CPU Control Design

ALU + Regfile + Memory + PC + Control FSM

Which addressing modes should you implement?

- Read the ISA.pdf, as well as the CR16 programmer reference manual on Canvas
- Some addressing modes we have to implement, others are your choice
- Register mode (R-type instructions): compulsory
  - ADD Rsrc Rdest: $R_{dest} \leftarrow R_{src} + R_{dest}$
  - Arithmetic, logical, Move: are R-type instructions
  - Make a list of all these instructions

Which addressing modes should you implement?

- Immediate mode (I-type instructions): compulsory
  - ADDI Imm Rdest: $R_{dest} \leftarrow \$Imm + R_{dest}$
  - Arithmetic, logical, Move: are included in I-type instructions
  - ISA.pdf shows: ADDI, SUBI, CMPI, ANDI, XORI,…, MOVI
  - They behave similarly as R-type instructions
  - Make a list of all these instructions

Which addressing modes should you implement?

- Direct/Absolute Addressing mode (Dir-type instructions): Not compulsory, can do without
  - ADD Rdest, [addr] : $R_{dest} \leftarrow R_{dest} + [mem\ addr]$
  - [mem addr] = data resides in memory, whose address is “mem addr”
  - Not implemented in CR16
  - More complicated, requires memory access

Which addressing modes should you implement?

- Indirect Addressing mode (Ind-type instructions): Compulsory
  - LOAD Rdest, Raddr : $R_{dest} \leftarrow [R_{addr}]$
    - Load data into Rdest, where data resides in memory whose address in stored in Raddr register
    - STOR Rsrc, Raddr: $[Raddr] \leftarrow Rsrc$
    - Store the data of Rsrc into memory at address [Raddr]
    - We will build a Load-Store machine: Use LOAD/STOR to fetch data into regfile, and then perform R-type instructions for computations!

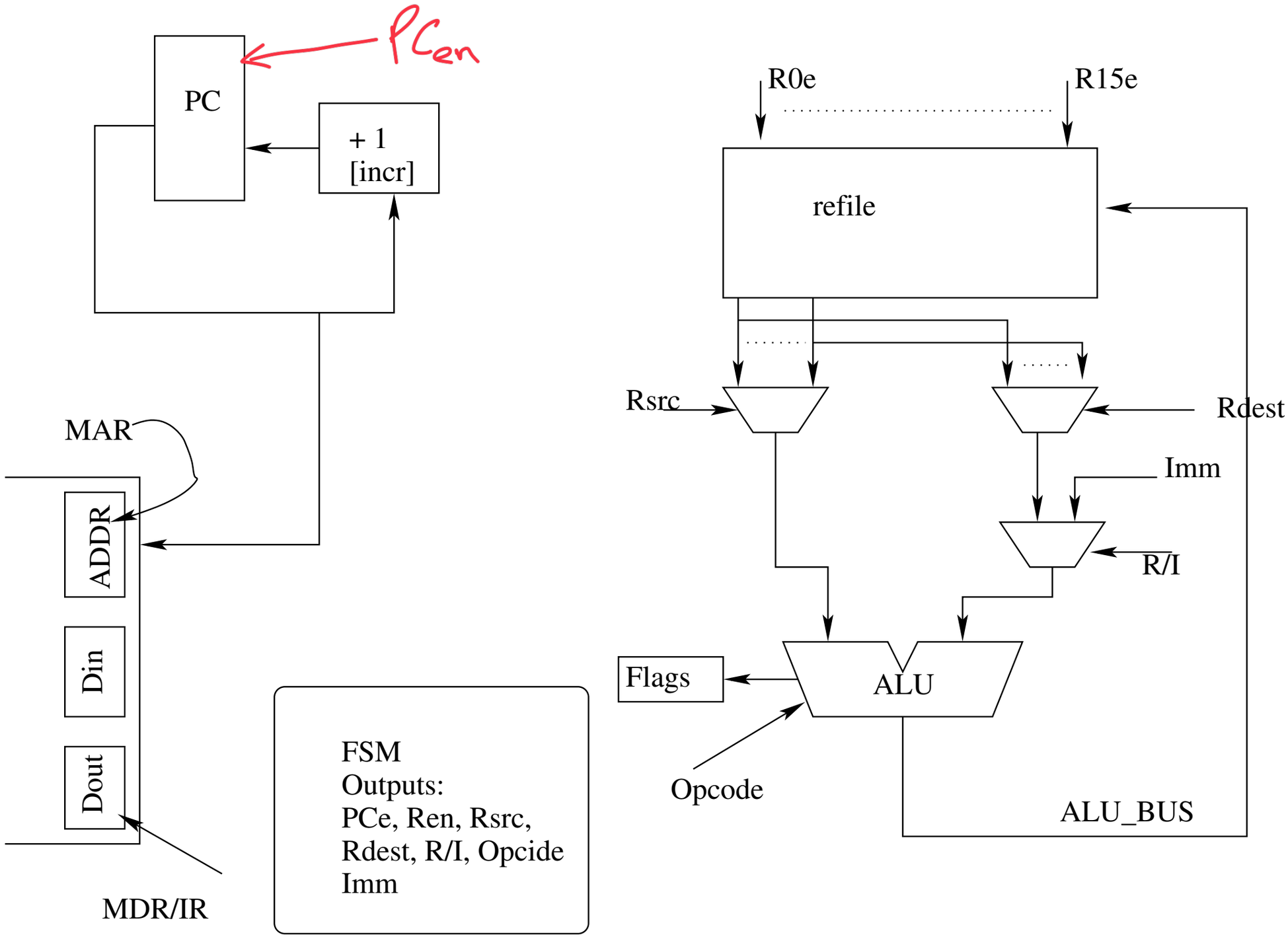
Which addressing modes should you implement?

