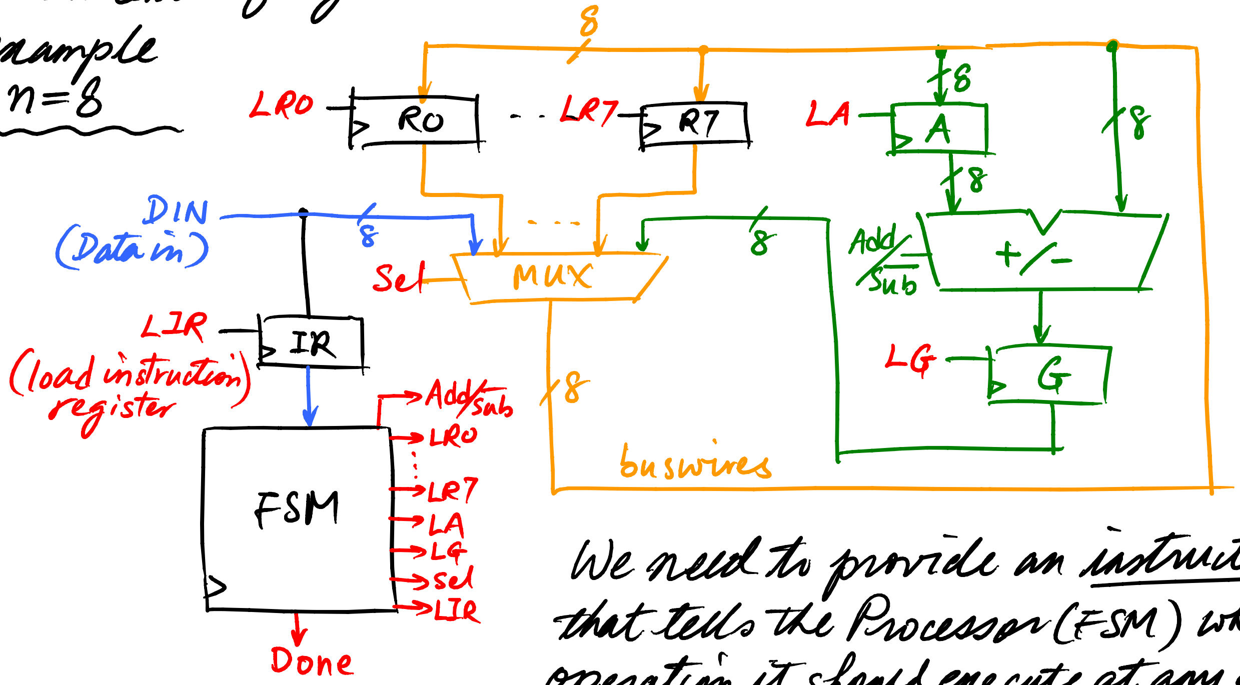
Processor Design: consider a set of n-bit registers $R_0 \dots R_7$. We wish to be able to $\boxed{\text{initialize}}$ a register with data, to $\boxed{\text{transfer}}$ content from one register to another, to $\boxed{\text{add/subtract}}$ contents of registers.

example
n=8



We need to provide an instruction that tells the Processor (FSM) what operation it should execute at any given time. To use IR to hold instructions.

IR

0.	mv	Rx, Ry	// copy Rx ← [Ry] (content)
1.	mvi	Rx #D	// initialize a register
2.	add	Rx, Ry	// Rx ← [Rx] + [Ry]
3	sub	Rx, Ry	// Rx ← [Rx] − [Ry]

encoding of instructions (to appear on DIN):

$\boxed{\text{IIXXXYYY}}$

mv=00, mvi=01, add=10, sub=11

example: to copy the content of R4 into R2:

mv R2, R4 $\longrightarrow$ $\underbrace{00}_{mv}\ \underbrace{010}_{R2}\ \underbrace{100}_{R4}$

to initialize R3: mvi R3, #7 $\longrightarrow$ $\underbrace{01}_{mvi}\ \underbrace{011}_{R3}$ddd (*)
$\longrightarrow$ 00000111

(*) we assume that after reading the mvi code, the processor can read #D from DIN mv, mvi, add. sub, etc are called assembly language instructions

⇒ all processors have a unique assembly language

The encoding of instructions is called the Opcode (IIXXXYYY)

⇒ an assembly tool produces Opcode and machine code

Execution of Instructions

→ each instruction appears on DIN, and is stored into IR. Call this clock cycle TΦ. Then in the following clock cycle the FSM will set the control signals (LRΦ, ··· Sel) as needed to complete the instruction.

