RV32I Reference Card

CS-200 — Computer Architecture

10th July 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	Preserved across Call?
x0	zero	Hard-wired zero	—
x1	ra	Return Address	No
x2	sp	Stack Pointer	Yes
x3	gp	Global Pointer	—
x4	tp	Thread Pointer	—
x5	t0	Temporary/alternate link register	No
x6–x7	t1–t2	Temporaries	No
x8	s0/fp	Saved register/Frame Pointer	Yes
x9	s1	Saved register	Yes
x10–x11	a0–a1	Function Arguments/return values	No
x12–x17	a2–a7	Function Arguments	No
x18–x27	s2–s11	Saved registers	Yes
x28–x31	t3–t6	Temporaries	No
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 or Translation	funct3	opcode
Move						
`nop`		*Nothing*		`addi zero, zero, 0`		
`mv`	`rd, rs1`	$\texttt{rd} \leftarrow \texttt{rs1}$		`addi rd, rs1, 0`		
`li`	`rd, imm`	$\texttt{rd} \leftarrow \texttt{imm}$[†]		*Various translations*		
`la`	`rd, imm`	$\texttt{rd} \leftarrow$ address of `imm`[§]		*Various translations*		
`lui`	`rd, imm`	$\texttt{rd} \leftarrow \texttt{imm} \ll 12$	U			0x37
`auipc`	`rd, imm`	$\texttt{rd} \leftarrow \texttt{pc} + (\texttt{imm} \ll 12)$	U			0x17
Arithmetic						
`add`	`rd, rs1, rs2`	$\texttt{rd} \leftarrow \texttt{rs1} + \texttt{rs2}$	R	0x00	0x0	0x33
`addi`	`rd, rs1, imm`	$\texttt{rd} \leftarrow \texttt{rs1} + \text{sext}(\texttt{imm})$	I		0x0	0x13
`sub`	`rd, rs1, rs2`	$\texttt{rd} \leftarrow \texttt{rs1} - \texttt{rs2}$	R	0x20	0x0	0x33
`neg`	`rd, rs1`	$\texttt{rd} \leftarrow -\texttt{rs1}$		`sub rd, zero, rs1`		
Logical						
`xor`	`rd, rs1, rs2`	$\texttt{rd} \leftarrow \texttt{rs1} \mathbin{\hat{}} \texttt{rs2}$	R	0x00	0x4	0x33
`xori`	`rd, rs1, imm`	$\texttt{rd} \leftarrow \texttt{rs1} \mathbin{\hat{}} \text{sext}(\texttt{imm})$	I		0x4	0x13
`or`	`rd, rs1, rs2`	$\texttt{rd} \leftarrow \texttt{rs1} \mid \texttt{rs2}$	R	0x00	0x6	0x33
`ori`	`rd, rs1, imm`	$\texttt{rd} \leftarrow \texttt{rs1} \mid \text{sext}(\texttt{imm})$	I		0x6	0x13
`and`	`rd, rs1, rs2`	$\texttt{rd} \leftarrow \texttt{rs1} \,\&\, \texttt{rs2}$	R	0x00	0x7	0x33
`andi`	`rd, rs1, imm`	$\texttt{rd} \leftarrow \texttt{rs1} \,\&\, \text{sext}(\texttt{imm})$	I		0x7	0x13
`not`	`rd, rs1`	$\texttt{rd} \leftarrow \sim\texttt{rs1}$		`xori rd, rs1, -1`		
Shift						
`sll`	`rd, rs1, rs2`	$\texttt{rd} \leftarrow \texttt{rs1} \ll \texttt{rs2}$	R	0x00	0x1	0x33
