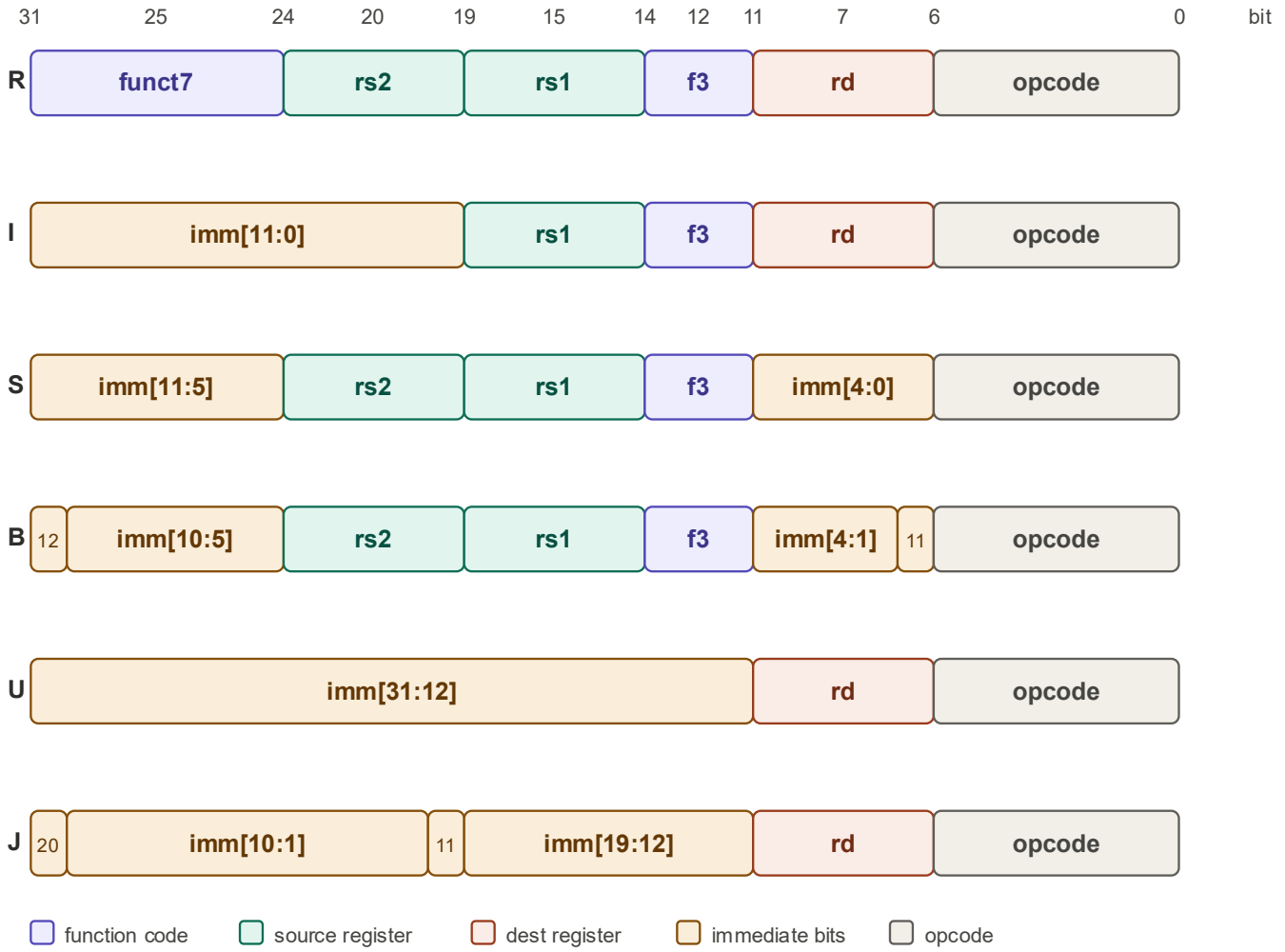
The RV32I Single-Cycle Processor

CMPU 224 – Computer Organization

Jason Waterman

RISC-V Instruction Formats

- Six types of instruction formats
  - Opcode bits specify which type
- Needed because all instructions are 32 bits long
- Opcode, rs1, rs2, rd, and funct3 are always located in the same bit positions
- Immediate bits are scrambled among the different instruction types
  - Needed to keep rs1, rs2, and rd at a fixed location across instructions
  - Bit-31 is always the sign bit of the immediate value

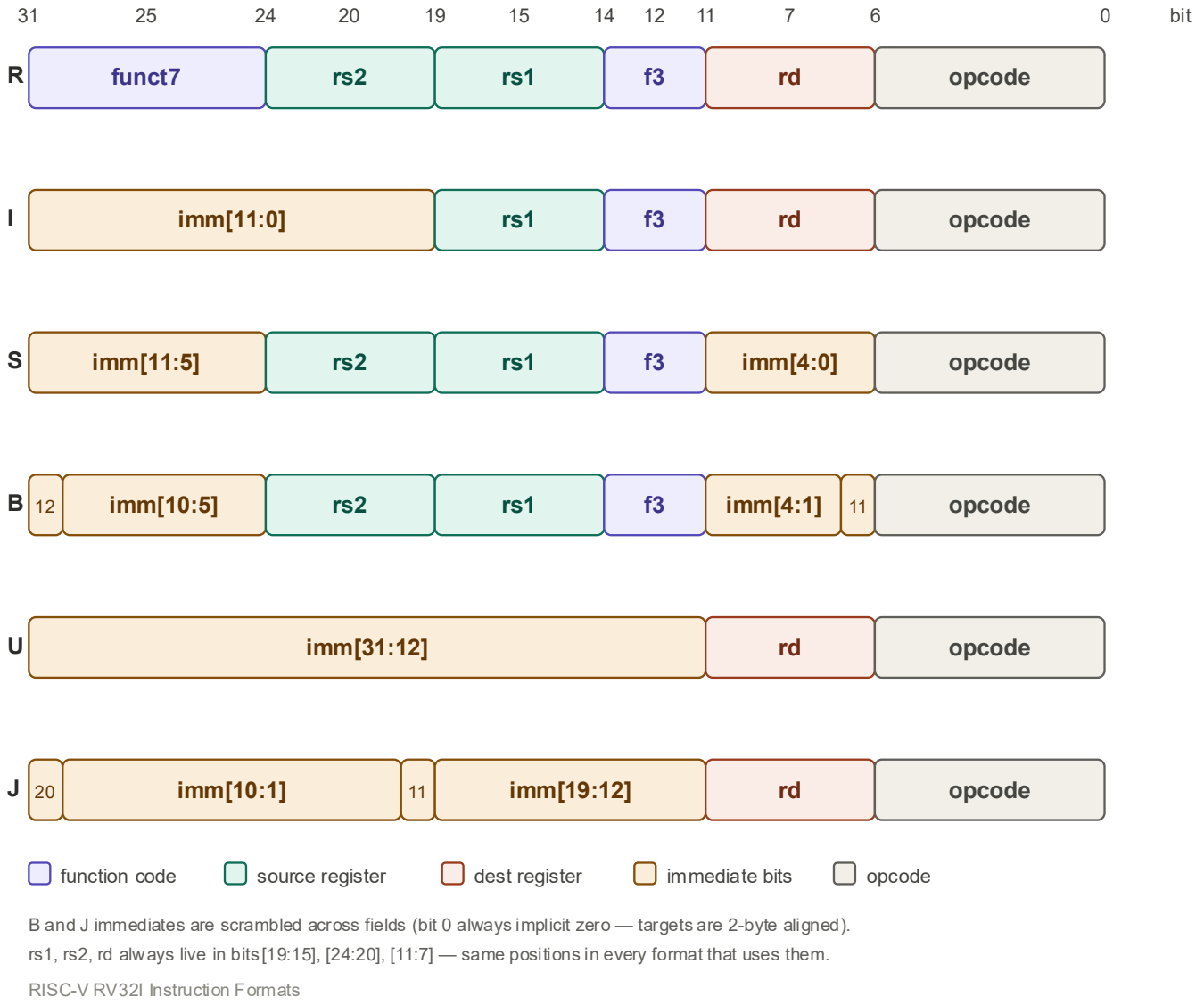


B and J immediates are scrambled across fields (bit 0 always implicit zero — targets are 2-byte aligned).
rs1, rs2, rd always live in bits[19:15], [24:20], [11:7] — same positions in every format that uses them.

