Programmable Processors

Control Unit

Refresh – The Sizes of Things

- Instruction memory (ROM):128 X 16
- Data memory (RAM): 256 X 16
- Register file: 16 X 16
- ALU: Two 16-bit inputs, 16-bit output

- Instruction Set – List of allowable instructions and their representation in memory
- Each of our instructions is 16 bits long
- Most of them contain some address information
- General form : **operation source destination**

NOOP instruction – **0000 0000 0000 0000**

STORE instruction – **0001 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$**

LOAD instruction – **0010 $d_7d_6d_5d_4d_3d_2d_1d_0$ $r_3r_2r_1r_0$**

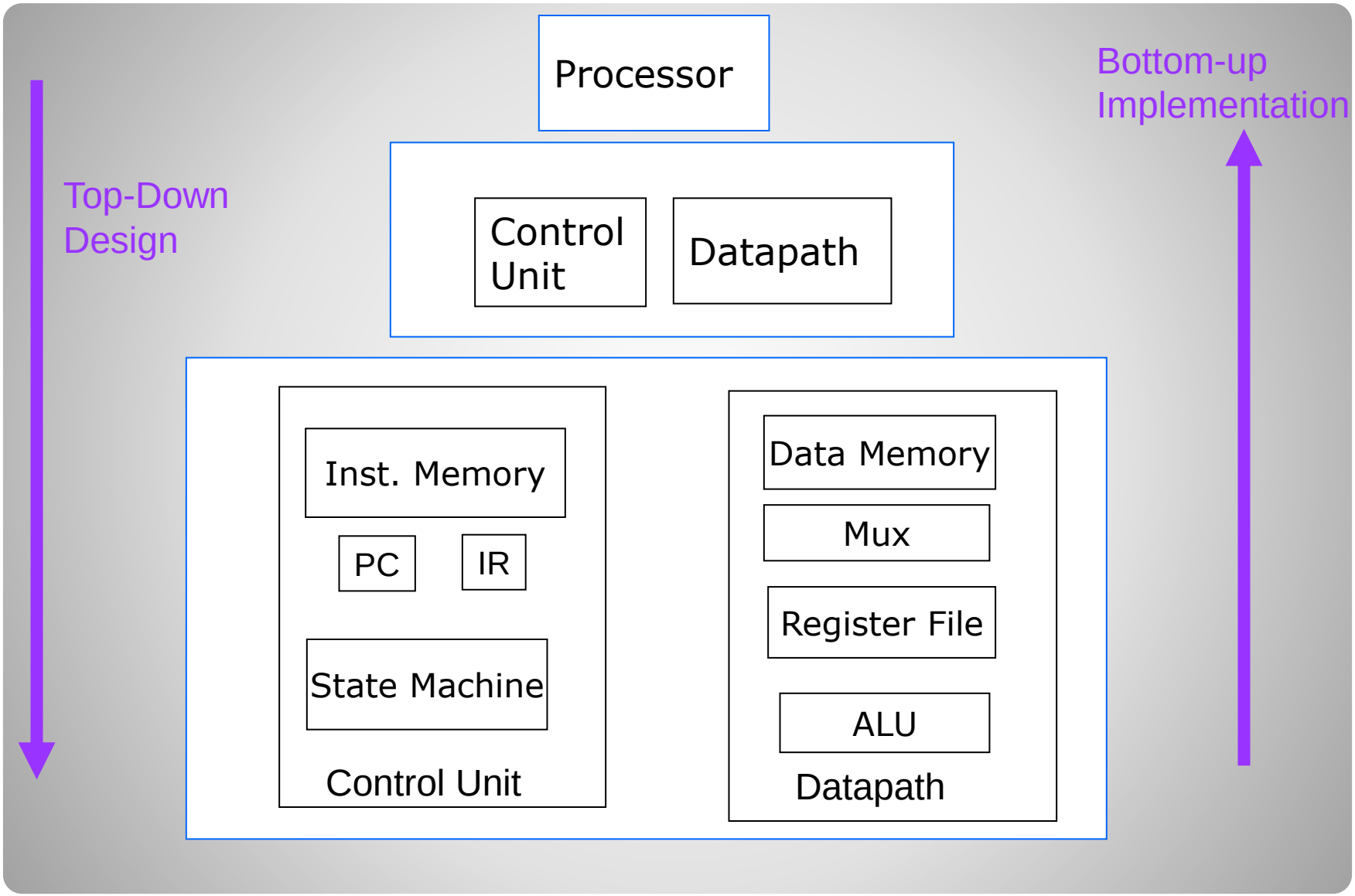
ADD instruction – **0011 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**