`slli`	`rd, rs1, imm`	$\texttt{rd} \leftarrow \texttt{rs1} \ll \texttt{imm}$	I	0x00	0x1	0x13
`srl`	`rd, rs1, rs2`	$\texttt{rd} \leftarrow \texttt{rs1} \gg_u \texttt{rs2}$	R	0x00	0x5	0x33
`srli`	`rd, rs1, imm`	$\texttt{rd} \leftarrow \texttt{rs1} \gg_u \texttt{imm}$	I	0x00	0x5	0x13
`sra`	`rd, rs1, rs2`	$\texttt{rd} \leftarrow \texttt{rs1} \gg_s \texttt{rs2}$	R	0x20	0x5	0x33
`srai`	`rd, rs1, imm`	$\texttt{rd} \leftarrow \texttt{rs1} \gg_s \texttt{imm}$	I	0x20	0x5	0x13
Compare						
`slt`	`rd, rs1, rs2`	$\texttt{rd} \leftarrow \texttt{rs1} <_s \texttt{rs2}$	R	0x00	0x2	0x33
`slti`	`rd, rs1, imm`	$\texttt{rd} \leftarrow \texttt{rs1} <_s \text{sext}(\texttt{imm})$	I		0x2	0x13
`sltu`	`rd, rs1, rs2`	$\texttt{rd} \leftarrow \texttt{rs1} <_u \texttt{rs2}$	R	0x00	0x3	0x33
`sltiu`	`rd, rs1, imm`	$\texttt{rd} \leftarrow \texttt{rs1} <_u \text{sext}(\texttt{imm})$	I		0x3	0x13
`seqz`	`rd, rs1`	$\texttt{rd} \leftarrow \texttt{rs1} = 0$		`sltiu rd, rs1, 1`		
`snez`	`rd, rs1`	$\texttt{rd} \leftarrow \texttt{rs1} \neq 0$		`sltu rd, zero, rs1`		
`sltz`	`rd, rs1`	$\texttt{rd} \leftarrow \texttt{rs1} <_s 0$		`slt rd, rs1, zero`		
`sgtz`	`rd, rs1`	$\texttt{rd} \leftarrow \texttt{rs1} >_s 0$		`slt rd, zero, rs1`		
Memory						
`lb`	`rd, imm(rs1)`	$\texttt{rd} \leftarrow \text{sext}(\text{mem}[\texttt{rs1} + \text{sext}(\texttt{imm})][7:0])$	I		0x0	0x03
`lbu`	`rd, imm(rs1)`	$\texttt{rd} \leftarrow \text{zext}(\text{mem}[\texttt{rs1} + \text{sext}(\texttt{imm})][7:0])$	I		0x4	0x03
`lh`	`rd, imm(rs1)`	$\texttt{rd} \leftarrow \text{sext}(\text{mem}[\texttt{rs1} + \text{sext}(\texttt{imm})][15:0])$	I		0x1	0x03
`lhu`	`rd, imm(rs1)`	$\texttt{rd} \leftarrow \text{zext}(\text{mem}[\texttt{rs1} + \text{sext}(\texttt{imm})][15:0])$	I		0x5	0x03
`lw`	`rd, imm(rs1)`	$\texttt{rd} \leftarrow \text{mem}[\texttt{rs1} + \text{sext}(\texttt{imm})]$	I		0x2	0x03
`sb`	`rs2, imm(rs1)`	$\text{mem}[\texttt{rs1} + \text{sext}(\texttt{imm})] \leftarrow \texttt{rs2}[7:0]$	S		0x0	0x23
`sh`	`rs2, imm(rs1)`	$\text{mem}[\texttt{rs1} + \text{sext}(\texttt{imm})] \leftarrow \texttt{rs2}[15:0]$	S		0x1	0x23
`sw`	`rs2, imm(rs1)`	$\text{mem}[\texttt{rs1} + \text{sext}(\texttt{imm})] \leftarrow \texttt{rs2}$	S		0x2	0x23