- PC-Relative/Displacement Addressing mode (Rel-type instructions): Compulsory
  - Conditional Branches and Jumps: *e.g., "Bcond disp":*
  - If condition cond is met, branch to memory address (PC + disp)
    - PC <= PC + disp
    - ISA.pdf: *disp* = 8-bit 2's complement integer (bit vector), disp is given in the opcode
    - Branch versus Jump: "Jcond R_{target}": Jump to address that is stored in R_{target}
      - If condition is met, then PC = R_{target}
    - Condition codes are given on the page 6 in ISA.pdf. You have to implement a few, EQ, NE: Zero flags, CS, CC: Carry, GT, LE: Negative, FS, FC: Overflow

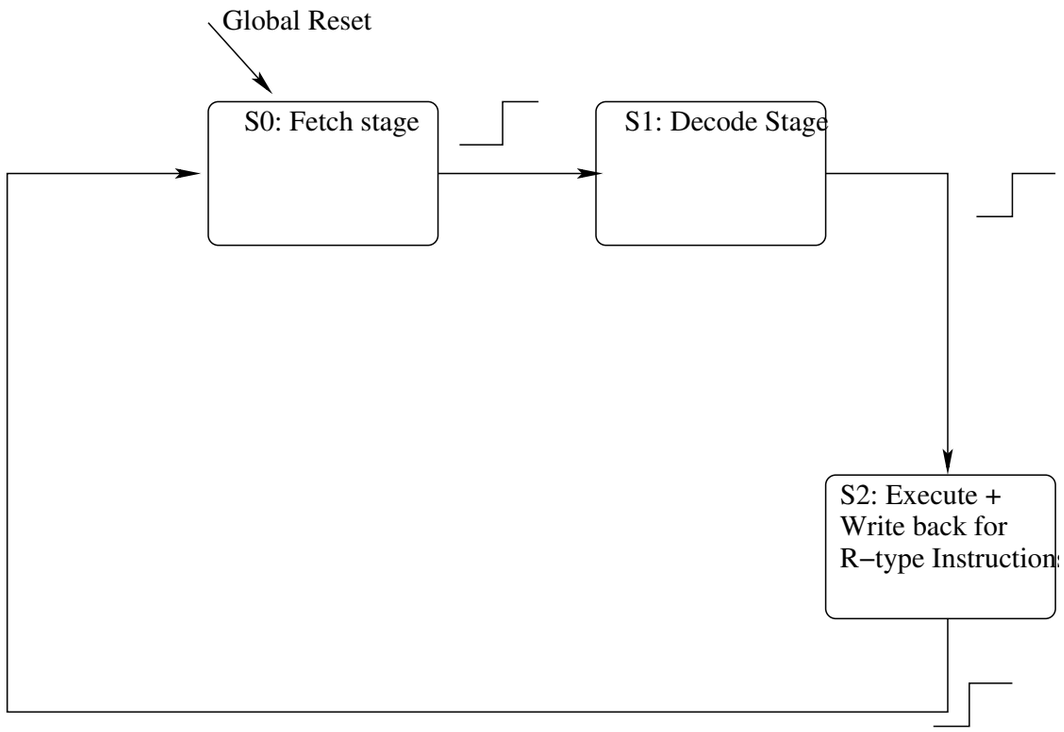
Approach

- First you should implement R-type instructions
    - Design Datapath to support R-type instructions: Program Counter [PC = PC + 1]
    - Design FSM
- Then implement Load/Store instructions
    - Augment the Datapath: [PC = Raddr]
    - Augment your FSM for Load/Store Instructions
- Finally, include Jumps and Branches