RISC-V RV32I Instruction Formats

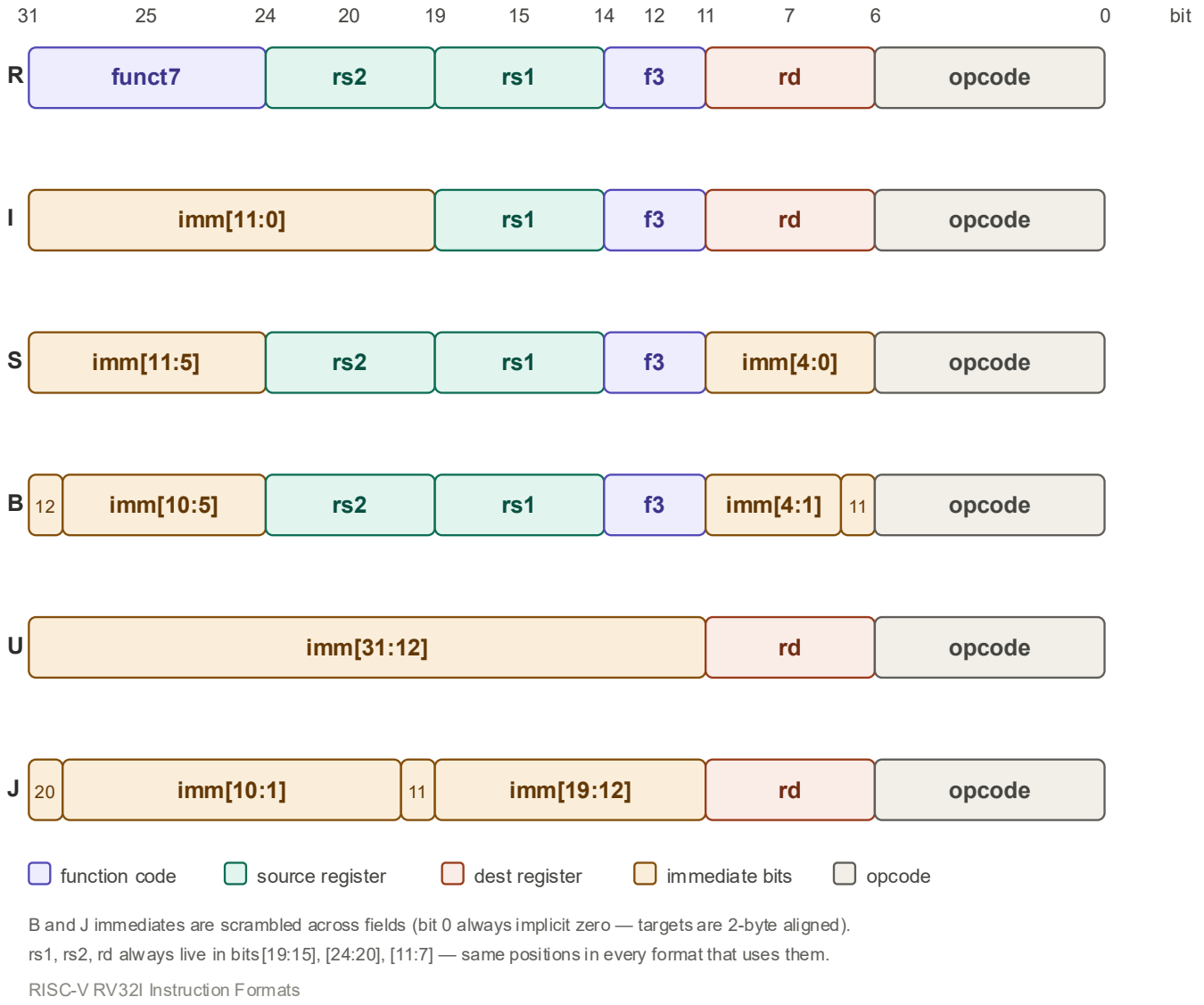
RISC-V Instructions: R-type (Register-Register)

- Both operands come from registers (rs1 and rs2)
- Result goes to a register (rd)
- The funct7 and funct3 fields together select the operations
- Used by arithmetic and logic instructions
  - Examples: add, sub, and, or, slt, sll, srl, sra



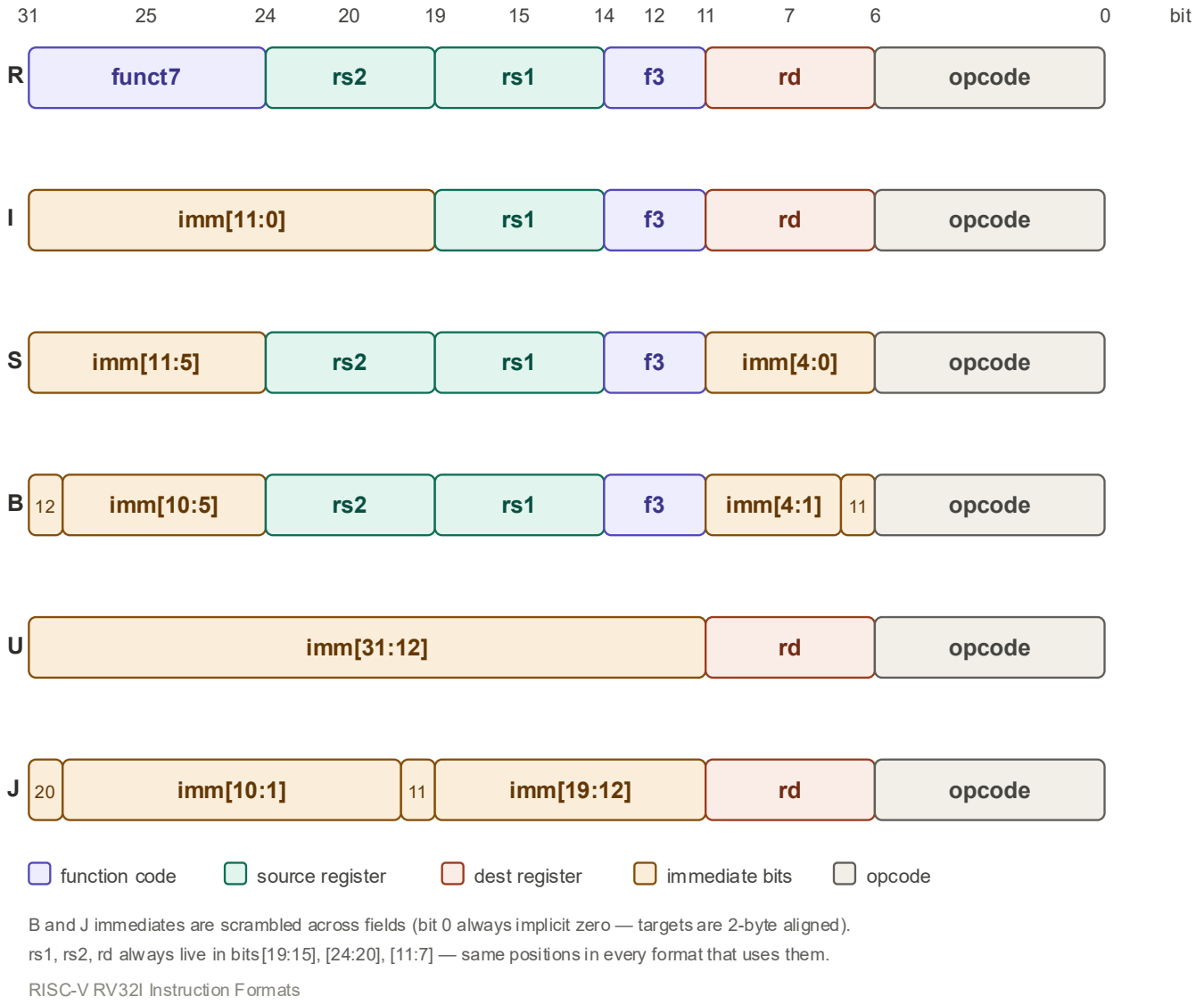
RISC-V Instructions: I-type (Immediate)