SUBTRACT instruction – **0100 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**

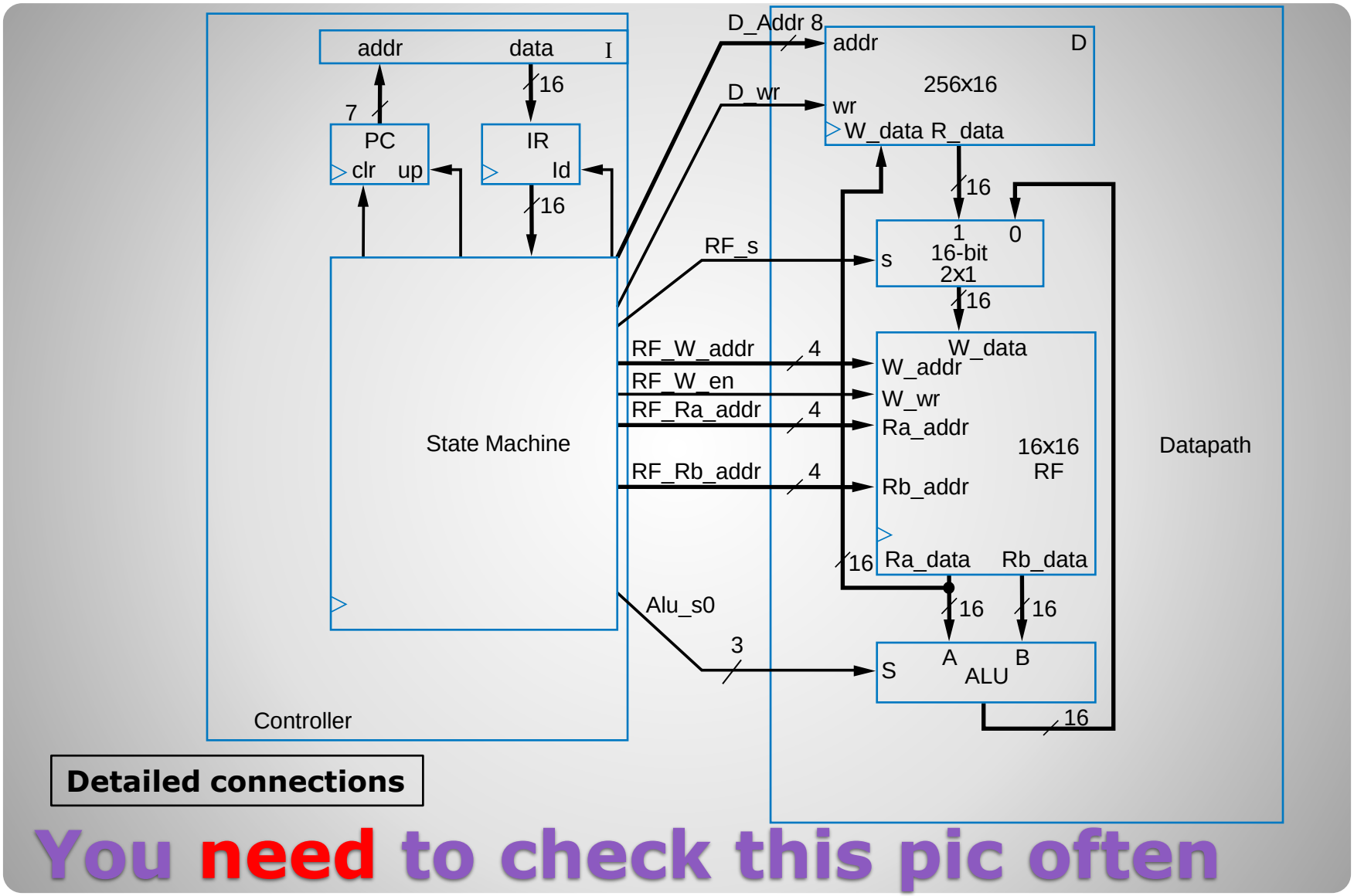
HALT instruction – **0101 0000 0000 0000**

