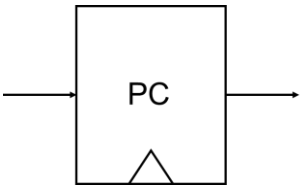
This design works for add, addi, sub, and, andi, or, ori, slt, slti, lw, sw, beq, bne, blt, bge, jal, jalr.

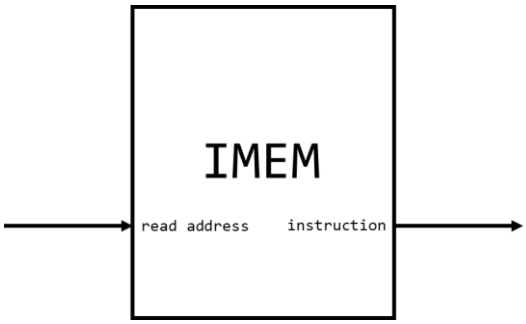
I. Modules

✓Program Counter



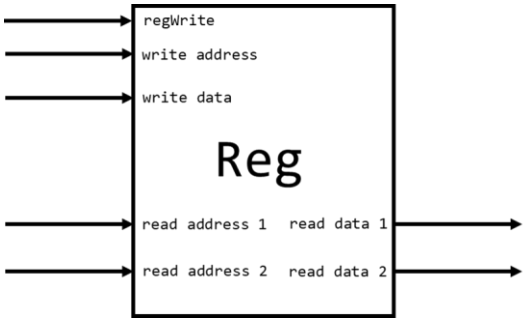
- store the instruction address
- positive edge triggered
- output the instruction address when triggered

✓ Instruction Memory



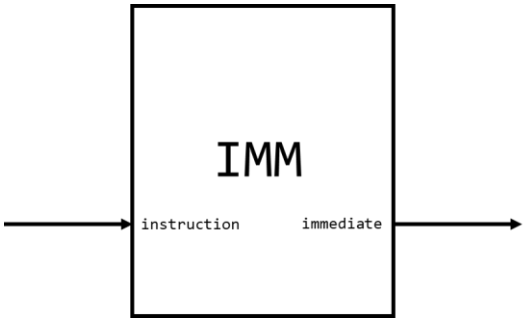
- store the instructions
- read only
- input 32-bit address
- output the instruction

✓ Register File



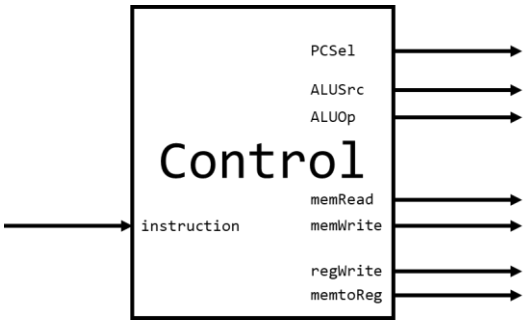
- 32 registers => 5-bit address
- if `regWrite` is asserted,
  write 32-bit data to the register
- output 2 32-bit data stored in the register

✓ Immediate Generator



- decode the instruction
- determine instruction type from opcode
  R / I / S / B / J
- refer to RISC-V ISA page

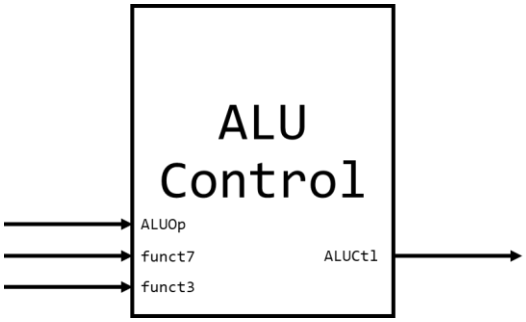
✓ Control Unit



- PCSel: select PC input, 2 bits, 3 options
  B / jal: 2'b01, PC + imm
  jalr: 2'b10, reg data 1 + imm
  R / I / load / store: 2'b00, PC + 4
- memRead: asserted only for load, 1 bit
- memWrite: asserted only for store, 1 bit
- regWrite: de-asserted only for store and branch
- memToReg: select written data to register file
  2 bits, 3 options,
  2'bxx (don't care) when regWrite = 0,
  R / I: 2'b00, output of ALU
  load: 2'b01, data from memory
  jal / jalr: 2'b10, PC+4

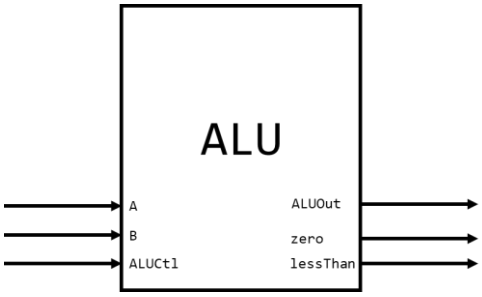
- ALUSrc: select ALU input B, 1 bit,
  R / B: 1'b0, reg data 2
  jal: 1'bx, no ALU operations needed
  I / load / store / jalr: 1'b1, immediate
- ALUOp: Used in module ALU Control to determine ALU control signal, 2 bits

