Chapter 25

RISC-V Assembly Programmer's Handbook

This chapter is a placeholder for an assembly programmer's manual.

Table 25.1 lists the assembler mnemonics for the `x` and `f` registers and their role in the first standard calling convention.

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	—
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	—
x4	tp	Thread pointer	—
x5	t0	Temporary/alternate link register	Caller
x6–7	t1–2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10–11	a0–1	Function arguments/return values	Caller
x12–17	a2–7	Function arguments	Caller
x18–27	s2–11	Saved registers	Callee
x28–31	t3–6	Temporaries	Caller
f0–7	ft0–7	FP temporaries	Caller
f8–9	fs0–1	FP saved registers	Callee
f10–11	fa0–1	FP arguments/return values	Caller
f12–17	fa2–7	FP arguments	Caller
f18–27	fs2–11	FP saved registers	Callee
f28–31	ft8–11	FP temporaries	Caller

Table 25.1: Assembler mnemonics for RISC-V integer and floating-point registers, and their role in the first standard calling convention.

There may be future different calling conventions, but note that registers `x1`*,* `x2`*, and* `x5` *have special meanings encoded in the standard ISA and/or the compressed extension.*

Tables 25.2 and 25.3 contain a listing of standard RISC-V pseudoinstructions.

pseudoinstruction	Base Instruction(s)	Meaning
`la rd, symbol` (*non-PIC*)	`auipc rd, delta[31:12]+delta[11]` `addi rd, rd, delta[11:0]`	Load absolute address, where `delta` = `symbol` − `pc`
`la rd, symbol` (*PIC*)	`auipc rd, delta[31:12]+delta[11]` `l{w|d} rd, rd, delta[11:0]`	Load absolute address, where `delta` = `GOT[symbol]` − `pc`
`lla rd, symbol`	`auipc rd, delta[31:12]+delta[11]` `addi rd, rd, delta[11:0]`	Load local address, where `delta` = `symbol` − `pc`
`l{b|h|w|d} rd, symbol`	`auipc rd, delta[31:12]+delta[11]` `l{b|h|w|d} rd, delta[11:0](rd)`	Load global
`s{b|h|w|d} rd, symbol, rt`	`auipc rt, delta[31:12]+delta[11]` `s{b|h|w|d} rd, delta[11:0](rt)`	Store global
`fl{w|d} rd, symbol, rt`	`auipc rt, delta[31:12]+delta[11]` `fl{w|d} rd, delta[11:0](rt)`	Floating-point load global
`fs{w|d} rd, symbol, rt`	`auipc rt, delta[31:12]+delta[11]` `fs{w|d} rd, delta[11:0](rt)`	Floating-point store global

The base instructions use `pc`*-relative addressing, so the linker subtracts* `pc` *from* `symbol` *to get* `delta`*. The linker adds* `delta[11]` *to the 20-bit high part, counteracting sign extension of the 12-bit low part.*

`nop`	`addi x0, x0, 0`	No operation
`li rd, immediate`	*Myriad sequences*	Load immediate
`mv rd, rs`	`addi rd, rs, 0`	Copy register
`not rd, rs`	`xori rd, rs, -1`	One's complement
`neg rd, rs`	`sub rd, x0, rs`	Two's complement
`negw rd, rs`	`subw rd, x0, rs`	Two's complement word
`sext.w rd, rs`	`addiw rd, rs, 0`	Sign extend word
`seqz rd, rs`	`sltiu rd, rs, 1`	Set if = zero
`snez rd, rs`	`sltu rd, x0, rs`	Set if ≠ zero
`sltz rd, rs`	`slt rd, rs, x0`	Set if < zero
`sgtz rd, rs`	`slt rd, x0, rs`	Set if > zero
`fmv.s rd, rs`	`fsgnj.s rd, rs, rs`	Copy single-precision register
`fabs.s rd, rs`	`fsgnjx.s rd, rs, rs`	Single-precision absolute value
`fneg.s rd, rs`	`fsgnjn.s rd, rs, rs`	Single-precision negate
`fmv.d rd, rs`	`fsgnj.d rd, rs, rs`	Copy double-precision register
`fabs.d rd, rs`	`fsgnjx.d rd, rs, rs`	Double-precision absolute value
`fneg.d rd, rs`	`fsgnjn.d rd, rs, rs`	Double-precision negate
`beqz rs, offset`	`beq rs, x0, offset`	Branch if = zero
`bnez rs, offset`	`bne rs, x0, offset`	Branch if ≠ zero
`blez rs, offset`	`bge x0, rs, offset`	Branch if ≤ zero
`bgez rs, offset`	`bge rs, x0, offset`	Branch if ≥ zero
`bltz rs, offset`	`blt rs, x0, offset`	Branch if < zero
`bgtz rs, offset`	`blt x0, rs, offset`	Branch if > zero
`bgt rs, rt, offset`	`blt rt, rs, offset`	Branch if >
`ble rs, rt, offset`	`bge rt, rs, offset`	Branch if ≤
`bgtu rs, rt, offset`	`bltu rt, rs, offset`	Branch if >, unsigned
`bleu rs, rt, offset`	`bgeu rt, rs, offset`	Branch if ≤, unsigned