Datapath for R-type Instructions

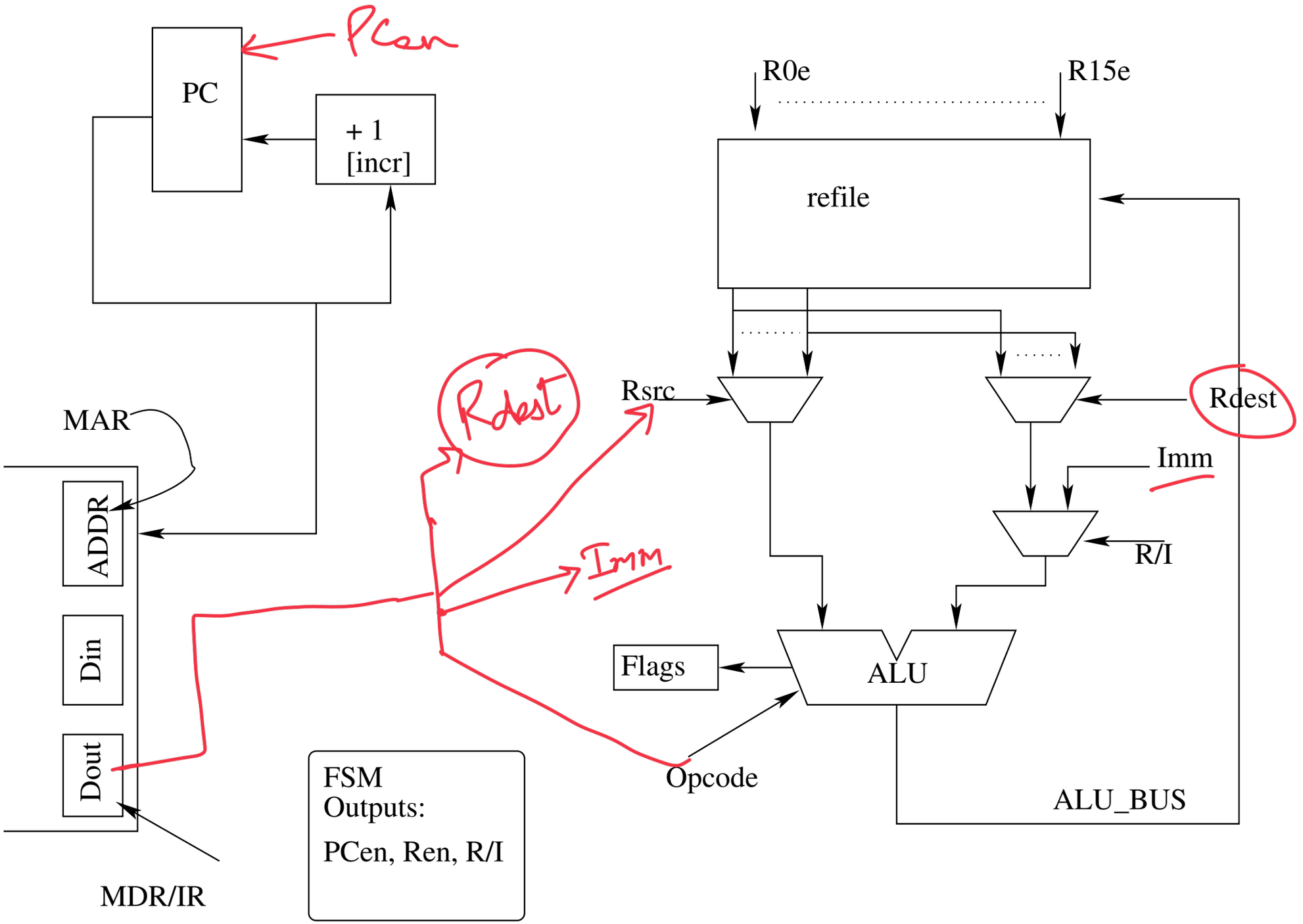


FSM Design for R-type



- State S0: PCe = 0; Ren=0; Rsrc = 4’bx; Rdest=4’bx; Opcode = 8’bx; R/I = 1’bx; Imm=8’bx
  - Next state = S1
- State S1:
  - If Opcode = R-type, then NS = S2
- PCe = 0; Ren=0; Rsrc = 4’bx; Rdest=4’bx; Opcode = 8’bx; R/I = 1’bx; Imm=8’bx
- State S2: **PCe = 1**
  // Don’t write PC = PC + 1 in your FSM’s logic in Verilog