'r's are Register File locations (4 bits each)
'd's are Data Memory locations (8 bits each)

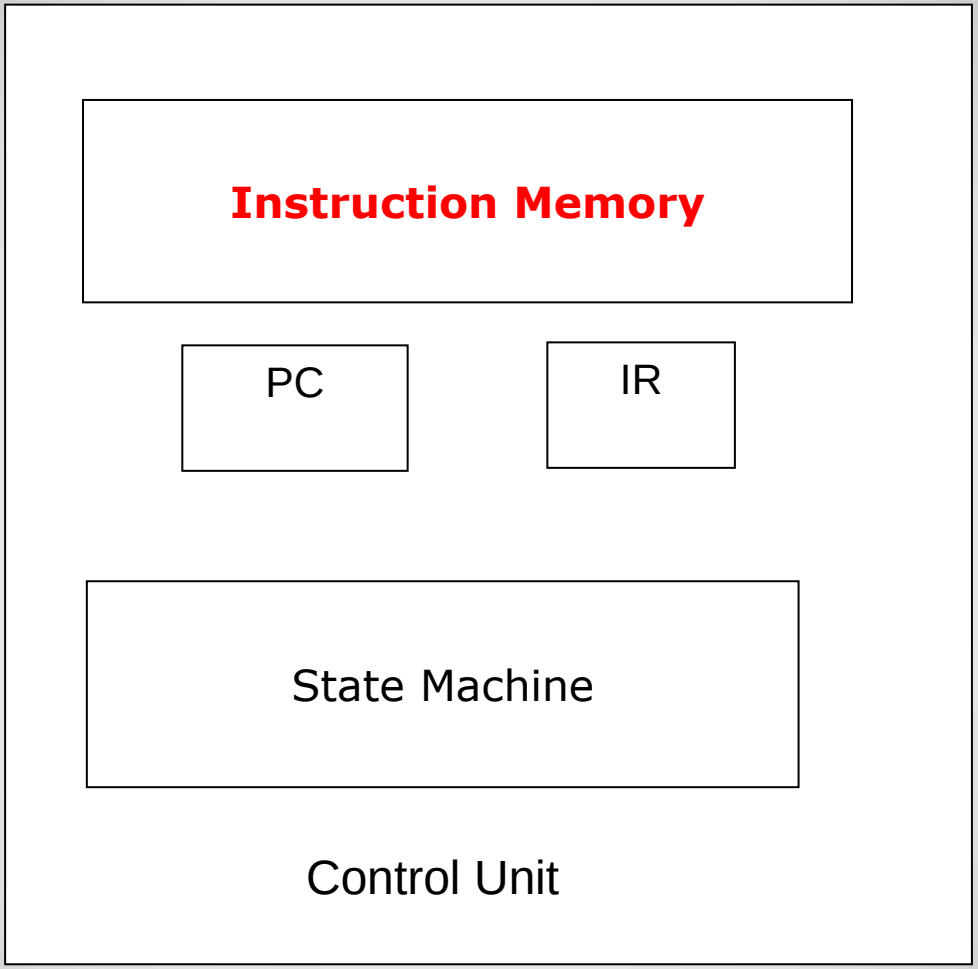
Refresh – 6-Instruction Processor



Pic of Processor Modules



You need to check this pic often when connecting multi-modules



Along the Control Unit side ...

