

# Tinker Instruction Manual

## Integer Arithmetic Instructions

`add rd, rs, rt`

Format:

0x0	r <sub>d</sub>	r <sub>s</sub>	r <sub>t</sub>	
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`

0x1	r <sub>d</sub>			L
4	9	14	19	31

Function:

$$r_d \leftarrow r_s + L$$

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

`sub rd, rs, rt`

Format:

0x2	r <sub>d</sub>	r <sub>s</sub>	r <sub>t</sub>	
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:

0x3	r <sub>d</sub>			L
4	9	14	19	31

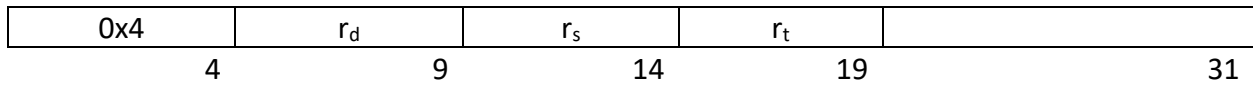
Function:

$$r_d \leftarrow r_s - L$$

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

`mul rd, rs, rt`

Format:



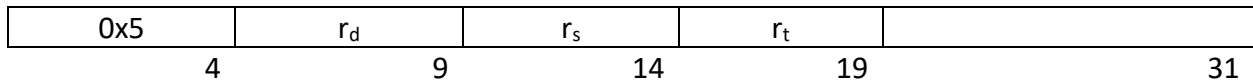
Function:

$$r_d \leftarrow r_s \times r_t$$

Performs signed multiplication of two 64-bit signed values in registers r<sub>s</sub> and r<sub>t</sub> and stores the result in register r<sub>d</sub>.

`div rd, rs, rt`

Format:



Function:

$$r_d \leftarrow r_s / r_t$$

Performs signed division of two 64-bit signed values in registers r<sub>s</sub> and r<sub>t</sub> and stores the result in register r<sub>d</sub>.

## Logic instructions

and  $r_d, r_s, r_t$

Format:

0x6	$r_d$	$r_s$	$r_t$	
4	9	14	19	31

Function:

$$r_d \leftarrow r_s \& r_t$$

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:

0x7	$r_d$	$r_s$	$r_t$	
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:

0x8	$r_d$	$r_s$	$r_t$	
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:

0x9	$r_d$	$r_s$		
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:

0xa	$r_d$	$r_s$	$r_t$	
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:

0xb	$r_d$			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:

0xc	$r_d$	$r_s$	$r_t$	
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:

0xd	$r_d$			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:

0x0e	$r_d$			
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:

0x0f	$r_d$			
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:

0x10				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:

0x11	$r_d$	$r_s$		
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:

0x12	$r_d$	$r_s$	$r_t$	
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:

0x13				
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:

0x14	$r_d$	$r_s$	$r_t$	
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.

Halt

Format:

0x1f				
4	9	14	19	31

Function:

Stops the processor. This is the last instruction in a program and should signal the end of the simulation.

## Data Movement Instructions

`mov rd, (rs)(L)`

Format:

0x15	r <sub>d</sub>	r <sub>s</sub>		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<sub>s</sub> as a base register and the literal value L as an index, and stores it in register r<sub>d</sub>.

`mov rd, rs`

Format:

0x16	r <sub>d</sub>	r <sub>s</sub>		
4	9	14	19	31

Function:

$r_d \leftarrow r_s$

Reads the value in register r<sub>s</sub> and stores it in register r<sub>d</sub>.

`mov rd, L`

Format:

0x17	r <sub>d</sub>			L
4	9	14	19	31

Function:

$r_d[52:63] \leftarrow L$

Sets bits 52:63 (inclusive) of register r<sub>d</sub> to the value of L.

`mov (rd)(L), rs`

Format:

0x18	r <sub>d</sub>	r <sub>s</sub>		L
4	9	14	19	31

Function:

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

Reads the value in register r<sub>s</sub> and stores it in the memory location pointed to by the value composed of the value in register r<sub>d</sub>. as a base register and the literal L as an index.

## Floating Point Instructions

addf  $r_d, r_s, r_t$

Format:

0x19	$r_d$	$r_s$	$r_t$	
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:

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 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:

0x1b	$r_d$	$r_s$	$r_t$	
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:

0x1c	$r_d$	$r_s$	$r_t$	
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  $r_d$ .



## I/O Instructions

in  $r_d, r_s$

Format:

0x1d	$r_d$	$r_s$		
4	9	14	19	31

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  $r_d, r_s$

Format:

0x1e	$r_d$	$r_s$		
4	9	14	19	31

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  $r_d$ .

## Useful Macros

clr  $r_d$  ==

```
xor  $r_d$ ,  $r_d$ ,  $r_d$ 
```

ld  $r_d$ , L ==

```
clr  $r_d$   
addi  $r_d$ , L[0:11]  
shiftli  $r_d$ , 12  
addi  $r_d$ , L[12:23]  
shiftli  $r_d$ , 12  
addi  $r_d$ , L[24, 35]  
shiftli  $r_d$ , 12  
addi  $r_d$ , L[36, 47]  
shiftli  $r_d$ , 12  
addi  $r_d$ , L[48, 59]  
shiftli  $r_d$ , 4  
addi  $r_d$ , L[60, 63]
```

push  $r_d$  ==

```
mov ( $r_{31}$ )(-8),  $r_d$   
subi  $r_{31}$ , 8
```

pop  $r_d$  ==

```
mov  $r_d$ , ( $r_{31}$ )  
addi  $r_{31}$ , 8
```