RV32I is 32 Bits

Making RV32I processor

R type = Register / Register ALU
I Type = register + imm operations
S Type = Load/Store operations
B type = Branch operations

RV32I 3 STAGE CPU CORE

-Instructions needed for project:
- R type: ADD, SUB, AND, OR, XOR, SLT
- I type: ADDI, ANDI, ORI, XORI, LW
- S type: SW
- B type: BEQ, BNE

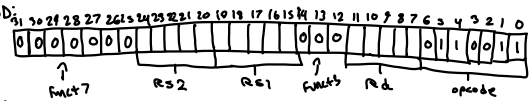
RS1, RS2 are source registers

Rd = Destination Register

Funct 3 and Funct 7 combined with opcodes distinguish instructions

-Explanation for all instructions;

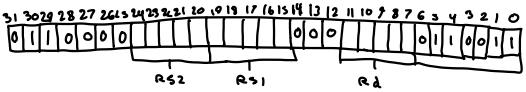
- ADD:



—> add rd, rs1, rs2

- adds registers rs1 and rs2 and stores the result in rd.
- Arithmetic overflow is ignored and the result is simply the low XLEN bits of result

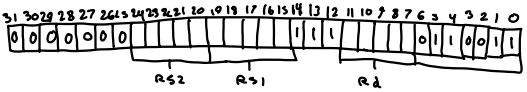
- SUB:



—> Sub rd, rs1, rs2

- subs registers rs1 and rs2 and stores the result in rd.
- Arithmetic overflow is ignored and the result is simply the low XLEN bits of result

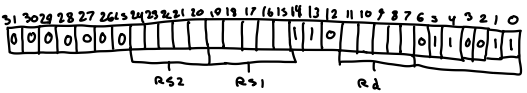
- AND:



—> and rd, rs1, rs2

- performs bitwise AND on registers rs1 and rs2 and place result in rd

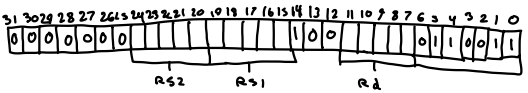
- OR:



—> or rd, rs1, rs2

- performs bitwise OR on registers rs1 and rs2 and place result in rd

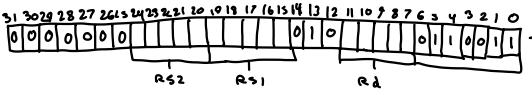
- XOR



—> XOR rd, rs1, rs2

- Performs bitwise XOR on registers rs1 and RS2 and places result in rd

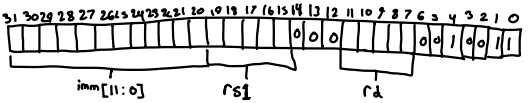
- SLT



—> SLT rd, rs1, rs2

- places the value 1 in register rd if register rs1 is less than register rs2 when both are treated as signed numbers, else 0 is written

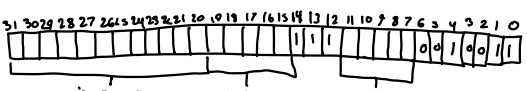
- ADDI



—> addi rd, rs1, imm

- Adds the sign extended 12 bit immediate to register rs1. Arithmetic overflow is ignored and the result is simply the low XLEN bits of the result

- ANDI



-> andi rd, rs1, imm

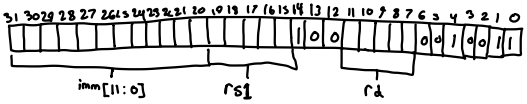
- performs bitwise AND on register rs1 and the sign extended 12-bit immediate and place the result in rd

- ORI

—> ori rd, rs1, imm

- Performs bitwise OR on RS1 and imm sign extended and place result in RD

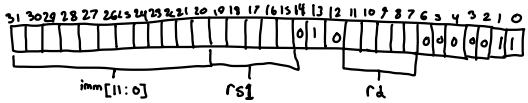
- XORI



→ xori rd, rs1, imm

- performs bitwise XOR on register rs1 and the sign extended imm and places result in rd

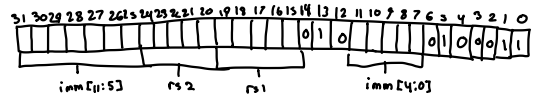
- LW



→ lw rd, offset(rs1)

- loads a 32 bit value from memory of imm address and sign extends to XLEN bits before storing it in register rd

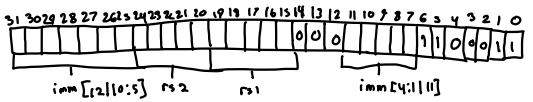
- SW



→ sw rs2, offset(rs1)