- Used in instructions with one register operand and an immediate
- The immediate is a 12-bit two's complement number sign-extended to 32 bits before use
- Examples:
  - loads (lw, lb) – the memory address of the load is calculated by rs1 + imm
  - Arithmetic with a constant (addi, andi, ori)
  - Jump and link register (jalr) – jump address is rs1 + imm



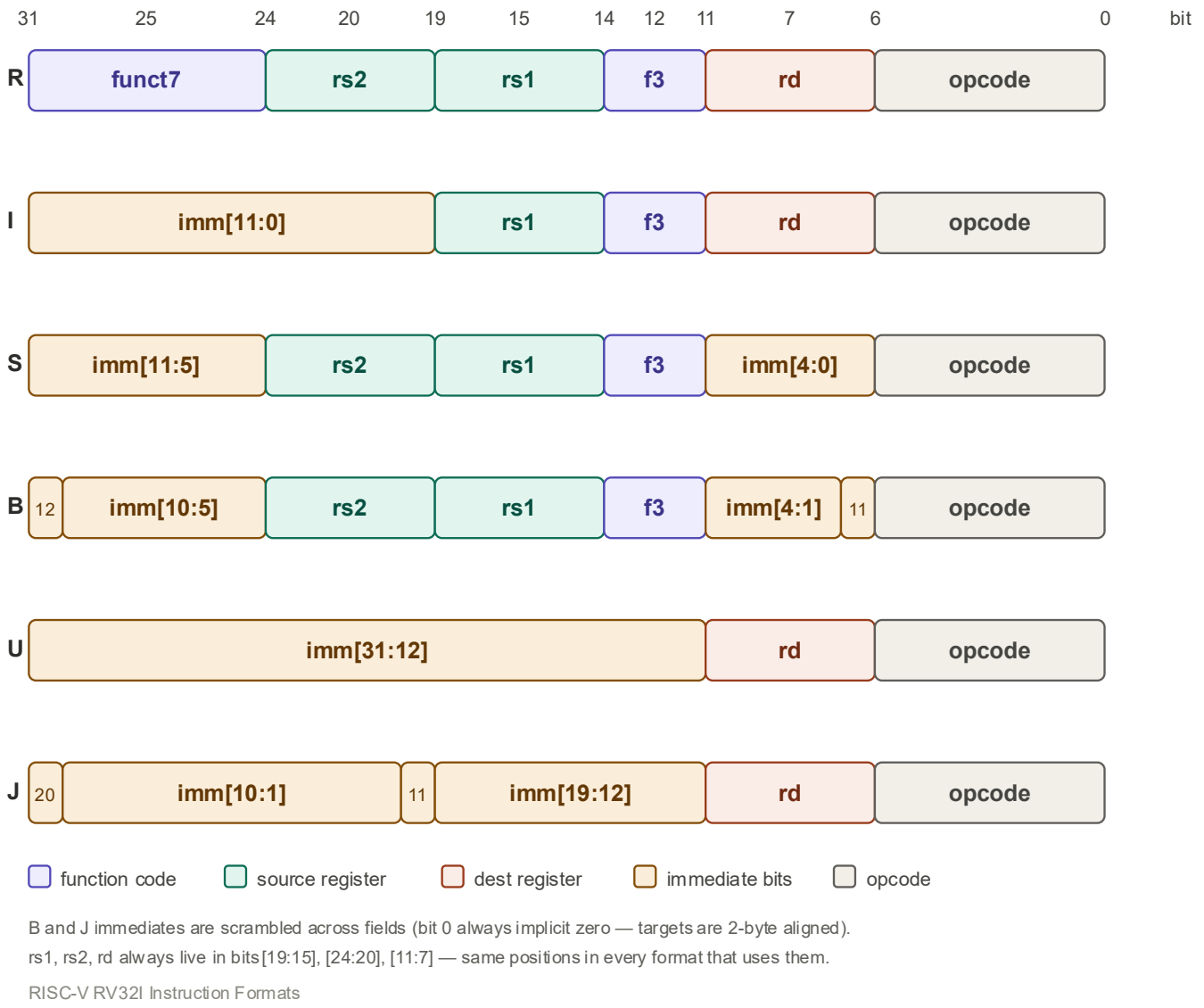
RISC-V Instructions: S-type (Store)

- Stores a register value to memory
- The immediate is split across two fields
  - Keeps rs1 and rs2 in the same position
- Examples:
  - sw, sb



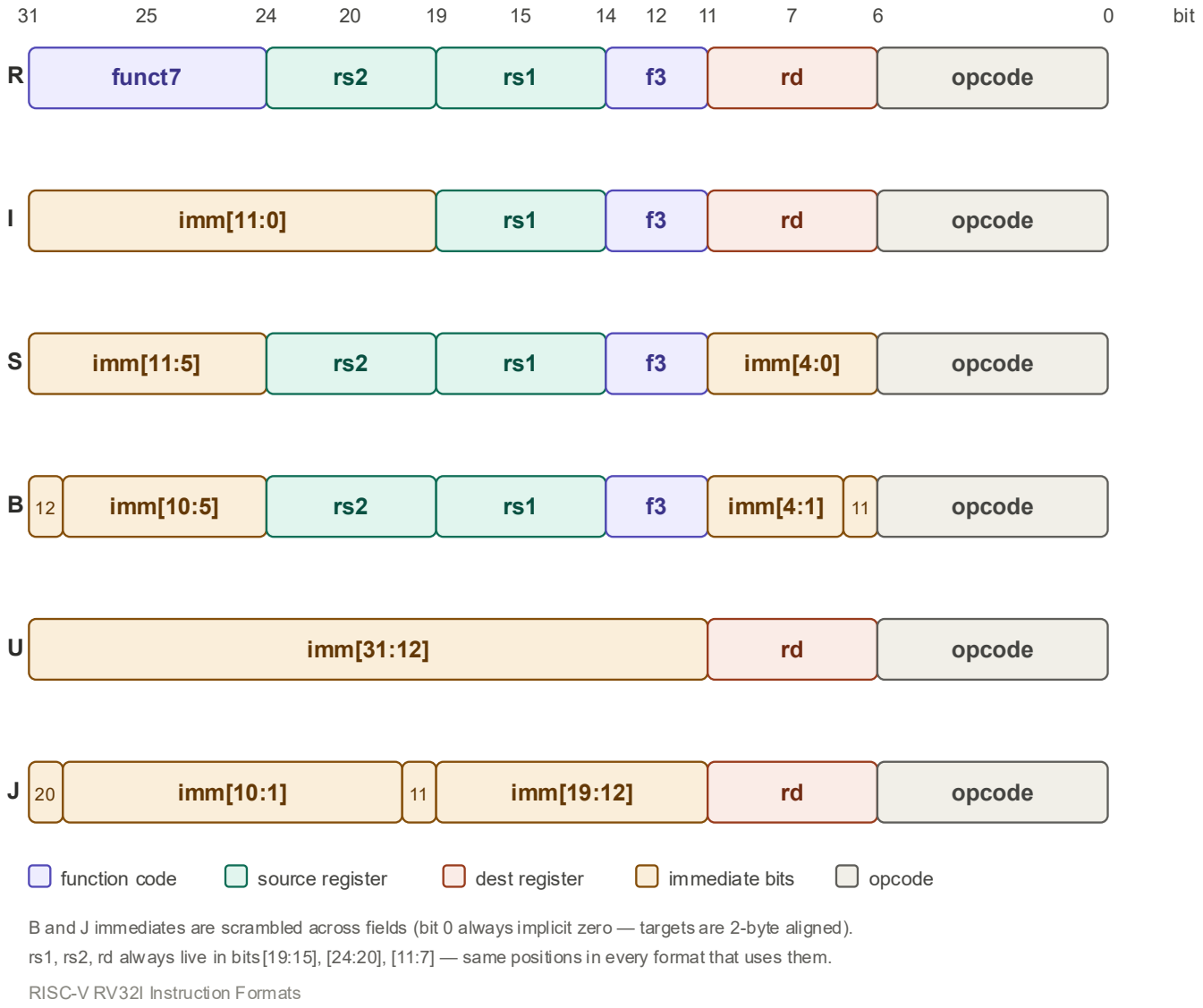
RISC-V Instructions: B-type (Branch)

