

Tinker Instruction Manual

Integer Arithmetic Instructions

`add rd, rs, rt`

Format:

0x18	r _d	r _s	r _t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s + r_t$$

Performs signed addition of two 64-bit signed values in registers r_s and r_t and stores the result in register r_d .

`addi rd, L`

0x19	r _d			L
4	9	14	19	31

Function:

$$r_d \leftarrow r_d + L$$

Adds the unsigned value L to the value in register r_d .

`sub rd, rs, rt`

Format:

0x1a	r _d	r _s	r _t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s - r_t$$

Performs signed subtraction of two 64-bit signed values in registers r_s and r_t and stores the result in register r_d .

`subi rd, L`

Format:

0x1b	r _d			L
4	9	14	19	31

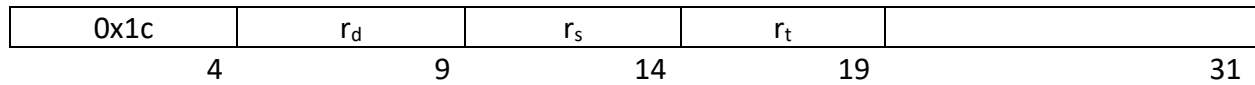
Function:

$$r_d \leftarrow r_d - L$$

Subtracts the unsigned value L from the value in register r_d .

$\text{mul } r_d, r_s, r_t$

Format:



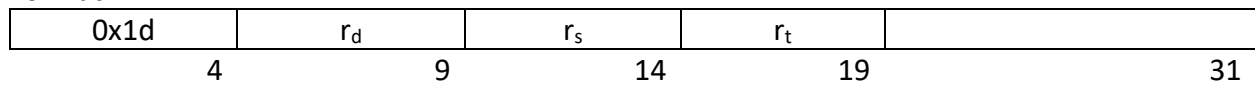
Function:

$$r_d \leftarrow r_s \times r_t$$

Performs signed multiplication of two 64-bit signed values in registers r_s and r_t and stores the result in register r_d .

$\text{div } r_d, r_s, r_t$

Format:



Function:

$$r_d \leftarrow r_s / r_t$$

Performs signed division of two 64-bit signed values in registers r_s and r_t and stores the result in register r_d .

Logic instructions

and r_d, r_s, r_t

Format:

0x0	r_d	r_s	r_t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s \& r_t$$

Performs bitwise “and” of two 64-bit values in registers r_s and r_t and stores the result in register r_d .

or r_d, r_s, r_t

Format:

0x1	r_d	r_s	r_t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s | r_t$$

Performs bitwise “or” of two 64-bit values in registers r_s and r_t and stores the result in register r_d .

xor r_d, r_s, r_t

Format:

0x2	r_d	r_s	r_t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s \wedge r_t$$

Performs bitwise “xor” of two 64-bit values in registers r_s and r_t and stores the result in register r_d .

not r_d, r_s

Format:

0x3	r_d	r_s		
4	9	14	19	31

Function:

$$r_d \leftarrow \sim r_s$$

Performs bitwise “not” (one’s complement) of a 64-bit value in register r_s and stores the result in register r_d .

shftr r_d, r_s, r_t

Format:

0x4	r_d	r_s	r_t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s \gg r_t$$

Shifts the value in register r_s to the right by the number of bits specified in the value in register r_t and stores the result in register r_d .

shftri r_d, L

Format:

0x5	r_d			L
4	9	14	19	31

Function:

$$r_d \leftarrow r_d \gg L$$

Shifts the value in register r_d to the right by the number of bits specified by L.

shftl r_d, r_s, r_t

Format:

0x6	r_d	r_s	r_t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s \ll r_t$$

Shifts the value in register r_s to the left by the number of bits specified in the value in register r_t and stores the result in register r_d .

shftli r_d, L

Format:

0x7	r_d			L
4	9	14	19	31

Function:

$$r_d \leftarrow r_d \ll L$$

Shifts the value in register r_d to the left by the number of bits specified by L.

Control instructions

`br r_d`

Format:

0x8	r_d			
4	9	14	19	31

Function:

$pc \leftarrow r_d$

Jumps to the instruction address specified by the value in register r_d .

`brr r_d`

Format:

0x9	r_d			
4	9	14	19	31

Function:

$pc \leftarrow pc + r_d$

Jumps to the instruction address specified by adding the value in register r_d to the program counter.

`brr L`

Format:

0xa				L
4	9	14	19	31

Function:

$pc \leftarrow pc + L$

Jumps to the instruction address specified by adding L to the program counter (L can be negative).