Inst.	TΦ	T1	T2	T3

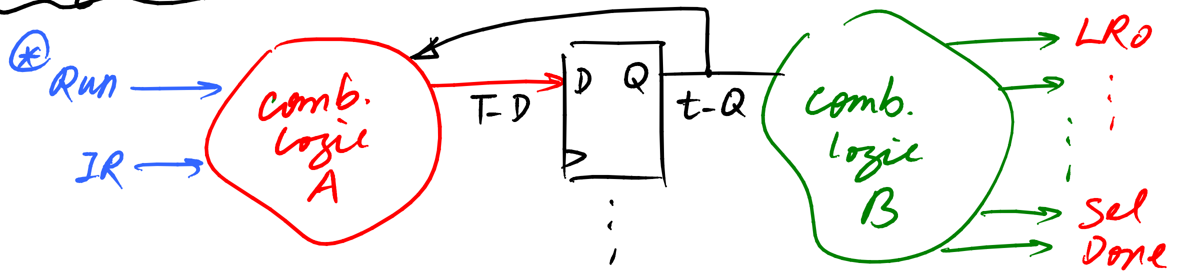
mv	LIR	Sel=Ry
		LRx=1, Done		

mvi	LIR	Sel=DIN
		LRx=1, Done		

add	LIR	Sel=Rx,
		LA=1
			Sel=Ry
			Add/sub=1, LG
				Sel=G
				LRx=1, Done

sub	LIR	Sel=Rx,
		LA=1,
			Sel=Ry
			Add/sub=0, LG
				Sel=G
				LRx=1, Done

Verilog code



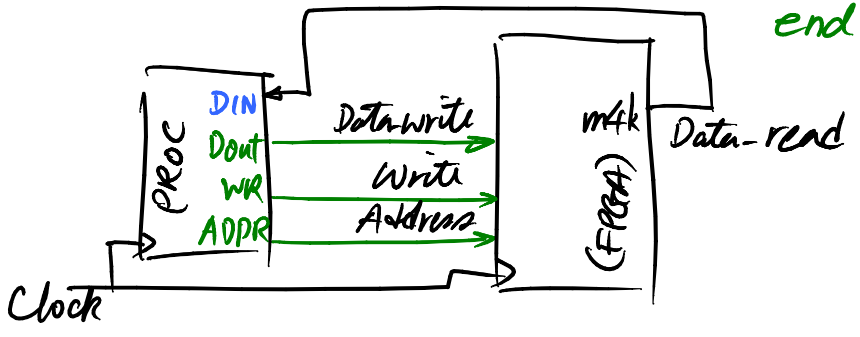
*Run: starts/stops processor

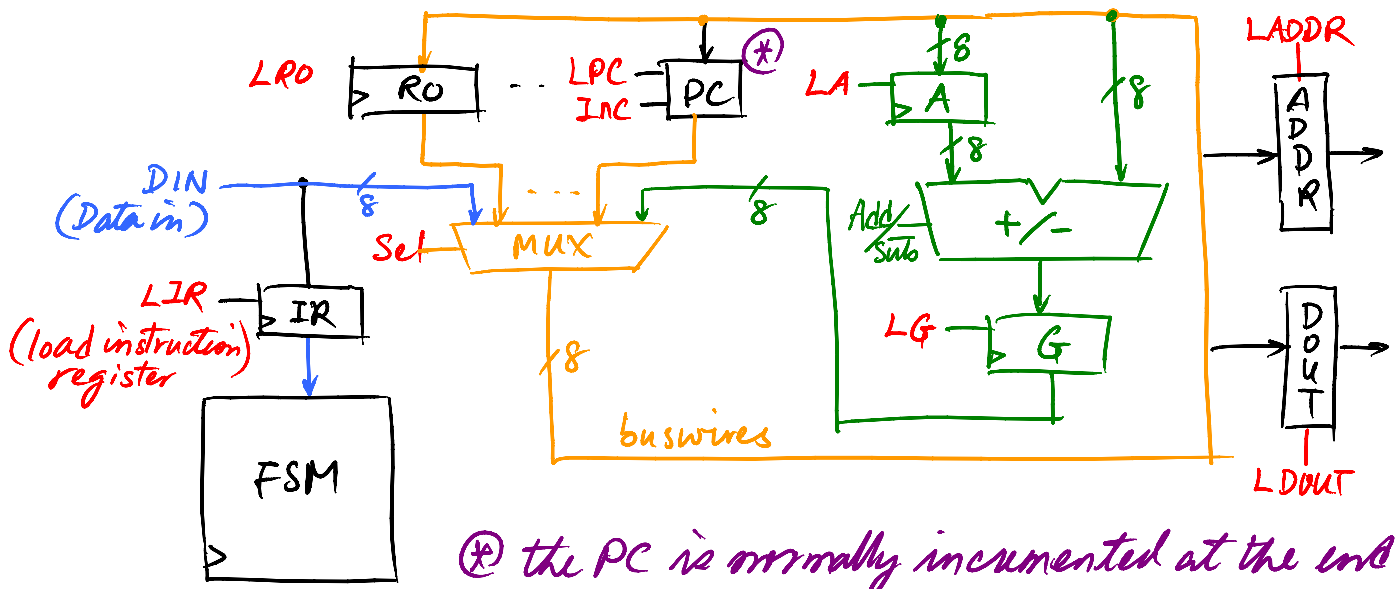
```
module processor(DIN, Resetn, Clock, Run, Done);
    input [7:0] DIN ;
    input Resetn, clock, Run;
    output Done ;
    reg [2:1] T_D, t_Q;
    parameter TØ=2'b00, T1=2'b01, T2=2'b10, T3=2'b11;
      :
```

```
// FSM state Table
  always@(t_Q, Run, Done)
    case(t_Q)
      TØ: if(!Run) T_D=TØ;
          else T_D=T1;
      T1: if(Done) T_D=TØ;
          else T_D=T2;
      T2: T_D=T3 ;
      T3: T_D=TØ;
    endcase
```

```
// FSM outputs
  always@(t_Q, II, XXX, YYY)
    case(t_Q)
     // default values
     LRO=0, LR1=0, ... LA=0,
     LG=0, Done=0, LIR=0;
     TØ : LIR=1;
     T1 : case(II)
            2'b00: begin                 copy Rx←[Ry]
                     sel=YYY;
                     LR=XXX;
                     Done=1;
                   end
            2'b01: begin                 initialization
                     sel=DIN;
                     LR=XXX;
                     Done=1
                   end
              :
          end case
     T2: case(II)
           etc. ...
  endcase
```

This processor needs to be enhanced to allow it to automatically read instructions from a memory device, also it should be able to read/write data from/to that memory





(*) the PC is normally incremented at the end of each instruction, so that the next inst. can be read from memory. Also PC can be loaded from the buswires to performs a branch (loop) to an arbitrary address.