Table 25.2: RISC-V pseudoinstructions.

pseudoinstruction	Base Instruction	Meaning
`j offset`	`jal x0, offset`	Jump
`jal offset`	`jal x1, offset`	Jump and link
`jr rs`	`jalr x0, 0(rs)`	Jump register
`jalr rs`	`jalr x1, 0(rs)`	Jump and link register
`ret`	`jalr x0, 0(x1)`	Return from subroutine
`call offset`	`auipc x1, offset[31:12] + offset[11]` `jalr x1, offset[11:0](x1)`	Call far-away subroutine
`tail offset`	`auipc x6, offset[31:12] + offset[11]` `jalr x0, offset[11:0](x6)`	Tail call far-away subroutine
`fence`	`fence iorw, iorw`	Fence on all memory and I/O
`rdinstret[h] rd`	`csrrs rd, instret[h], x0`	Read instructions-retired counter
`rdcycle[h] rd`	`csrrs rd, cycle[h], x0`	Read cycle counter
`rdtime[h] rd`	`csrrs rd, time[h], x0`	Read real-time clock
`csrr rd, csr`	`csrrs rd, csr, x0`	Read CSR
`csrw csr, rs`	`csrrw x0, csr, rs`	Write CSR
`csrs csr, rs`	`csrrs x0, csr, rs`	Set bits in CSR
`csrc csr, rs`	`csrrc x0, csr, rs`	Clear bits in CSR
`csrwi csr, imm`	`csrrwi x0, csr, imm`	Write CSR, immediate
`csrsi csr, imm`	`csrrsi x0, csr, imm`	Set bits in CSR, immediate
`csrci csr, imm`	`csrrci x0, csr, imm`	Clear bits in CSR, immediate
`frcsr rd`	`csrrs rd, fcsr, x0`	Read FP control/status register
`fscsr rd, rs`	`csrrw rd, fcsr, rs`	Swap FP control/status register
`fscsr rs`	`csrrw x0, fcsr, rs`	Write FP control/status register
`frrm rd`	`csrrs rd, frm, x0`	Read FP rounding mode
`fsrm rd, rs`	`csrrw rd, frm, rs`	Swap FP rounding mode
`fsrm rs`	`csrrw x0, frm, rs`	Write FP rounding mode
`frflags rd`	`csrrs rd, fflags, x0`	Read FP exception flags
`fsflags rd, rs`	`csrrw rd, fflags, rs`	Swap FP exception flags
`fsflags rs`	`csrrw x0, fflags, rs`	Write FP exception flags

Table 25.3: RISC-V pseudoinstructions.

```
# Spike compatible calls
GetCWD =     17
Close =      57
LSeek =      62
Read =       63
Write =      64
Exit2 =      93
Open =       1024

# Almost compatible
Sbrk =       9
Time =       30

# MARS / SPIM compatability
PrintInt =    1
PrintFloat =  2
PrintDouble = 3
PrintString = 4
ReadInt =     5
ReadFloat =   6
ReadDouble = 7
ReadString =  8
Exit =       10
PrintChar =  11
ReadChar =   12
MidiOut =    31
Sleep =      32
MidiOutSync = 33
PrintIntHex = 34
PrintIntBinary = 35
PrintIntUnsigned = 36
RandSeed =   40
RandInt =    41
RandIntRange = 42
RandFloat =  43
RandDouble = 44
ConfirmDialog = 50
InputDialogInt = 51
InputDialogFloat = 52
InputDialogDouble = 53
InputDialogString = 54
MessageDialog = 55
MessageDialogInt = 56
MessageDialogDouble = 58
```

MessageDialogString = 59

# Not compatible
# Collided with Close (57)
MessageDialogFloat = 60