

Exercise walk-through

3.4

Machine instructions

University Computer Science

Machine instructions: registers, immediates, memory

- 1 **Shapes** add rd, rs1, rs2 (registers); addi (immediate); lw/sw with base+offset; beq/bne compare and branch.
- 2 **Loop** addi R1, R1, -1; bne R1, R0, loop — two instructions per iteration, exactly the C while pair.
- 3 **Indexing** $a[i] = a[i] + 1 \rightarrow \text{lw} \cdot \text{addi} \cdot \text{sw}$: the index scales into the offset; a pointer is a base register.

Instruction and register trace

a loop, unrolled into machine steps

instruction	action	R1	mem
addi R1, R0, 3	$R1 \leftarrow 0 + 3$	3	—
loop: addi R1, R1, -1	$R1 \leftarrow R1 - 1$	2	—
bne R1, R0, loop	$R1 \neq 0 \rightarrow \text{jump}$	2	—
addi R1, R1, -1	$R1 \leftarrow R1 - 1$	1	—
bne R1, R0, loop	$R1 \neq 0 \rightarrow \text{jump}$	1	—
addi R1, R1, -1	$R1 \leftarrow R1 - 1$	0	—
bne R1, R0, loop	$R1 = 0 \rightarrow \text{fall through}$	0	—

two instructions per iteration — the C while loop is exactly this pair + body

$a[i] = a[i] + 1 \rightarrow \text{lw} \cdot \text{addi} \cdot \text{sw}$ (index scaled into the offset)

addressing
register
 add R3, R1, R2
immediate
 addi R1, R1, 1
base+offset
 lw R1, 0(R2)
 memory: lw / sw only

Each line: fetch, execute, the register delta — a loop unrolled.

Exercises 2.1-2.3

2.1 In `lw R1, 8(R2)`, what is the 8? [3]

2.2 Which instruction family touches memory? [3]

2.3 Write the pair that counts R1 down and loops while nonzero. [3]

Exercises 2.1-2.3

2.1 In `lw R1, 8(R2)`, what is the 8? [3]

2.1

- The offset added to base R2 $\rightarrow$ address $R2 + 8$ ✓.

2.2 Which instruction family touches memory? [3]

2.2

- `lw` / `sw` (load/store) only ✓.

2.3 Write the pair that counts R1 down and loops while nonzero. [3]

2.3

- `addi R1, R1, -1` then `bne R1, R0, loop` ✓.

Exercise 3.1

3.1 Assemble: sum the five integers at mem[200..216] into R5.

Write the instruction sequence, naming any labels you use. State the register that serves as the loop counter and the one as the running sum. Give the iteration count and the final offset on R2. [8]

Exercise 3.1

3.1 Assemble: sum the five integers at mem[200..216] into R5.

Write the instruction sequence, naming any labels you use. State the register that serves as the loop counter and the one as the running sum. Give the iteration count and the final offset on R2. [8]

3.1

- Init: `addi R2, R0, 200; add R5, R0, R0; addi R4, R0, 5` ✓.
- Body: `lw R3, 0(R2); add R5, R5, R3; addi R2, R2, 4; addi R4, R4, -1; bne R4, R0, loop` ✓.
- Counter R4, sum R5; 5 iterations; R2 ends at $200 + 20$ ✓.

Exercise 3.2

3.2 Map C constructs to instruction shapes.

For each of `while (n > 0), a[i] = a[i] * 2`, and `if (x == y) goto L`, give the instructions it compiles to. Name the addressing mode of every operand. Explain in one line why keeping arithmetic register-only helps pipelining. [7]

Exercise 3.2

3.2 Map C constructs to instruction shapes.

For each of `while (n > 0), a[i] = a[i] * 2`, and `if (x == y) goto L`, give the instructions it compiles to. Name the addressing mode of every operand. Explain in one line why keeping arithmetic register-only helps pipelining. [7]

3.2

- `while` $\rightarrow$ `addi R1, R1, -1 + bne (register + immediate)` ✓.
- `a[i]*2` $\rightarrow$ `lw R3, 0(R2), add R3, R3, R3, sw R3, 0(R2)` (base+offset, register, base+offset) ✓.
- `if-equal` $\rightarrow$ `beq R1, R2, L` (register, register) ✓.
- Register-only arithmetic: one decode shape, no memory wait in the execute stage – pipelines stay full ✓.