- Conditional branches
- 12-bit signed offset represents a 13-bit offset from the PC
  - Known as PC-relative addressing
  - Has an implied lsb of 0
  - All RISCV instructions must start on an address that is multiple of 2
  - Range: +/- 4KiB
- bit-31 is the sign bit
- Other immediate bits are scrambled to overlap with S-type instructions
- Examples:
  - beq, bne, blt, bge



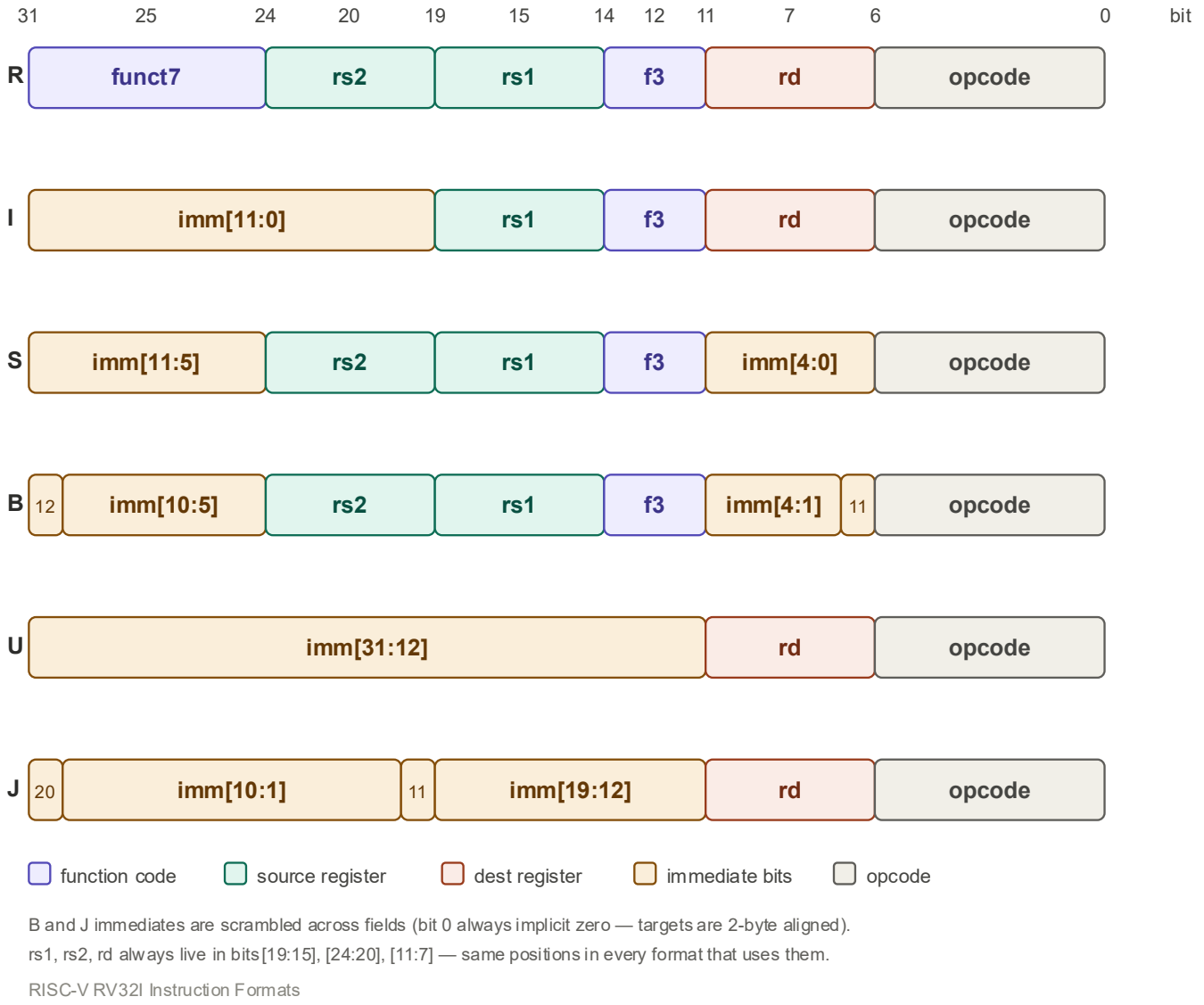
RISC-V Instructions: U-type (Upper Immediate)

- Loads a 20-bit constant into the upper 20 bits of a register
- Typically paired with an I-type instruction (addi) to build a 32-bit constant or 32-bit address
- Instructions:
  - lui – load upper immediate
  - auipc – add upper immediate to PC



RISC-V Instructions: J-type (Jump)

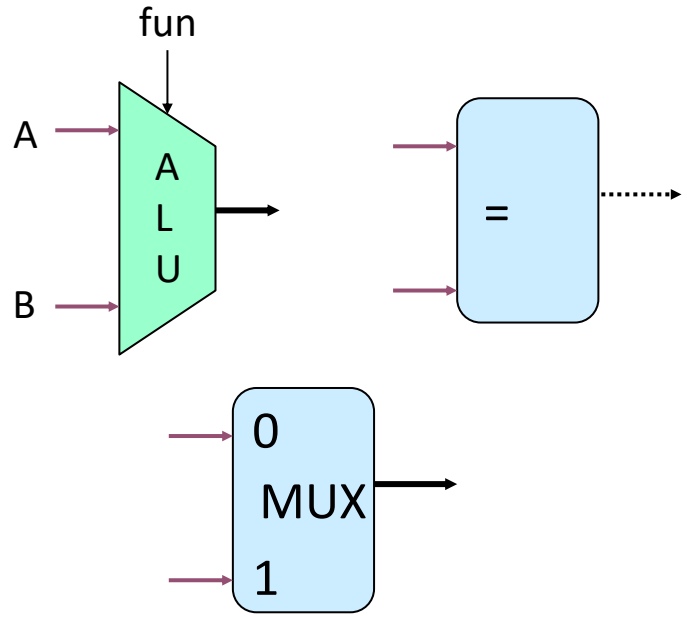
- Used only be jal (jump and link)
- Encodes a 21-bit signed offset
  - Also has implicit zero in bit zero
- Range: +/- 1 MiB
- Scrambled imm value to overlap with B-type instructions
- rd receives the return address (PC + 4)



B and J immediates are scrambled across fields (bit 0 always implicit zero — targets are 2-byte aligned).
rs1, rs2, rd always live in bits[19:15], [24:20], [11:7] — same positions in every format that uses them.