- Stores 32 bit value of register rs2 to memory of rs1 + offset of imm

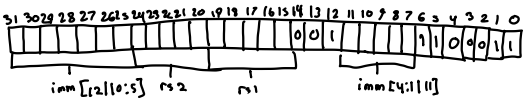
- BEQ



→ beq rs1, rs2, offset

- Take the branch if registers rs1 and rs2 are equal pc += sext(offset)

- BNE



→ bne rs1, rs2, offset

- Take the branch if registers rs1 and rs2 are not equal

3 STAGE PIPELINE:

- IF = PC -> Instruction memory (not data memory)
- Ex = Decode + ALU + branch (data memory access), regWrite signal, memtoreg
- WB = Register writeback

- Instruction Fetch
  - Program Counter
  - Instruction Memory
  - PC + 4 adder
- Execute Stage
  - Register File
  - Immediate Generator
  - ALU
  - Branch comparator
  - Data memory
- Writeback Stage
  - Writeback MUX
  - Register file write port

HAZARDS:

- Load Use Hazard: When a instruction needs data from a load instruction that hasn't finished retrieving the data from memory yet, creating a data dependency where the next instruction tries to read the value too early

  Solution: Stall IF for 1 clock cycle
  insert bubble in Ex

- Branch Taken Hazard: When a processor incorrectly guesses whether a conditional branch will jump to a new instruction or continue sequentially, loading the wrong instructions into the pipeline, which then needs to be flushed, wasting time and slowing performance

  Solution: FLUSH next instruction, update pc to target

Control Signals:

- RegWrite = enable register write
- MemRead = read memory
- MemWrite = write memory
- MemToReg = select ALU vs memory
- Branch = branch enable
- ALUSrc = immediate vs register (rs2)
- ALUOp = ALU operation select

Pipeline Registers:

- IF/EX Register
  - instruction
  - PC
  - PC + 4
- Ex/WB Register:
  - ALU Result
  - memory read data
  - rd
  - RegWrite
  - MemToReg

SystemVerilog Modules:

- ☆ PC.sv
- ☆ imem.sv
- ☆ regfile.sv
- ☆ alu.sv
- ☆ imm_gen.sv
- ☆ control.sv
- ☆ if_ex_reg.sv
- ☆ ex_wb_reg.sv
- ☆ top.sv

Control table:

R-Type
RegWrite = 1
MemRead = 0
MemWrite = 0
MemToReg = 0
ALUSrc = 0
Branch = 0
ALUOp = Func

I-Type
RegWrite = 1
MemRead = 0
MemWrite = 0
MemToReg = 0
ALUSrc = 1
Branch = 0
ALUOp = Func (Either Func3 or Func7)

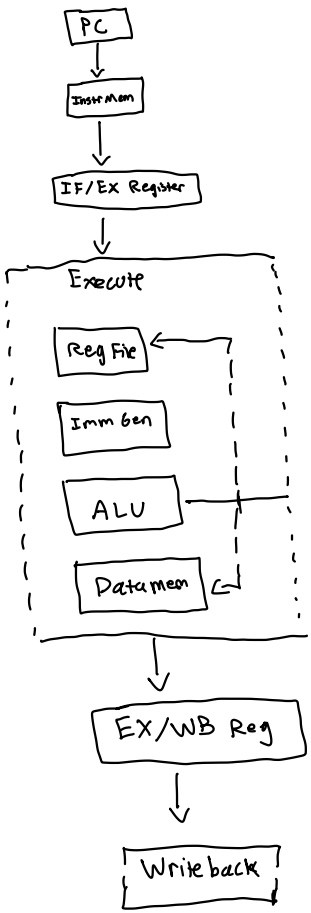
Load
RegWrite = 1
MemRead = 1
MemWrite = 0
MemToReg = 1
ALUSrc = 1
Branch = 0
ALUOp = ADD

Store
RegWrite = 0
MemRead = 0
MemWrite = 1
MemToReg = X
ALUSrc = 1
Branch = 0
ALUOp = ADD

Branch
RegWrite = 0
MemRead = 0
MemWrite = 0
MemToReg = X
ALUSrc = 0
Branch = 1
ALUOp = SUB

☆ For a load instruction, the write back MUX selects the data read from the memory rather than the ALU result because the value to be written comes from the memory, not the address computation

High Level Block Diagram:



IF/EX Register:
- instr
- pc_plus4

Ex/WB Register:
- alu_result
- mem_data
- rd
- RegWrite
- MemToReg

Full 3-Stage RISC-V Datapath: