RV32I Reference Card

CS-173 — Fundamentals of Digital Systems

12th May 2025 © EPFL

1. Assembler Directives

Directive	Effect
`.text`	Switch to the code segment and place what follows there
`.data`	Switch to the data segment and place what follows there
`.asciiz "Hello, World!"`	Place at current location an ASCII string followed by a null-terminator
`.byte 0xC, 0xA, 0xF`	Place at current location value(s) as byte(s)
`.word 0xCAFE`	Place at current location value(s) as 32-bit word(s)
`.equ coffee, 0xCAFE`	Define a symbol as a constant

2. Registers

Register	Mnemonic	Description
x0	zero	Hard-wired zero
x1–x31		General registers; Temporaries t0–t2 and t3–t6 correspond to x5–x7 and x28–x31, resp.
pc		Program counter

3. Instruction Types and Encodings

	31–25	24–20	19–15	14–12	11–7	6–0	
R	funct7	rs2	rs1	funct3	rd	opcode	Register-Register
I	imm[11:0]		rs1	funct3	rd	opcode	Register-Immediate
I	funct7	imm[4:0]	rs1	funct3	rd	opcode	Register-Immediate Shift
S	imm[11:5]	rs2	rs1	funct3	imm[4:0]	opcode	Store
B	imm[12|10:5]	rs2	rs1	funct3	imm[4:1|11]	opcode	Branch
U	imm[31:12]				rd	opcode	Upper Immediate
J	imm[20|10:1|11|19:12]				rd	opcode	Jump

4. Standard Instructions

Instruction or Pseudoinstruction		Pseudocode	Type	funct7	funct3	opcode
Branch						
`beq`	`rs1, rs2, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} = \texttt{rs2}$	B		0x0	0x63
`bne`	`rs1, rs2, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} \neq \texttt{rs2}$	B		0x1	0x63
`blt`	`rs1, rs2, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} <_s \texttt{rs2}$	B		0x4	0x63
`bltu`	`rs1, rs2, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} <_u \texttt{rs2}$	B		0x6	0x63
`bge`	`rs1, rs2, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} \geq_s \texttt{rs2}$	B		0x5	0x63
`bgeu`	`rs1, rs2, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} \geq_u \texttt{rs2}$	B		0x7	0x63
`ble`	`rs1, rs2, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} \leq_s \texttt{rs2}$		`bge`	`rs2, rs1, imm`	
`bleu`	`rs1, rs2, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} \leq_u \texttt{rs2}$		`bgeu`	`rs2, rs1, imm`	
`bgt`	`rs1, rs2, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} >_s \texttt{rs2}$		`blt`	`rs2, rs1, imm`	
`bgtu`	`rs1, rs2, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} >_u \texttt{rs2}$		`bltu`	`rs2, rs1, imm`	
`beqz`	`rs1, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} = 0$		`beq`	`rs1, zero, imm`	
`bnez`	`rs1, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} \neq 0$		`bne`	`rs1, zero, imm`	
`bltz`	`rs1, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} <_s 0$		`blt`	`rs1, zero, imm`	
`bgez`	`rs1, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} \geq_s 0$		`bge`	`rs1, zero, imm`	
`blez`	`rs1, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} \leq_s 0$		`bge`	`zero, rs1, imm`	
`bgtz`	`rs1, imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$, if $\texttt{rs1} >_s 0$		`blt`	`zero, rs1, imm`	
Jump						
`j`	`imm`	$\texttt{pc} \leftarrow \texttt{pc} + \text{sext}(\texttt{imm} \ll 1)$		`jal`	`zero, imm`	