$\sim$	Bitwise NOT	$\ll$	Left shift	mem	Memory access
&	Bitwise AND	$\gg$	Right shift	sext	Sign extend
|	Bitwise OR	op_s	Signed *op*eration	zext	Zero extend
^	Bitwise XOR	op_u	Unsigned *op*eration		

Instructions on a dark gray background are seldom used in CS-200

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

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

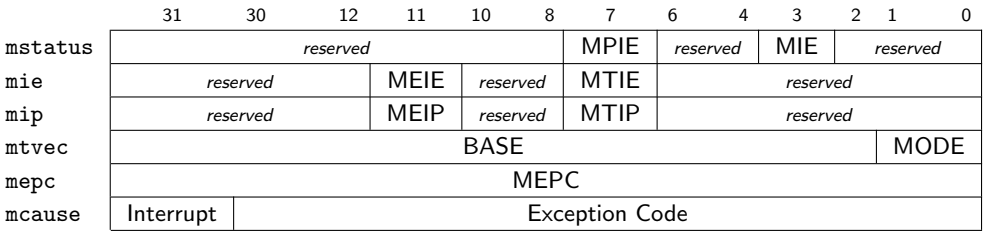
4. Standard Instructions (cont'd)

Instruction or Pseudoinstruction		Pseudocode	Type	funct7	funct3 or Translation	opcode
Branch						
beq	rs1, rs2, imm	pc ← pc + sext(imm ≪ 1), if rs1 = rs2	B		0x0	0x63
bne	rs1, rs2, imm	pc ← pc + sext(imm ≪ 1), if rs1 ≠ rs2	B		0x1	0x63
blt	rs1, rs2, imm	pc ← pc + sext(imm ≪ 1), if rs1 $<_s$ rs2	B		0x4	0x63
bltu	rs1, rs2, imm	pc ← pc + sext(imm ≪ 1), if rs1 $<_u$ rs2	B		0x6	0x63
bge	rs1, rs2, imm	pc ← pc + sext(imm ≪ 1), if rs1 $\geq_s$ rs2	B		0x5	0x63
bgeu	rs1, rs2, imm	pc ← pc + sext(imm ≪ 1), if rs1 $\geq_u$ rs2	B		0x7	0x63
ble	rs1, rs2, imm	pc ← pc + sext(imm ≪ 1), if rs1 $\leq_s$ rs2	bge	rs2, rs1, imm		
bleu	rs1, rs2, imm	pc ← pc + sext(imm ≪ 1), if rs1 $\leq_u$ rs2	bgeu	rs2, rs1, imm		
bgt	rs1, rs2, imm	pc ← pc + sext(imm ≪ 1), if rs1 $>_s$ rs2	blt	rs2, rs1, imm		
bgtu	rs1, rs2, imm	pc ← pc + sext(imm ≪ 1), if rs1 $>_u$ rs2	bltu	rs2, rs1, imm		
beqz	rs1, imm	pc ← pc + sext(imm ≪ 1), if rs1 = 0	beq	rs1, zero, imm		
bnez	rs1, imm	pc ← pc + sext(imm ≪ 1), if rs1 ≠ 0	bne	rs1, zero, imm		
bltz	rs1, imm	pc ← pc + sext(imm ≪ 1), if rs1 $<_s$ 0	blt	rs1, zero, imm		
bgez	rs1, imm	pc ← pc + sext(imm ≪ 1), if rs1 $\geq_s$ 0	bge	rs1, zero, imm		
blez	rs1, imm	pc ← pc + sext(imm ≪ 1), if rs1 $\leq_s$ 0	bge	zero, rs1, imm		
bgtz	rs1, imm	pc ← pc + sext(imm ≪ 1), if rs1 $>_s$ 0	blt	zero, rs1, imm		
Jump						
j	imm	pc ← pc + sext(imm ≪ 1)	jal	zero, imm		
jal	imm	ra ← pc + 4 pc ← pc + sext(imm ≪ 1)	jal	ra, imm		
ret		pc ← ra & ∼1	jalr	zero, 0(ra)		
jal	rd, imm	rd ← pc + 4 pc ← pc + sext(imm ≪ 1)	J			0x6F
jalr	rd, imm(rs1)	rd ← pc + 4 pc ← (rs1 + sext(imm)) & (∼1)	I		0x0	0x67
jr	rs1	pc ← rs1 & ∼1	jalr	zero, 0(ra)		
jalr	rs1	ra ← pc + 4 pc ← rs1 & ∼1	jalr	ra, 0(rs1)		

5. M-mode Instructions

Instruction or Pseudoinstruction		Pseudocode or Translation	Meaning	Type	funct3
csrr	rd, csr	csrrs rd, csr, zero	Read CSR		
csrw	csr, rs	csrrw zero, csr, rs	Write CSR		
csrs	csr, rs	csrrs zero, csr, rs	Set bits in CSR		
csrc	csr, rs	csrrc zero, csr, rs	Clear bits in CSR		
csrwi	csr, imm	csrrwi zero, csr, imm	Write CSR, immediate		
csrsi	csr, imm	csrrsi zero, csr, imm	Set bits in CSR, immediate		
csrci	csr, imm	csrrci zero, csr, imm	Clear bits in CSR, immediate		
csrrw	rd, csr, rs1	rd ← csr; csr ← rs1	Read/write CSR	CSR	0x1
csrrs	rd, csr, rs1	rd ← csr; csr ← csr | rs1	Read/set bits CSR	CSR	0x2
csrrc	rd, csr, rs1	rd ← csr; csr ← csr & (∼rs1)	Read/clear bits CSR	CSR	0x3
csrrwi	rd, csr, uimm	rd ← csr; csr ← zext(uimm)	Read/write CSR, immediate	CSR-I	0x5
csrrsi	rd, csr, uimm	rd ← csr; csr ← csr | zext(uimm)	Read/set bits in CSR, immediate	CSR-I	0x6
csrrci	rd, csr, uimm	rd ← csr; csr ← csr & (∼zext(uimm))	Read/clear bits in CSR, immediate	CSR-I	0x7
mret		mstatus.MIE ← mstatus.MPIE mstatus.MPIE ← 1 pc ← mepc	Return from M-mode	MRET	

6. Control and Status Registers (CSRs)



Register	Name	Description
`mstatus`	Machine Status	MIE (Interrupt enable) MPIE (Previous MIE value)
`mie`	Machine Interrupt-Enable	MEIE (Machine-level external interrupt enable) MTIE (Machine timer interrupt enable)
`mip`	Machine Interrupt-Pending	MEIP (Machine-level external interrupt pending) MTIP (Machine timer interrupt pending)
`mtvec`	Machine Trap-Vector Base-Address	Holds trap vector base address
`mepc`	Machine Exception Program Counter	Holds address of the instruction that caused the exception
`mcause`	Machine Cause	Interrupt (set if the trap was caused by an interrupt) Exception Code (identifies the event)

7. Common Machine Cause Register (`mcause`) Values

Interrupt	Exception Code	Description
1	3	Software interrupt
1	7	Timer interrupt
1	11	External interrupt
0	2	Illegal instruction
0	3	Breakpoint
0	12	Instruction page fault
0	13	Load page fault
0	15	Store page fault

8. M-mode Instruction Types and Encodings