✓ ALU Control



- load / store / jalr: ALUOp = 0, choose "+"
- branch: ALUOp = 1, choose "-" to compare input A and B of ALU
- R: ALUOp = 2, determine corresponding operation with funct7 and funct3, which is 30 and 14:12 bits in the instruction
- I: ALUOp = 3, determine operation with funct3

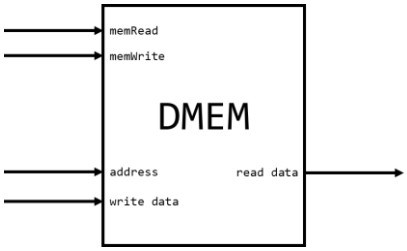
✓ Arithmetic and Logic Unit



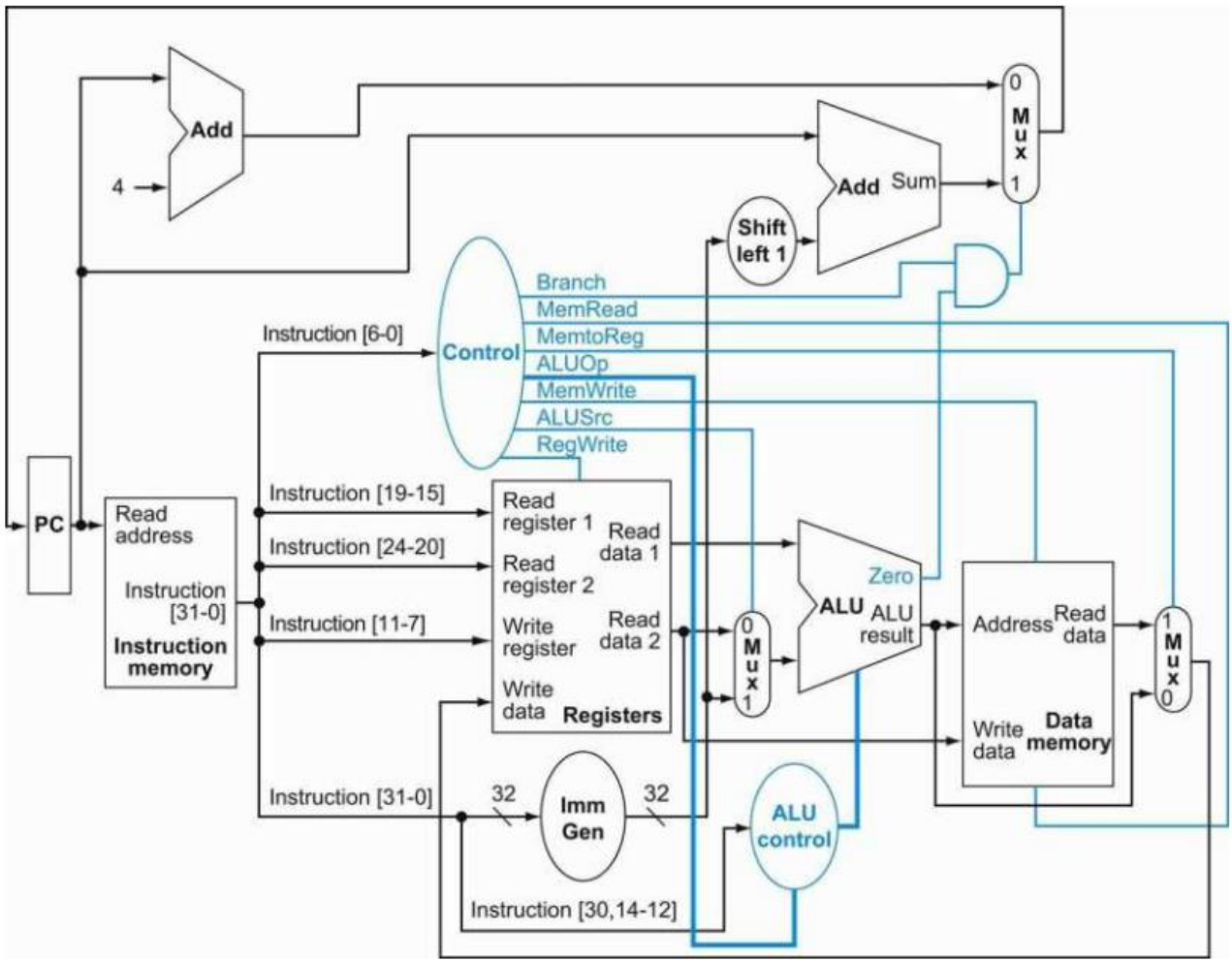
ALU control lines	Function
0000	AND
0001	OR
0010	add
0110	subtract

textbook p.500

✓ Data Memory



II. Datapath



The picture above is from page 508 of the textbook.

But immediate generator generates 32-bit immediate.

In the code, the MUX at the right of the data memory and the one determining the input of the program counter becomes 3×1, and they are implemented with 4×1 MUX.

The condition of when branch is taken is still determined directly in the `SingleCycleCPU.v`, but you can add a branch comparator and implement it in the module if you like.