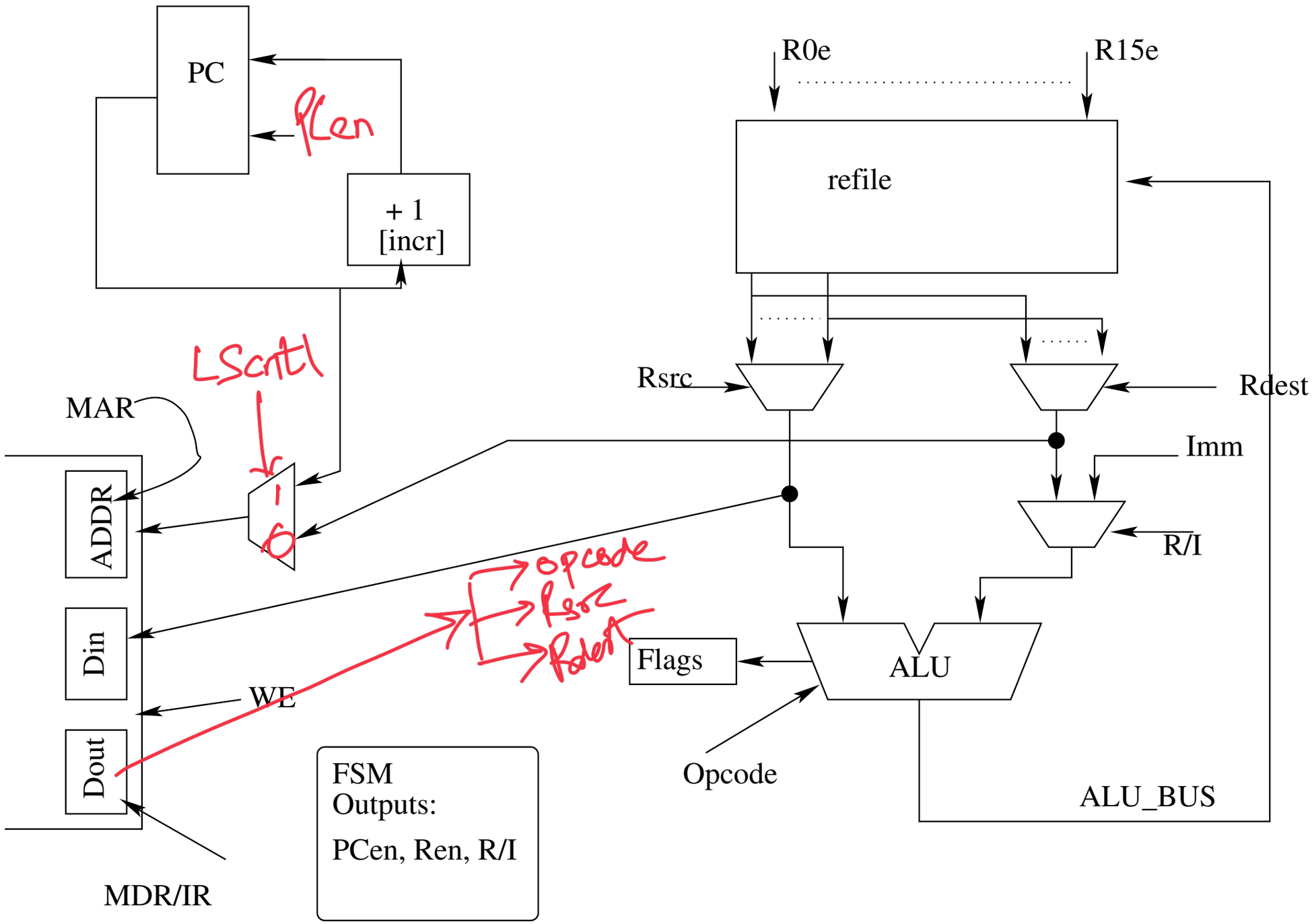
A More Structural Design



- More structural design, leaner/simpler FSM

Store Instruction

- STOR Rsrc Raddr: Store data (in Rsrc) in memory at address given in Raddr



Update your FSM with Store Instruction