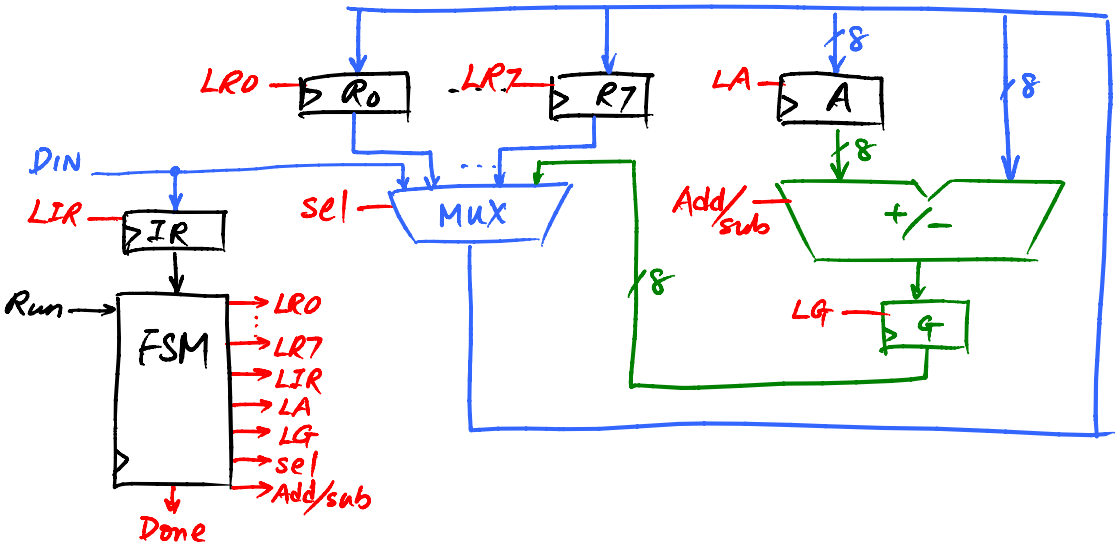
I.2 Processor Design

Consider a set of n-bit registers $R_0, \cdots R_7$, we wish to be able to initialize a register with data, to transfer content from one register to another register, and to add/sub contents of registers



→ the processor needs instructions to know what to do at any given time. we use instruction register (IR) to hold instructions.

Code	Operation	
00	mv Rx, Ry	// Copy Rx ← [Ry]
01	mvi Rx, #D	// initialize Rx with some data
10	add Rx, Ry	// Rx ← [Rx] + [Ry]
11	sub Rx, Ry	// Rx ← [Rx] − [Ry]

encode IR using 8 bits

`11xxxyyy`

00 → mv

eg. to copy content of R_4 in R_2 = mv R2, R4 ⟹ 00 010 100

(mv, R2, R4)

to initialize R_3 = mvi R3, #7 ⟹ 01 011 ddd

00000111 ← #7

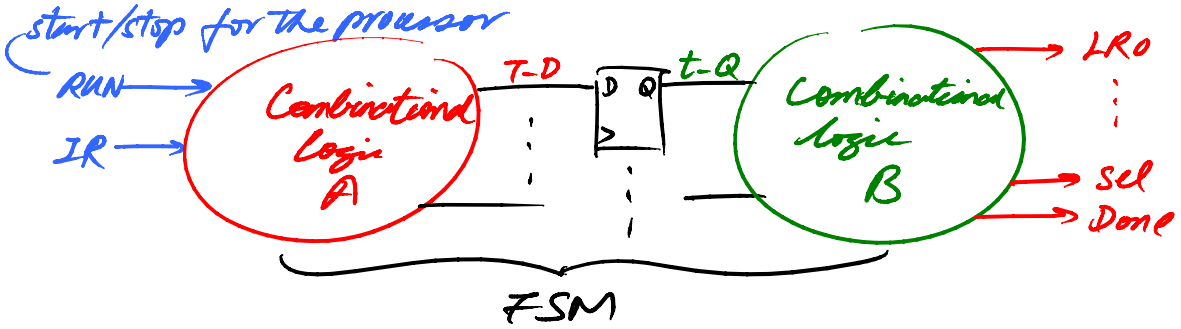
(mvi, R3)

here we are assuming data on DIN next will be taken as #7.

⟹ mv, mvi, add, sub are called assembly language instructions
the encoding of instructions is called Opcode `IIXXXYYY`
⟹ an assembly tool processes Opcode and produces machine code

Execution of instructions: each instruction appears on DIN, and is stored in IR. Then in the following clock cycles the FSM will set the signals (sel, LR0, LR1, ... LG) to complete the instruction.