`brnz r_d, r_s`

Format:

0xb	r_d	r_s		
4	9	14	19	31

Function:

If $r_s = 0$

$pc \leftarrow pc + 4$

else

$pc \leftarrow r_d$

Jumps to the instruction address specified by the value in register r_d if r_s is nonzero, otherwise continue to the next instruction.

call r_d, r_s, r_t

Format:

0xc	r_d	r_s	r_t	
4	9	14	19	31

Function:

$\text{Mem}[r_{31} - 8] = \text{pc} + 4$

$\text{Pc} \leftarrow r_d$

Calls the function that starts at the address specified by r_d and stores the return address on the stack.

return

Format:

0xd				
4	9	14	19	31

Function:

$\text{pc} \leftarrow \text{Mem}[r_{31} - 8]$

Restores the program counter from the stack and returns to the caller.

brgt r_d, r_s, r_t

Format:

0xe	r_d	r_s	r_t	
4	9	14	19	31

Function:

If $r_s \leq r_t$

$\text{pc} \leftarrow \text{pc} + 4$

else

$\text{pc} \leftarrow r_d$

Jumps to the instruction address specified by the value in register r_d if r_s is greater than r_t , r_s and r_t being signed integers; otherwise continue to the next instruction.

Privileged instructions

priv r_d , r_s , r_t , L

Format:

0xf	r_d	r_s	r_t	L
4	9	14	19	31

Function:

The “priv” instruction is a privileged instruction whose function depends on the field L.

Currently, the following instructions are defined according to the value in L as follows:

- 0x0: Halt instruction. This stops the simulation. It is the last instruction run by a program.
- 0x3: Input instruction.
Function:
$$r_d \leftarrow \text{Input}[r_s]$$

Reads from the input port pointed to by the value in register r_s and stores it in register r_d .

By convention, port 0 is always connected to the keyboard, while port 1 is connected to the console output.
- 0x4: Output instruction.
Function:
$$\text{Output}[r_d] \leftarrow r_s$$

Reads the value in register r_s and writes it to the output port pointed to by the value in register r_d .

By convention, port 0 is always connected to the keyboard, while port 1 is connected to the console output.
- Any other value of L designates an illegal instruction.

Data Movement Instructions

`mov rd, (rs)(L)`

Format:

0x10	r _d	r _s		L
4	9	14	19	31

Function:

$r_d \leftarrow \text{Mem}[r_s + L]$

Reads the value in the memory location pointed to by the value composed of the value in register r_s as a base register and the literal value L as an index, and stores it in register r_d.

`mov rd, rs`

Format:

0x11	r _d	r _s		
4	9	14	19	31

Function:

$r_d \leftarrow r_s$

Reads the value in register r_s and stores it in register r_d.

`mov rd, L`

Format:

0x12	r _d			L
4	9	14	19	31

Function:

$r_d[52:63] \leftarrow L$

Sets bits 52:63 (inclusive) of register r_d to the value of L.

`mov (rd)(L), rs`

Format:

0x13	r _d	r _s		L
4	9	14	19	31

Function:

$\text{Mem}[r_d + L] \leftarrow r_s$

Reads the value in register r_s and stores it in the memory location pointed to by the value composed of the value in register r_d. as a base register and the literal L as an index.

Floating Point Instructions

`addf r_d, r_s, r_t`

Format:

0x14	r_d	r_s	r_t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s + r_t$$

Performs signed addition of two double precision values in registers r_s and r_t , and stores the result in register r_d .

`subf r_d, r_s, r_t`

Format:

0x15	r_d	r_s	r_t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s - r_t$$

Performs signed subtraction of two double precision values in registers r_s and r_t , and stores the result in register r_d .

`mulf r_d, r_s, r_t`

Format:

0x16	r_d	r_s	r_t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s \times r_t$$

Performs signed multiplication of two double precision values in registers r_s and r_t , and stores the result in register r_d .

`divf r_d, r_s, r_t`

Format:

0x17	r_d	r_s	r_t	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s / r_t$$

Performs signed division of two double precision values in registers r_s and r_t , and stores the result in register r_d .

Useful Macros

`in r_d, r_s`

Function:

$r_d \leftarrow \text{Input}[r_s]$

Reads from the input port pointed to by the value in register r_s and stores it in register r_d .

`out r_d, r_s`

Function:

$\text{Output}[r_d] \leftarrow r_s$

Reads the value in register r_s and writes it to the output port pointed to by the value in register r_d .

`clr r_d`

Function:

Set register r_d to zero.

`ld r_d, L`

Function:

Loads a 64-bit literal L into register r_d .

`push r_d`

Function:

Pushes the value contained in r_d on the stack.

`pop r_d`

Function:

Reads the value contained on top of the stack into r_d . The stack is popped.