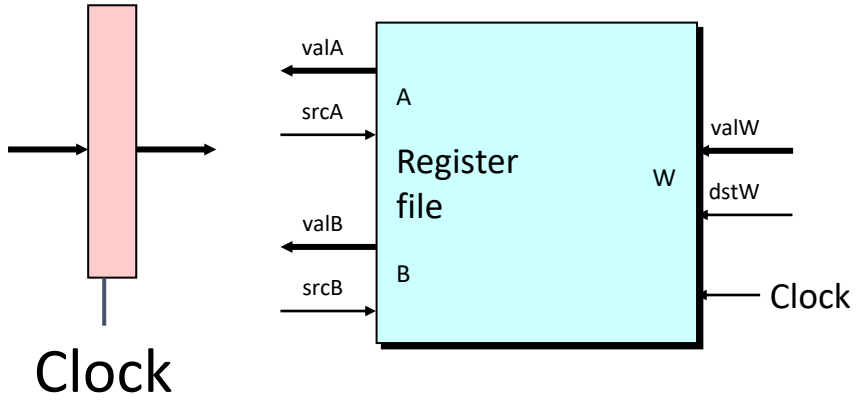
RISC-V RV32I Instruction Formats

Building Blocks



- Combinational Logic
  - Compute Boolean functions of inputs
  - Continuously respond to input changes
  - Operate on data and implement control

- Storage Elements
  - Store bits
  - Registers
  - Addressable memories
  - Loaded only as clock rises



Processor State

- There are four categories of state that are updated each clock cycle

1. Program Counter (PC)
   - 32-bit register holding the address of the current instruction
   - Updated every cycle to the next instruction address
2. Register File
   - 32 registers x 32 bits
   - Two read ports (rs1, rs2) and one write port (rd)
   - Register x0 is hardwired to zero
   - Read is combinational; write happens on the rising edge of the clock
3. Instruction Memory
   - Addressed by PC
   - Outputs the 32-bit instruction
   - Considered read-only memory
4. Data Memory
   - Used by load and store instructions
   - One address input, one write-data input, one read-data output

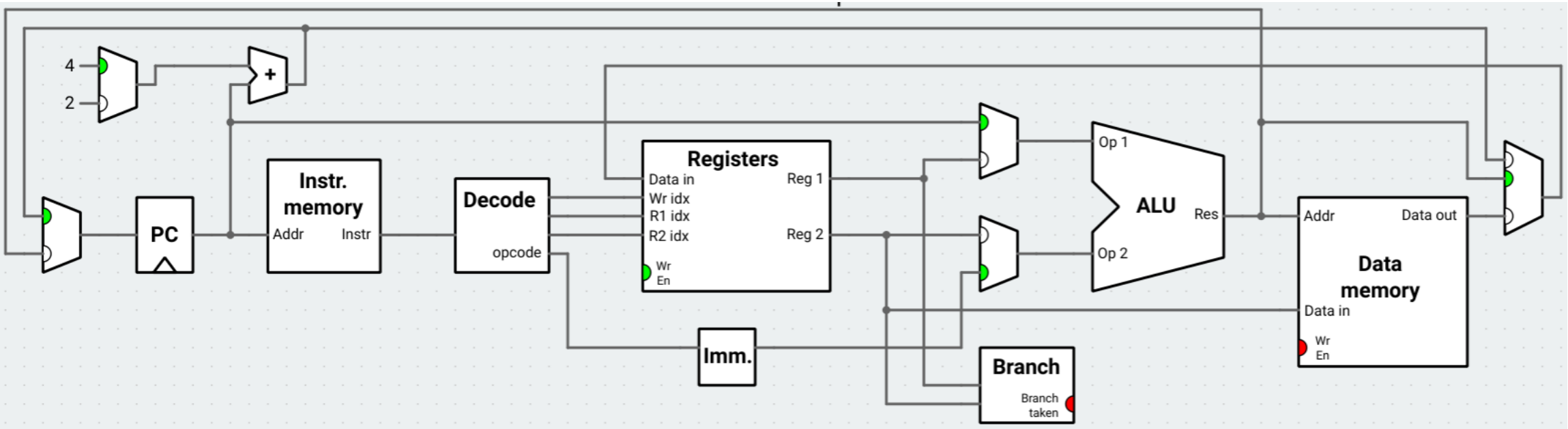
Single-Cycle Datapath Overview

- Even though the single-cycle processor executes an instruction in one clock cycle, we conceptually divide execution into five stages

1. Fetch (IF)
   - Read instruction from instruction memory at address PC
   - Compute PC+4 (next instruction)
2. Decode (ID)
   - Read source registers rs1 and rs2
   - Generate immediate
   - Decode control signals
3. Execute (EX)
   - Alu performs operation (arithmetic, address calculation, or comparison)
4. Memory (MEM)
   - Read from or write to memory (loads and stores only)
5. Writeback (WB)
   - Write result back to the destination register rd

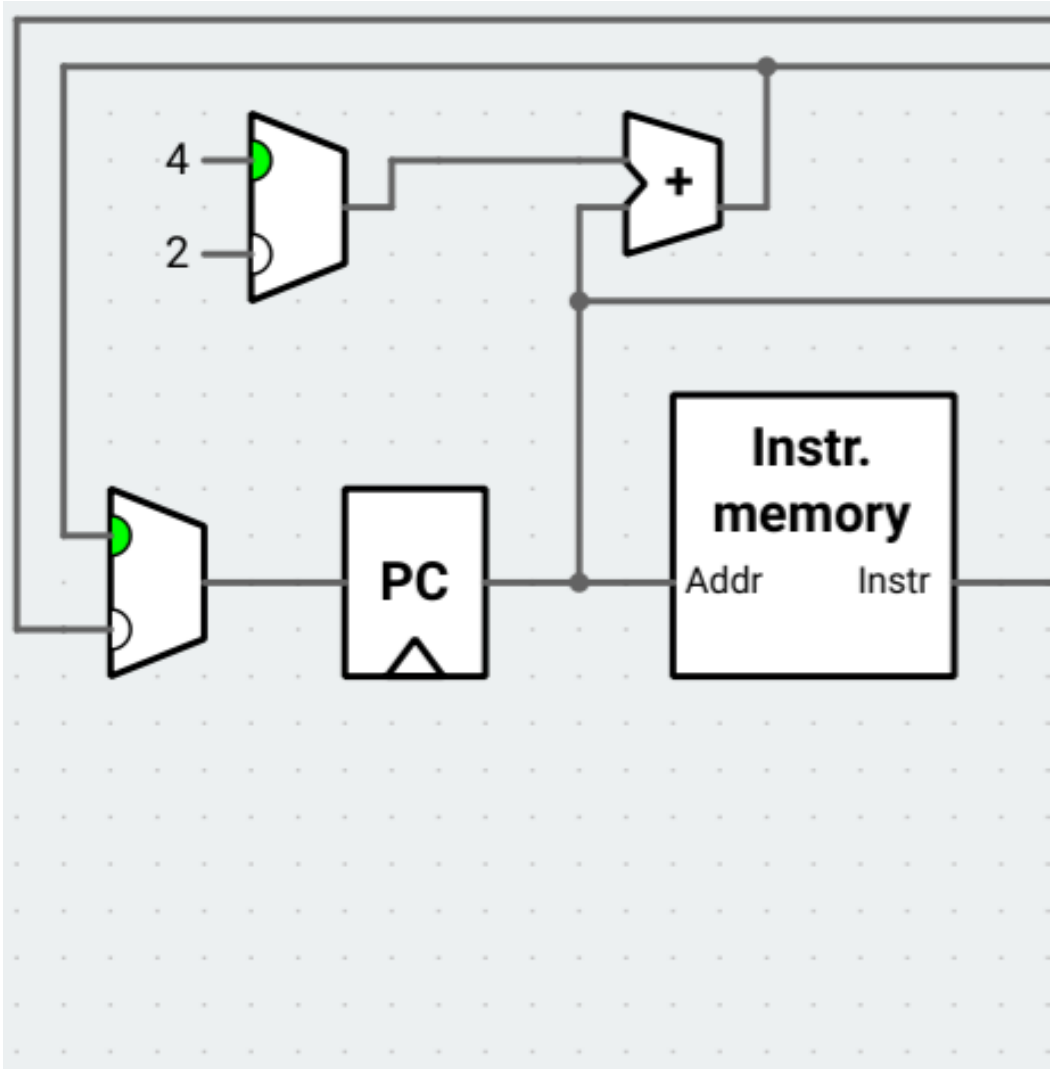
Datapath Diagram

- Data flows from left to write through each of the stages
- Control signals (not shown) from the Decode stage to every other stage, telling the muxes and functional units what do



