See second page for shifts, branches, jumps.

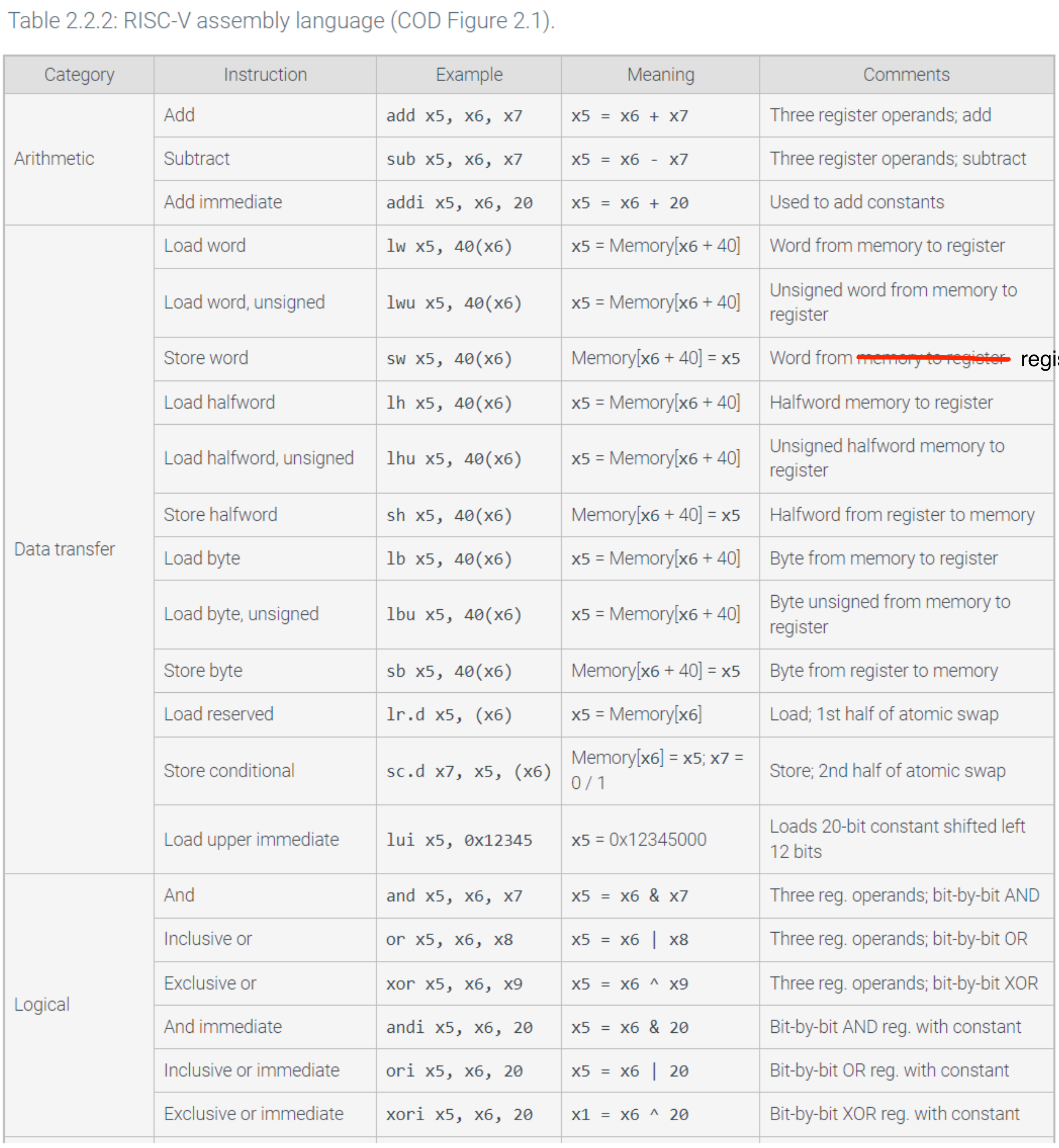
Table 2.2.2: RISC-V assembly language (COD Figure 2.1).