Instruction Memory Module

- Similar as previous examples
  - Under Quartus, create a new .mif file, name it say, `A.mif`. Later manually type into each location a decimal number converted from your 16-bits instructions.
  - Use Altera Library and build a 1-port ROM module, say, `InstMemory.v`. Make sure during the configuration procedure you have associated `InstMemory.v` with `A.mif`.

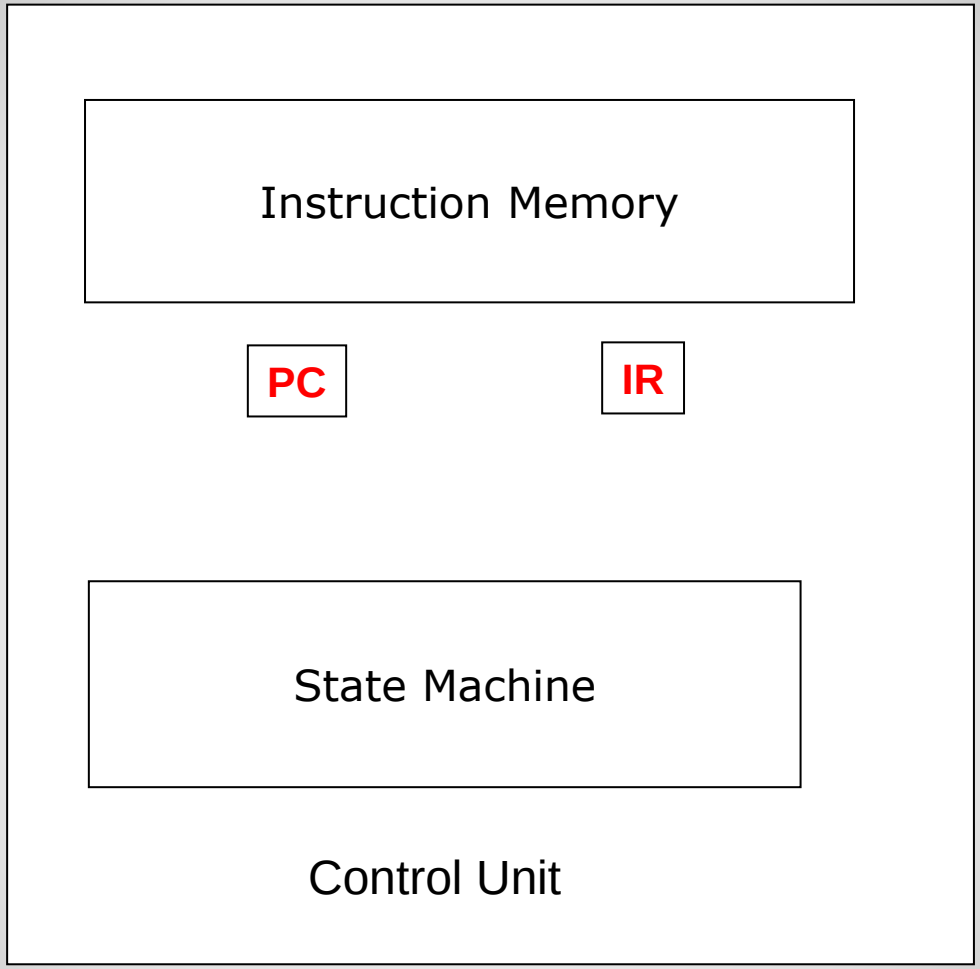
How to test the ROM memory?