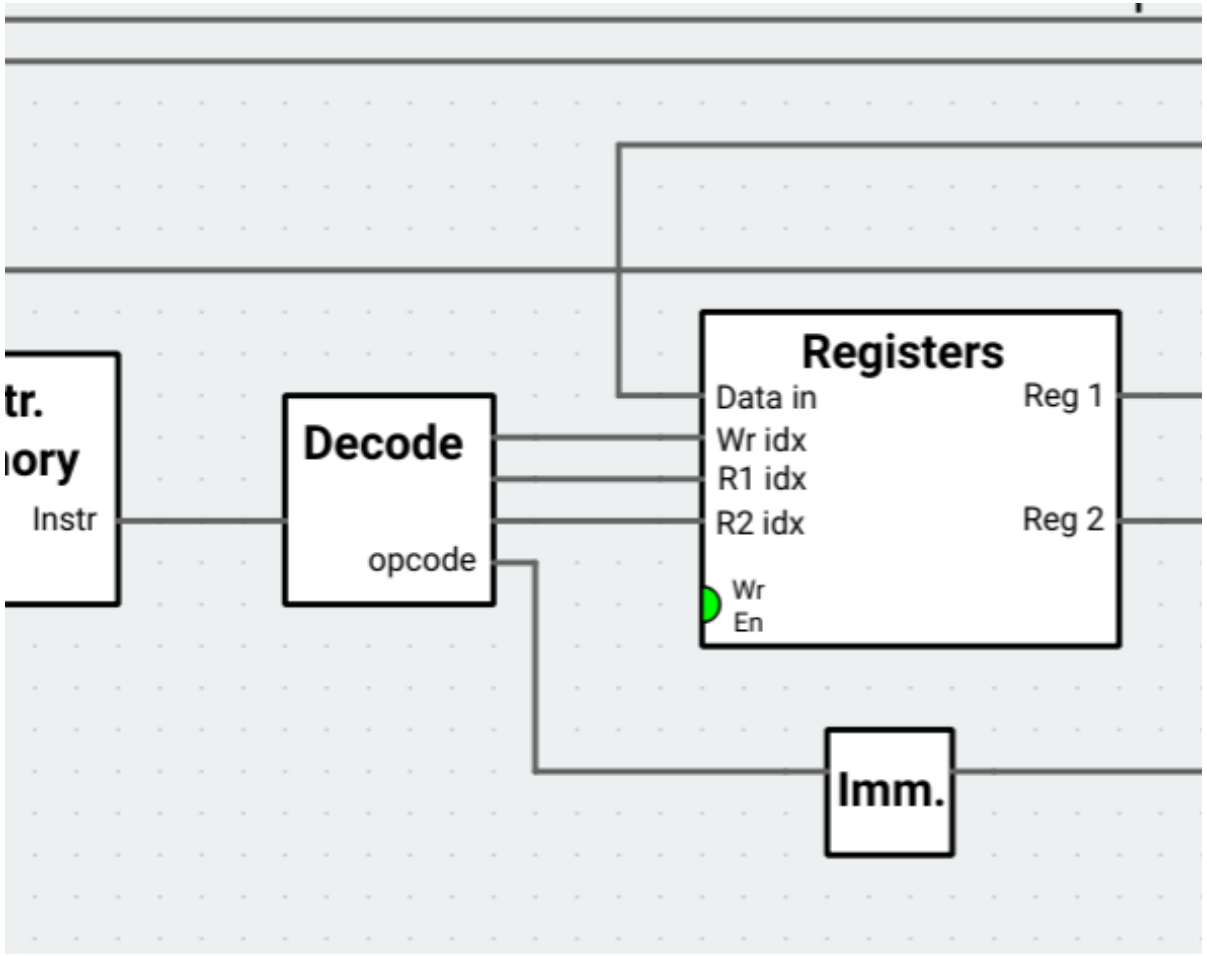
Fetch Stage (IF)

- Read the current instruction and computer the address of the next one
- The PC supplies the address to Instruction Memory
- Instruction Memory outputs the 32-bit instruction word
- Mux selects the next PC value
  - PC+4 for sequential execution
  - Branch target (PC+imm) if branch is taken (from ALU)
  - Jump target for jal / jalr (from ALU)



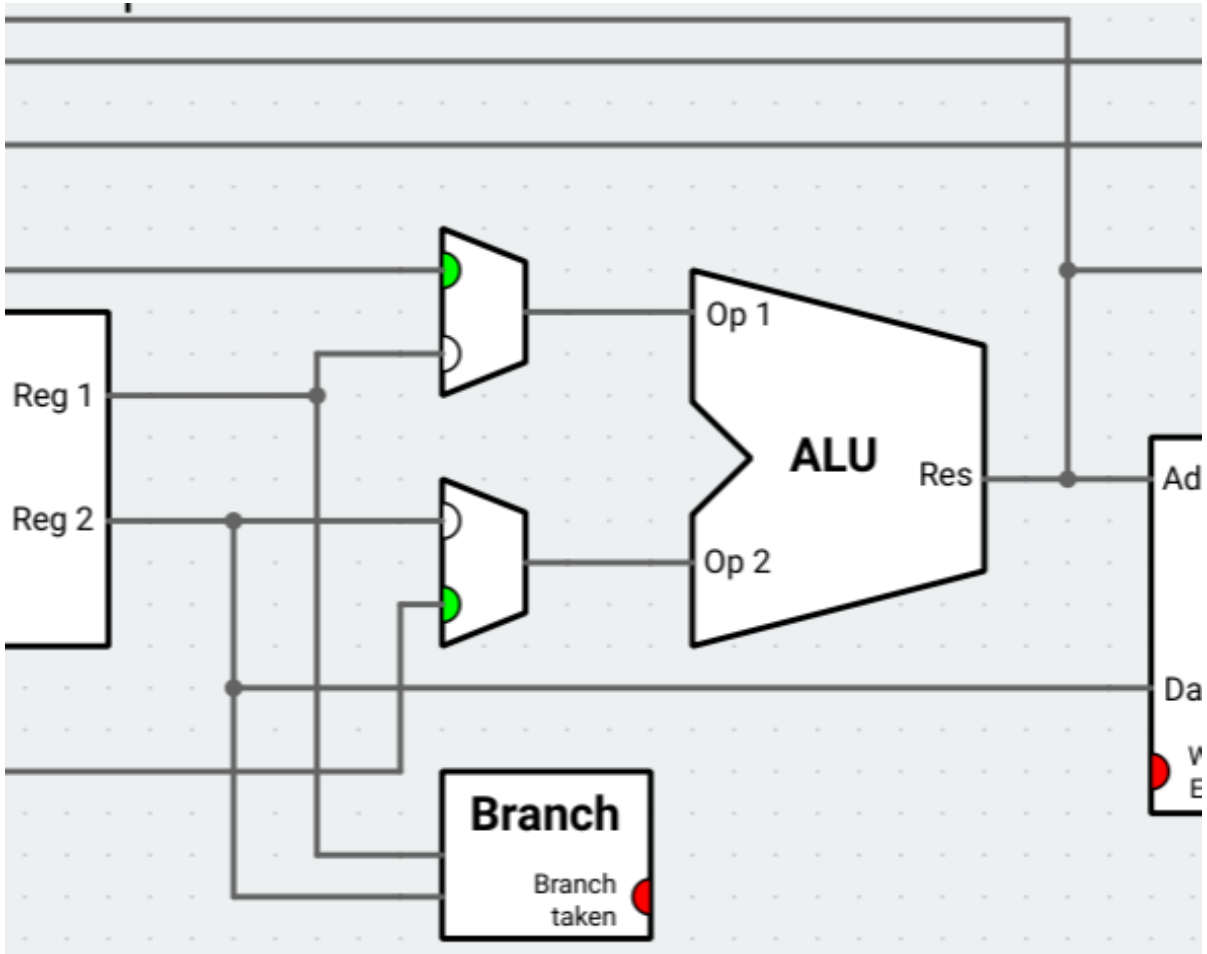
Decode Stage (ID)