Category	Instruction	Example	Meaning	Comments
Arithmetic	Add	`add x5, x6, x7`	`x5 = x6 + x7`	Three register operands; add
	Subtract	`sub x5, x6, x7`	`x5 = x6 - x7`	Three register operands; subtract
	Add immediate	`addi x5, x6, 20`	`x5 = x6 + 20`	Used to add constants
Data transfer	Load word	`lw x5, 40(x6)`	x5 = Memory[x6 + 40]	Word from memory to register
	Load word, unsigned	`lwu x5, 40(x6)`	x5 = Memory[x6 + 40]	Unsigned word from memory to register
	Store word	`sw x5, 40(x6)`	Memory[x6 + 40] = x5	Word from ~~memory to register~~ register to memory
	Load halfword	`lh x5, 40(x6)`	x5 = Memory[x6 + 40]	Halfword memory to register
	Load halfword, unsigned	`lhu x5, 40(x6)`	x5 = Memory[x6 + 40]	Unsigned halfword memory to register
	Store halfword	`sh x5, 40(x6)`	Memory[x6 + 40] = x5	Halfword from register to memory
	Load byte	`lb x5, 40(x6)`	x5 = Memory[x6 + 40]	Byte from memory to register
	Load byte, unsigned	`lbu x5, 40(x6)`	x5 = Memory[x6 + 40]	Byte unsigned from memory to register
	Store byte	`sb x5, 40(x6)`	Memory[x6 + 40] = x5	Byte from register to memory
	Load reserved	`lr.d x5, (x6)`	x5 = Memory[x6]	Load; 1st half of atomic swap
	Store conditional	`sc.d x7, x5, (x6)`	Memory[x6] = x5; x7 = 0 / 1	Store; 2nd half of atomic swap
	Load upper immediate	`lui x5, 0x12345`	x5 = 0x12345000	Loads 20-bit constant shifted left 12 bits
Logical	And	`and x5, x6, x7`	`x5 = x6 & x7`	Three reg. operands; bit-by-bit AND
	Inclusive or	`or x5, x6, x8`	`x5 = x6 | x8`	Three reg. operands; bit-by-bit OR
	Exclusive or	`xor x5, x6, x9`	`x5 = x6 ^ x9`	Three reg. operands; bit-by-bit XOR
	And immediate	`andi x5, x6, 20`	`x5 = x6 & 20`	Bit-by-bit AND reg. with constant
	Inclusive or immediate	`ori x5, x6, 20`	`x5 = x6 | 20`	Bit-by-bit OR reg. with constant
	Exclusive or immediate	`xori x5, x6, 20`	`x1 = x6 ^ 20`	Bit-by-bit XOR reg. with constant

Shift	Shift left logical	`sll x5, x6, x7`	`x5 = x6 << x7`	Shift left by register
	Shift right logical	`srl x5, x6, x7`	`x5 = x6 >> x7`	Shift right by register
	Shift right arithmetic	`sra x5, x6, x7`	`x5 = x6 >> x7`	Arithmetic Shift right by register
	Shift left logical immediate	`slli x5, x6, 3`	`x5 = x6 << 3`	Shift left by immediate
	Shift right logical immediate	`srli x5, x6, 3`	`x5 = x6 >> 3`	Shift right by immediate
	Shift right logical arithmetic immediate	`srai x5, x6, 3`	`x5 = x6 >> 3`	Arithmetic shift right by immediate
Conditional branch	Branch if equal	`beq x5, x6, 100`	if (`x5 == x6`) go to PC + 100	PC-relative branch if registers equal
	Branch if not equal	`bne x5, x6, 100`	if (`x5 != x6`) go to PC + 100	PC-relative branch if registers not equal
	Branch if less than	`blt x5, x6, 100`	if (`x5 < x6`) go to PC + 100	PC-relative branch if registers less
	Branch if greater or equal	`bge x5, x6, 100`	if (`x5 >= x6`) go to PC + 100	PC-relative branch if registers greater or equal
	Branch if less, unsigned	`bltu x5, x6, 100`	if (`x5 < x6`) go to PC + 100	PC-relative branch if registers less, unsigned
	Branch if greater or equal, unsigned	`bgeu x5, x6, 100`	if (`x5 >= x6`) go to PC + 100	PC-relative branch if registers greater or equal, unsigned
Unconditional jump	Jump and link	`jal x1, 100`	x1 = PC+4; go to PC + 100	PC-relative procedure call
	Jump and link register	`jalr x1, 100(x5)`	x1 = PC+4; go to x5 + 100	Procedure return; indirect call