Regarding the memory.v file

Here should be the mif file you created for your RAM/ROM

```
defparam
        altsyncram_component.clock_enable_input_a = "BYPASS",
        altsyncram_component.clock_enable_output_a = "BYPASS",
        altsyncram_component.init_file = "DataMemory.mif",
        altsyncram_component.intended_device_family = "Cyclone IV E",
        altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=YES,INSTANCE_NAME=DATA",
        altsyncram_component.lpm_type = "altsyncram",
        altsyncram_component.numwords_a = 256,
        altsyncram_component.operation_mode = "SINGLE_PORT",
        altsyncram_component.outdata_aclr_a = "NONE",
        altsyncram_component.outdata_reg_a = "UNREGISTERED",
        altsyncram_component.power_up_uninitialized = "FALSE",
        altsyncram_component.read_during_write_mode_port_a = "NEW_DATA_NO_NBE_READ",
        altsyncram_component.widthad_a = 8,
        altsyncram_component.width_a = 16,
        altsyncram_component.width_byteena_a = 1;
```

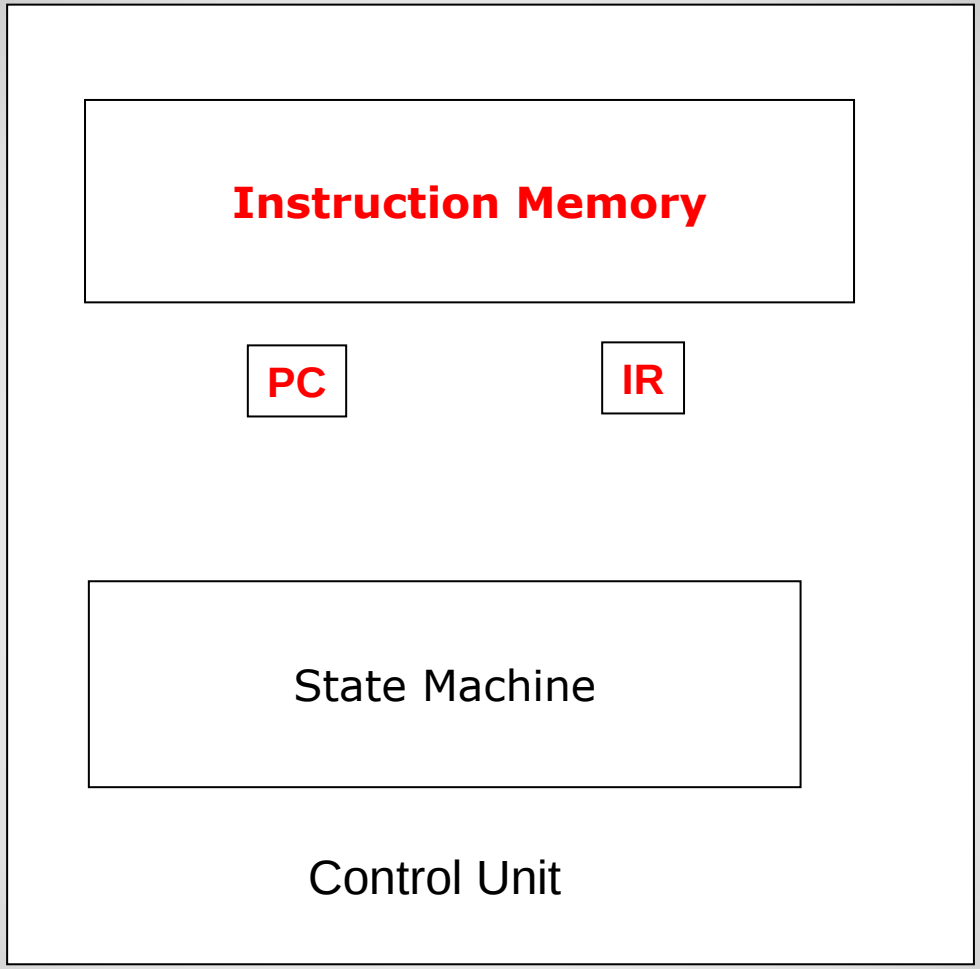
Here if showing “clock” change it to be “UNREGISTERED”



