OTTER (RV32I) Assembly Instructions

1 Base Instructions

Format	Base Instruction	Name	Description (in Verilog)
U	`lui rd,imm`	Load Upper Immediate	R[rd] = {imm,12'b0}
U	`auipc rd,imm`	Add Upper Immediate to PC	R[rd] = PC + {imm,12'b0}
J	`jal rd,imm`	Jump and Link	R[rd] = PC+4;
			PC = PC + {imm,1'b0}
I	`jalr rd,rs1,imm`	Jump and Link Register	R[rd] = PC+4;
			PC = R[rs1] + imm
B	`beq rs1,rs2,imm`	Branch Equal	if(R[rs1]==R[rs2]) PC=PC+{imm,1'b0}
B	`bne rs1,rs2,imm`	Branch Not Equal	if(R[rs1]!=R[rs2]) PC=PC+{imm,1'b0}
B	`blt rs1,rs2,imm`	Branch Less Than	if(R[rs1]<R[rs2]) PC=PC+{imm,1'b0}
B	`bge rs1,rs2,imm`	Branch Greater than Equal	if(R[rs1]>= R[rs2]) PC=PC+imm,1'b0
B	`bltu rs1,rs2,imm`	Branch Less Than Unsigned	if(R[rs1]<R[rs2]) PC=PC+imm,1'b0
B	`bgeu rs1,rs2,imm`	Branch Greater than Equal Unsigned	if(R[rs1]>=R[rs2]) PC=PC+imm,1'b0
I	`lb rd,imm(rs1)`	Load Byte	R[rd] = 24'bM[R[rs1]+imm](7),M[R[rs1]+imm](7:0)
I	`lh rd,imm(rs1)`	Load Half	R[rd] = 16'bM[R[rs1]+imm](7),M[R[rs1]+imm](15:0)
I	`lw rd,imm(rs1)`	Load Word	R[rd] = M[R[rs1]+imm]
I	`lbu rd,imm(rs1)`	Load Byte Unsigned	R[rd] = 24'b0,M[R[rs1]+imm](7:0)
I	`lhu rd,imm(rs1)`	Load Half Unsigned	R[rd] = 16'b0,M[R[rs1]+imm](15:0)
S	`sb rs2,imm(rs1)`	Store Byte	M[R[rs1]+imm](7:0) = R[rs2](7:0)
S	`sh rs2,imm(rs1)`	Store Half	M[R[rs1]+imm](15:0) = R[rs2](15:0)
S	`sw rs2,imm(rs1)`	Store Word	M[R[rs1]+imm] = R[rs2]
I	`addi rd,rs1,imm`	Add Immediate	R[rd] = R[rs1] + imm
I	`slti rd,rs1,imm`	Set Less Than Immediate	R[rd] = ($signed(R[rs1]) < $signed(imm)) ? 1 : 0
I	`sltiu rd,rs1,imm`	Set Less Than Immediate Unsigned	R[rd] = (R[rs1] < imm) ? 1 : 0
I	`xori rd,rs1,imm`	Xor Immediate	R[rd] = R[rs1] ∧ imm
I	`ori rd,rs1,imm`	Or Immediate	R[rd] = R[rs1] | imm
I	`andi rd,rs1,imm`	And Immediate	R[rd] = R[rs1] & imm
I	`slli rd,rs1,imm`	Shift Left Logical Immediate	R[rd] = R[rs1] << imm(4:0)
I	`srli rd,rs1,imm`	Shift Right Logical Immediate	R[rd] = R[rs1] >> imm(4:0)
I	`srai rd,rs1,imm`	Shift Right Arithmetic Immediate	R[rd] = $signed(R[rs1]) >>> imm(4:0)
R	`add rd,rs1,rs2`	Add	R[rd] = R[rs1] + R[rs2]
R	`sub rd,rs1,rs2`	Subtract	R[rd] = R[rs1] - R[rs2]
R	`sll rd,rs1,rs2`	Shift Left Logical	R[rd] = R[rs1] << R[rs2](4:0)
R	`slt rd,rs1,rs2`	Set Less Than	R[rd] = ($signed(R[rs1]) < $signed(R[rs2])) ? 1 : 0
R	`sltu rd,rs1,rs2`	Set Less Than Unsigned	R[rd] = (R[rs1] < R[rs2]) ? 1 : 0
R	`xor rd,rs1,rs2`	Xor	R[rd] = R[rs1] ∧ R[rs2]
R	`srl rd,rs1,rs2`	Shift Right Logical	R[rd] = R[rs1] >> R[rs2](4:0)
R	`sra rd,rs1,rs2`	Shift Right Arithmetic	R[rd] = $signed(R[rs1]) >>> R[rs2](4:0)
R	`or rd,rs1,rs2`	Or	R[rd] = R[rs1] | R[rs2]
R	`and rd,rs1,rs2`	And	R[rd] = R[rs1] & R[rs2]
	`csrrw rd,csr,rs1`	csrrw	R[rd] = csr, csr = R[rs1]
	`mret`	mret	PC = mepc

Table 1: OTTER (RV32I) Instructions.