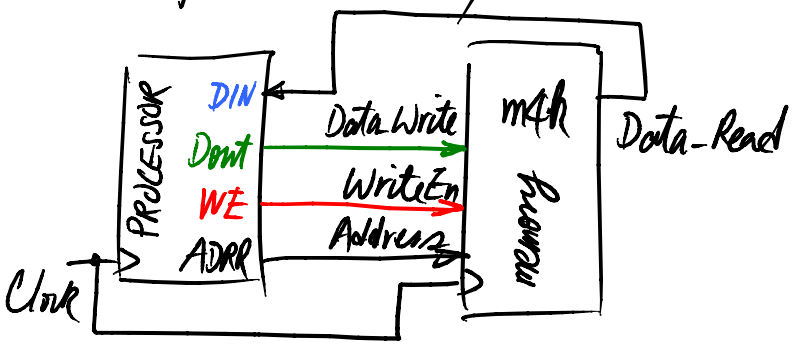
Instruction	T0	T1	T2	T3
mv	LIR	Sel=Ry LRx=1, Done		
mvi	LIR	Sel=DIN LRx=1, Done		
add	LIR	Sel=Rx LA=1	Sel=Ry Add/Sub=1, LG=1	Sel=G LRx=1, Done
sub	LIR	Sel=Rx LA=1	Sel=Ry Add/Sub=0, LG=1	Sel=G LRx=1, Done

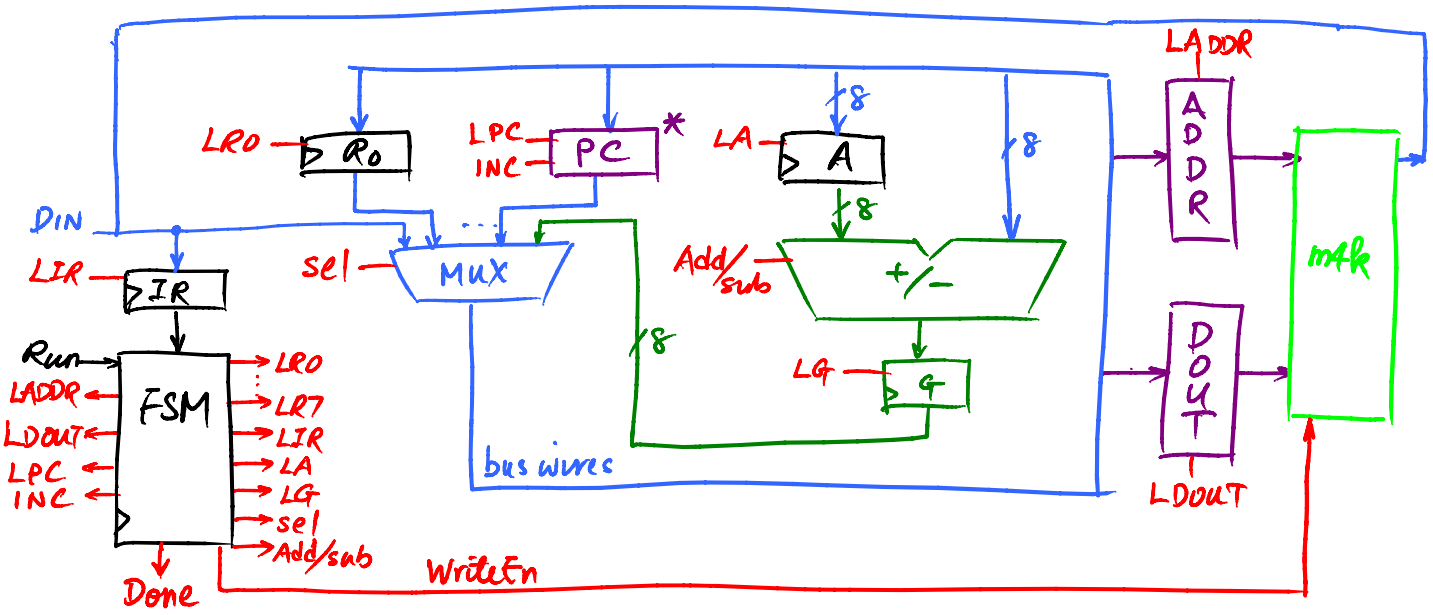



```
module simple_processor(DIN, Resetn, Clock, Run, Done);
    input [7:0] DIN;
    input Resetn, Clock, Run;
    output Done;
    reg [2:1] T_D, t_Q;
    parameter T0=2'b00, T1=2'b01, T2=2'b10, T3=2'b11;
      :
// FSM state transition
    always@(t_Q, Run, Done)          // t_Q is current state, t_D is next state
      case (t_Q)
        T0: if (!Run) T_D=T0;
            else T_D=T1;
        T1: if (Done) T_D=T0;
            else T_D=T2;
        T2: T_D=T3;
        T3: T_D=T0;
      endcase
```

```
// FSM outputs                    CS.   operation DIN[7:6]
always@(t-Q, II, XXX, YYY)                DIN[2:0] Register Y
  case(t-Q)                                DIN[5:3] Register X
    // default values
    LR0=0, LR1=0, ... LRA=0,
    LG=0, Done=0, LIR=0;
  T0: LIR=1;
  T1: case(II)
        2'b00: begin                    ~
                 sel = YYY;             Copy Rx←[Ry]
                 LR = XXX;
                 Done=1;
               end
        2'b01: begin                    ~
                 sel = DIN;             initializing Rx
                 LR = XXX;
                 Done=1;
               end
        :
      endcase
  T2: case(II)
      :
endcase
```

To enhance the processor, we want to allow information retrieving from/to memory





* PC – program counter is normally incremented at the end of each instruction, so that the next instruction can be read from memory. PC can also be loaded from the bus wires to perform a branch (loop) to an arbitrary address.