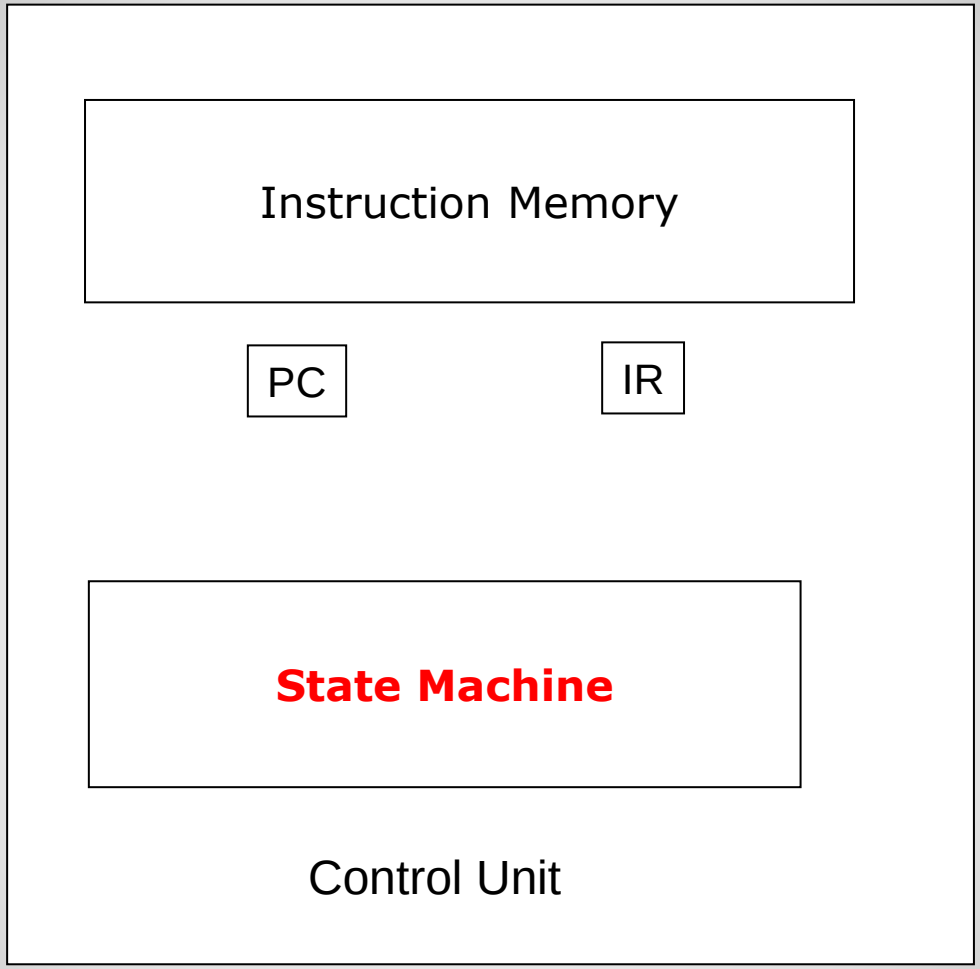
Along the Control Unit side ...