- Extract register specifiers, generate the immediate value, read the register file, and produce control signals (not shown)
- The **Immediate Generator** extracts and sign-extends the immediate based on instruction format (I, S, B, U, J)
- The **Register File** reads values at addresses rs1 and rs2, producing ReadData1 and ReadData2



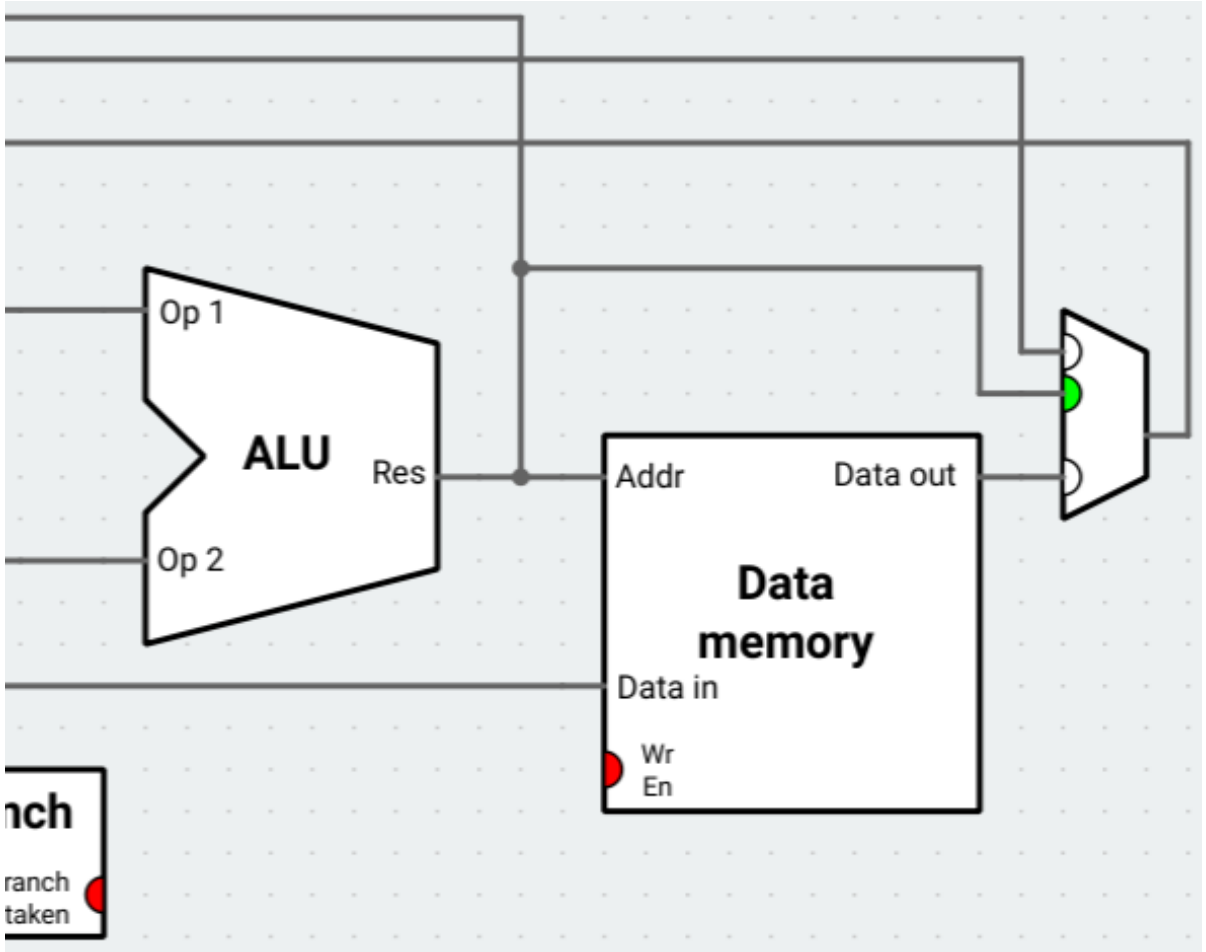
Execute Stage (EX)

- Perform the computation
  - Arithmetic, logic, address, or branch comparison
- The **ALU** receives two inputs:
  - **Op1:** ReadData1 (value of rs1) or PC
  - **Op2:** ReadData2 (value or rs2) or the sign-extended **immediate**
- The **Branch** logic is a separate adder for comparison
  - Generates a Boolean whether the branch should be taken for comparison instructions



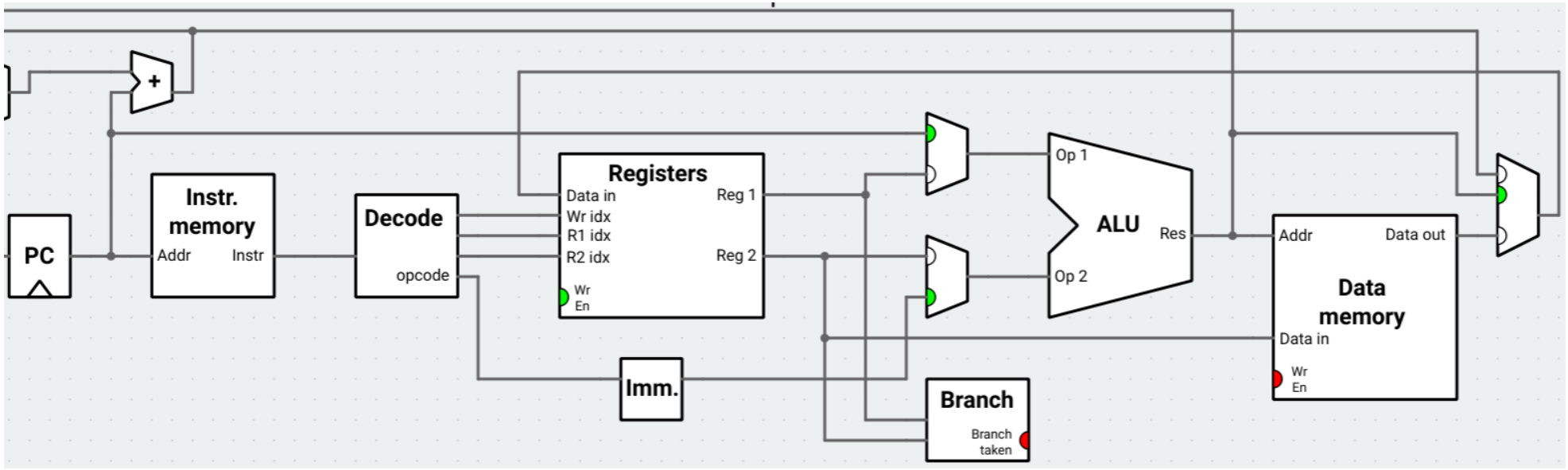
Memory Stage (MEM)

- Access data memory for load and store instructions
  - Other instructions pass through
- The ALU provides computed memory address (rs1+imm)
- For stores (sw, sh, sb): the value of rs2 is written to memory at the address
- For loads (lw, lh, lb): the data is read from the address