4. Standard Instructions (cont'd)

Instruction or Pseudoinstruction		Pseudocode	Type	funct7	funct3	opcode
Move						
`nop`		*Nothing*		`addi zero, zero, 0`		
`mv`	`rd, rs1`	rd ← rs1		`addi rd, rs1, 0`		
`li`	`rd, imm`	rd ← imm†		*Various translations*		
`la`	`rd, imm`	rd ← address of imm§		*Various translations*		
`lui`	`rd, imm`	rd ← imm ≪ 12	U			0x37
Arithmetic						
`add`	`rd, rs1, rs2`	rd ← rs1 + rs2	R	0x00	0x0	0x33
`addi`	`rd, rs1, imm`	rd ← rs1 + sext(imm)	I		0x0	0x13
`sub`	`rd, rs1, rs2`	rd ← rs1 − rs2	R	0x20	0x0	0x33
`neg`	`rd, rs1`	rd ← −rs1		`sub rd, zero, rs1`		
Logical						
`xor`	`rd, rs1, rs2`	rd ← rs1 ^ rs2	R	0x00	0x4	0x33
`xori`	`rd, rs1, imm`	rd ← rs1 ^ sext(imm)	I		0x4	0x13
`or`	`rd, rs1, rs2`	rd ← rs1 | rs2	R	0x00	0x6	0x33
`ori`	`rd, rs1, imm`	rd ← rs1 | sext(imm)	I		0x6	0x13
`and`	`rd, rs1, rs2`	rd ← rs1 & rs2	R	0x00	0x7	0x33
`andi`	`rd, rs1, imm`	rd ← rs1 & sext(imm)	I		0x7	0x13
`not`	`rd, rs1`	rd ← ∼rs1		`xori rd, rs1, -1`		
Shift						
`sll`	`rd, rs1, rs2`	rd ← rs1 ≪ rs2	R	0x00	0x1	0x33
`slli`	`rd, rs1, imm`	rd ← rs1 ≪ imm	I	0x00	0x1	0x13
`srl`	`rd, rs1, rs2`	rd ← rs1 $\gg_u$ rs2	R	0x00	0x5	0x33
`srli`	`rd, rs1, imm`	rd ← rs1 $\gg_u$ imm	I	0x00	0x5	0x13
`sra`	`rd, rs1, rs2`	rd ← rs1 $\gg_s$ rs2	R	0x20	0x5	0x33
`srai`	`rd, rs1, imm`	rd ← rs1 $\gg_s$ imm	I	0x20	0x5	0x13
Compare						
`slt`	`rd, rs1, rs2`	rd ← rs1 $<_s$ rs2	R	0x00	0x2	0x33
`slti`	`rd, rs1, imm`	rd ← rs1 $<_s$ sext(imm)	I		0x2	0x13
`sltu`	`rd, rs1, rs2`	rd ← rs1 $<_u$ rs2	R	0x00	0x3	0x33
`sltiu`	`rd, rs1, imm`	rd ← rs1 $<_u$ sext(imm)	I		0x3	0x13
`seqz`	`rd, rs1`	rd ← rs1 = 0		`sltiu rd, rs1, 1`		
`snez`	`rd, rs1`	rd ← rs1 ≠ 0		`sltu rd, zero, rs1`		
`sltz`	`rd, rs1`	rd ← rs1 $<_s$ 0		`slt rd, rs1, zero`		
`sgtz`	`rd, rs1`	rd ← rs1 $>_s$ 0		`slt rd, zero, rs1`		
Memory						
`lb`	`rd, imm(rs1)`	rd ← sext(mem[rs1 + sext(imm)][7 : 0])	I		0x0	0x03
`lbu`	`rd, imm(rs1)`	rd ← zext(mem[rs1 + sext(imm)][7 : 0])	I		0x4	0x03
`lw`	`rd, imm(rs1)`	rd ← mem[rs1 + sext(imm)]	I		0x2	0x03
`sb`	`rs2, imm(rs1)`	mem[rs1 + sext(imm)] ← rs2[7 : 0]	S		0x0	0x23
`sw`	`rs2, imm(rs1)`	mem[rs1 + sext(imm)] ← rs2	S		0x2	0x23

∼	Bitwise NOT	≪	Left shift	mem	Memory access
&	Bitwise AND	≫	Right shift	sext	Sign extend
|	Bitwise OR	op_s	Signed *op*eration	zext	Zero extend
^	Bitwise XOR	op_u	Unsigned *op*eration		

† Use `li` if `imm` is a symbol defined using `.equ` or a number

§ Use `la` if `imm` is a symbol defined as `label:`