contents of register
imm	immediate
imm(register)	contents of register + immediate
label	address of label
label $\pm$ imm	address of label $\pm$ immediate
label $\pm$ imm(register)	address of label $\pm$ (immediate+register)