2 Psuedo Instructions

pseudoinstruction	Base Instruction(s)	Meaning
`la rd, symbol`	`auipc rd, delta`$[31:12] +$ `delta`$[11]$ `addi rd, rd, delta[11:0]`	Load absolute address, where `delta` $=$ `symbol` $-$ `pc`
`lw rd, symbol`	`auipc rd, delta`$[31:12] +$ `delta`$[11]$ `lw rd, delta[11:0](rd)`	Load global
`sw rd, symbol, rt`	`auipc rt, delta`$[31:12] +$ `delta`$[11]$ `sw rd, delta[11:0](rt)`	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
`seqz rd, rs`	`sltiu rd, rs, 1`	Set if $=$ zero
`snez rd, rs`	`sltu rd, x0, rs`	Set if $\neq$ zero
`sltz rd, rs`	`slt rd, rs, x0`	Set if $<$ zero
`sgtz rd, rs`	`slt rd, x0, rs`	Set if $>$ zero
`beqz rs, offset`	`beq rs, x0, offset`	Branch if $=$ zero
`bnez rs, offset`	`bne rs, x0, offset`	Branch if $\neq$ zero
`blez rs, offset`	`bge x0, rs, offset`	Branch if $\leq$ zero
`bgez rs, offset`	`bge rs, x0, offset`	Branch if $\geq$ 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 $\leq$
`bgtu rs, rt, offset`	`bltu rt, rs, offset`	Branch if $>$, unsigned
`bleu rs, rt, offset`	`bgeu rt, rs, offset`	Branch if $\leq$, unsigned
`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

Table 2: RISC-V pseudoinstructions.

3 RV32I Registers

Table 3 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

Table 3: Assembler mnemonics for RISC-V integer and floating-point registers, and their role in the first standard calling convention.

4 Memory

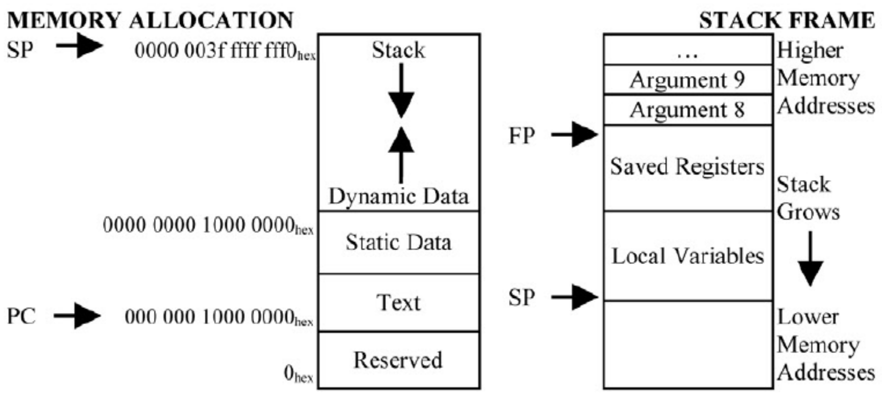

Figure 1: Memory Allocations

5 Instruction Format Listing

31 27 26 25	24 20	19 15	14 12	11 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type
imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	S-type
imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	B-type
imm[31:12]				rd	opcode	U-type
imm[20|10:1|11|19:12]				rd	opcode	J-type

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Table 4: RV32I Instruction Formats

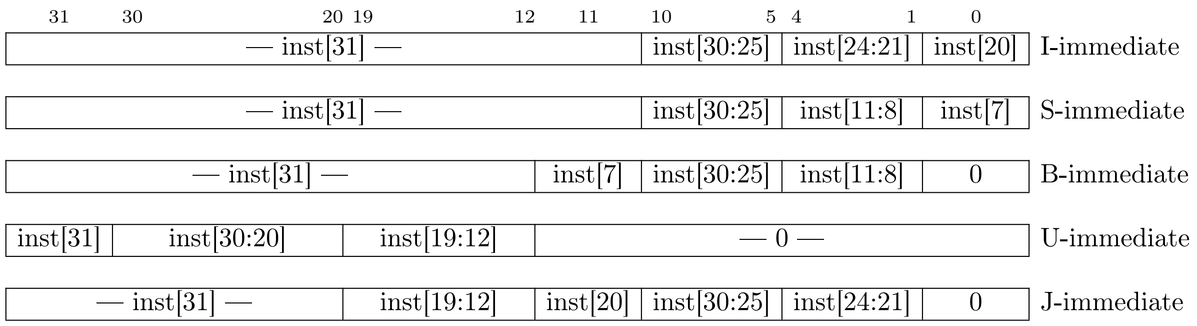

Figure 2: Types of immediate produced by RISC-V instructions. The fields are labeled with the instruction bits used to construct their value. Sign extension always uses inst[31].

Prefix	Symbol	Size (Power of 10)	Size (Power of 2)
kilo-	K	10^3	2^{10}
mega-	M	10^6	2^{20}
giga-	G	10^9	2^{30}
tera-	T	10^{12}	2^{40}
peta-	P	10^{15}	2^{50}
exa-	E	10^{18}	2^{60}
zetta-	Z	10^{21}	2^{70}
yotta-	Y	10^{24}	2^{80}

Table 5: Size prefixes and symbols

6 Machine/System Instruction Format Listing

31 27 26 25	24 20	19 15	14 12	11 7	6 0	
imm[11:0]		rs1	funct3	rd	opcode	I-type
RV32 Machine/System Instructions						
csr		rs1	001	rd	1110011	CSRRW
0011000	00010	00000	000	00000	1110011	MRET

Table 6: RV32 Machine/System Instruction Formats

pseudoinstruction	Base Instruction	Meaning
`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

Table 7: Pseudoinstructions for accessing control and status registers.