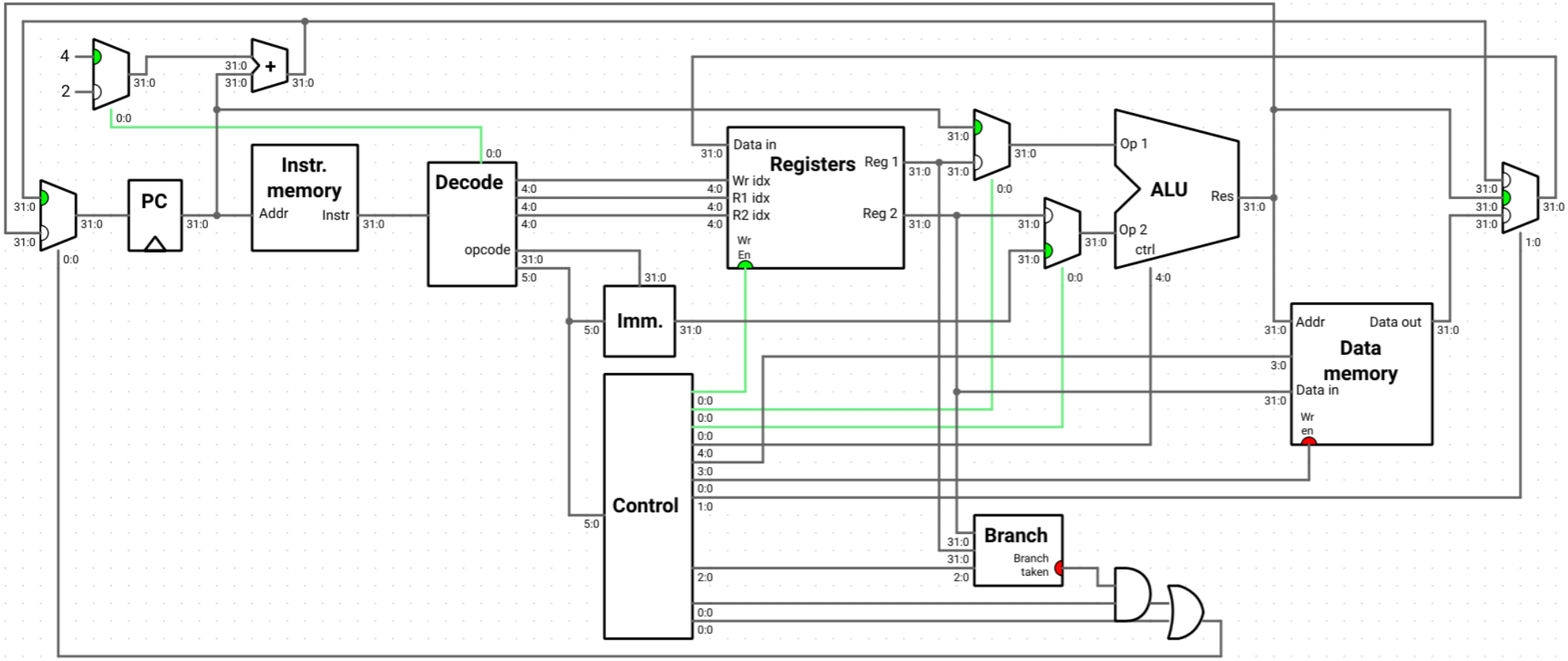
Write-Back Stage (WB)

- Write the result of the instruction back to the register file
  - ALU Result – for R-type and I-type arithmetic/logic
  - Memory Read Data – for load instructions
  - PC+4 – for jal and jalr (to save the return address)



Control and Datapath

- The Control Unit takes the opcode as primary input and produces all the control signals that steer the datapath



Stage Computation: R-type (add rd, rs1, rs2)

Stage	Action	Signals / Values
Fetch	Read instruction at PC Compute next address	inst ← IMem[PC] PC + 4
Decode	Read registers rs1, rs2	ReadData1 ← R[rs1] ReadData2 ← R[rs2]
Execute	ALU performs operation **OP** on the two register values	ALUResult ← ReadData1 OP ReadData2
Memory	Nothing	Pass through
Write Back	Write ALU result to rd	R[rd] ← ALUReslt

Stage Computation: I-type (addi rd, rs1, imm)

Stage	Action	Signals / Values
Fetch	Read instruction at PC Compute next address	inst ← IMem[PC] PC + 4
Decode	Read registers rs1 Generate immediate	ReadData1 ← R[rs1] imm ← SignExt(inst[31:20])
Execute	ALU performs operation OP on the register and immediate	ALUResult ← ReadData1 OP imm
Memory	Nothing	Pass through
Write Back	Write ALU result to rd	R[rd] ← ALUReslt

Stage Computation: Load (lw rd, imm(rs1))

Stage	Action	Signals / Values
Fetch	Read instruction at PC Compute next address	inst ← IMem[PC] PC + 4
Decode	Read base register rs1 Generate immediate	ReadData1 ← R[rs1] imm ← SignExt(inst[31:20])
Execute	Compute effective address	ALUResult ← ReadData1 + imm
Memory	Read data memory at address	MemData ← Dmem[ALUResult]
Write Back	Write ALU result to rd	R[rd] ← MemData

- The load instruction exercises **all five stages** meaningfully
- This is the critical-path instruction that determines the minimum clock period in a single-cycle design

Stage Computation: Store (sw rs2, imm(rs1))

Stage	Action	Signals / Values
Fetch	Read instruction at PC Compute next address	inst ← IMem[PC] PC + 4
Decode	Read base register rs1 Read data register rs2 Generate immediate	ReadData1 ← R[rs1] ReadData2 ← R[rs2] imm ← SignExt(inst[31:25], inst[11:7])
Execute	Compute effective address	ALUResult ← ReadData1 + imm
Memory	Read rs2 value to memory	Dmem[ALUResult] ← ReadData2
Write Back	Nothing	No register update

Stage Computation: jalr rd, rs1, offset

Stage	Action	Signals / Values
Fetch	Read instruction at PC Compute next address	inst ← IMem[PC] PC + 4
Decode	Read base register rs1 Generate immediate	ReadData1 ← R[rs1] imm ← SignExt(inst[31:20])
Execute	Compute target address	ALUResult ← (ReadData1 + imm) & ~1
Memory	Nothing	No register update
Write Back	Save return address in rd	R[rd] ← PC + 4

- The lower bit of the target address is set to 0 to ensure alignment
  - Instructions must start on an even memory address

Single-Cycle Performance Limitations

- The clock period must accommodate the slowest instruction – lw (uses every stage)
- Example component delays:
  - Instruction Memory: 200 ps
  - Register File Read: 100 ps
  - ALU: 200 ps
  - Data Memory: 200 ps
  - Register File Write: 100 ps
  - Mux and wiring: 50 ps
- Total for lw: 850 ps
- Even though add only needs 650 ps (no data memory access), the clock period must be 850 ps to support lw
  - Every instruction takes one full cycle regardless of complexity
- Also, hardware units sit idle for most of each cycle
- Solution: Pipelining (next lecture)

Single-Cycle Processor Summary

- **Implementation approach:**
  - Express every instruction as a series of simple steps across five stages
  - Follow the same general flow for each instruction type
  - Assemble registers, memories, and predesigned combinational blocks
  - Connect with control logic (muxes selected by the control unit)
- **RISC-V advantages for hardware:**
  - Fixed 32-bit instruction length simplifies fetch dramatically
  - Regular instruction format (register fields always in the same place) simplifies decode
  - Load-store architecture keeps the memory stage clean
- **Limitations of single-cycle design:**
  - Clock period determined by slowest instruction (the lw critical path)
  - Hardware units active for only a fraction of each cycle
  - Too slow to be practical