PC and IR Modules

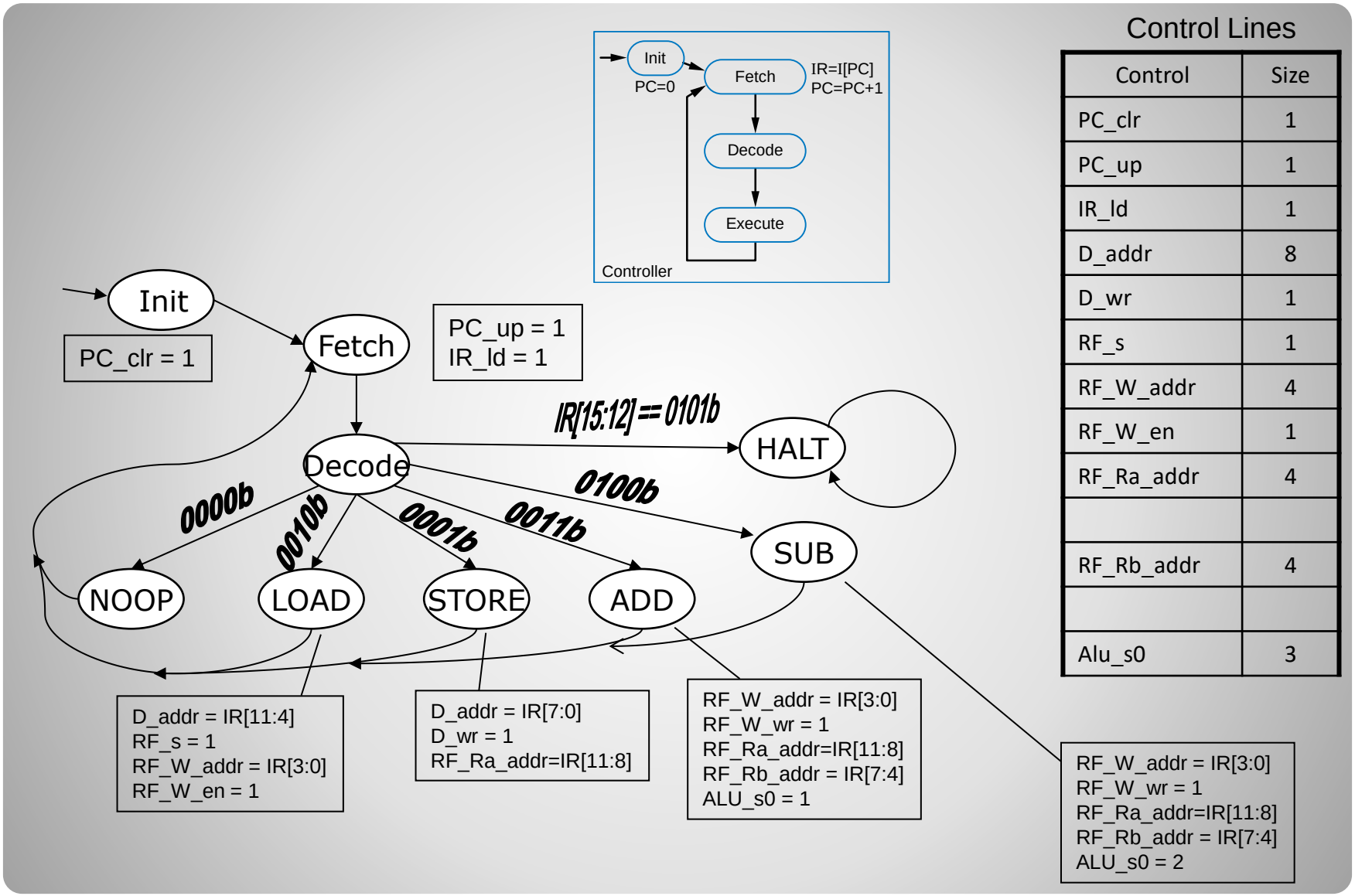
- PC is purely a counter
  - Required input signals: Clock, Clr, Up
  - Required output signal: address (7-bit) to access instruction memory
  - Notice here Clr is active high – from FSM
- IR is purely a group of flip-flops which will latch out the input signal
  - Required input signals: Clock, ld, instruction from instruction memory
  - Required output signal: instruction to the finite state machine



Combing PC+ROM+IR & Test!



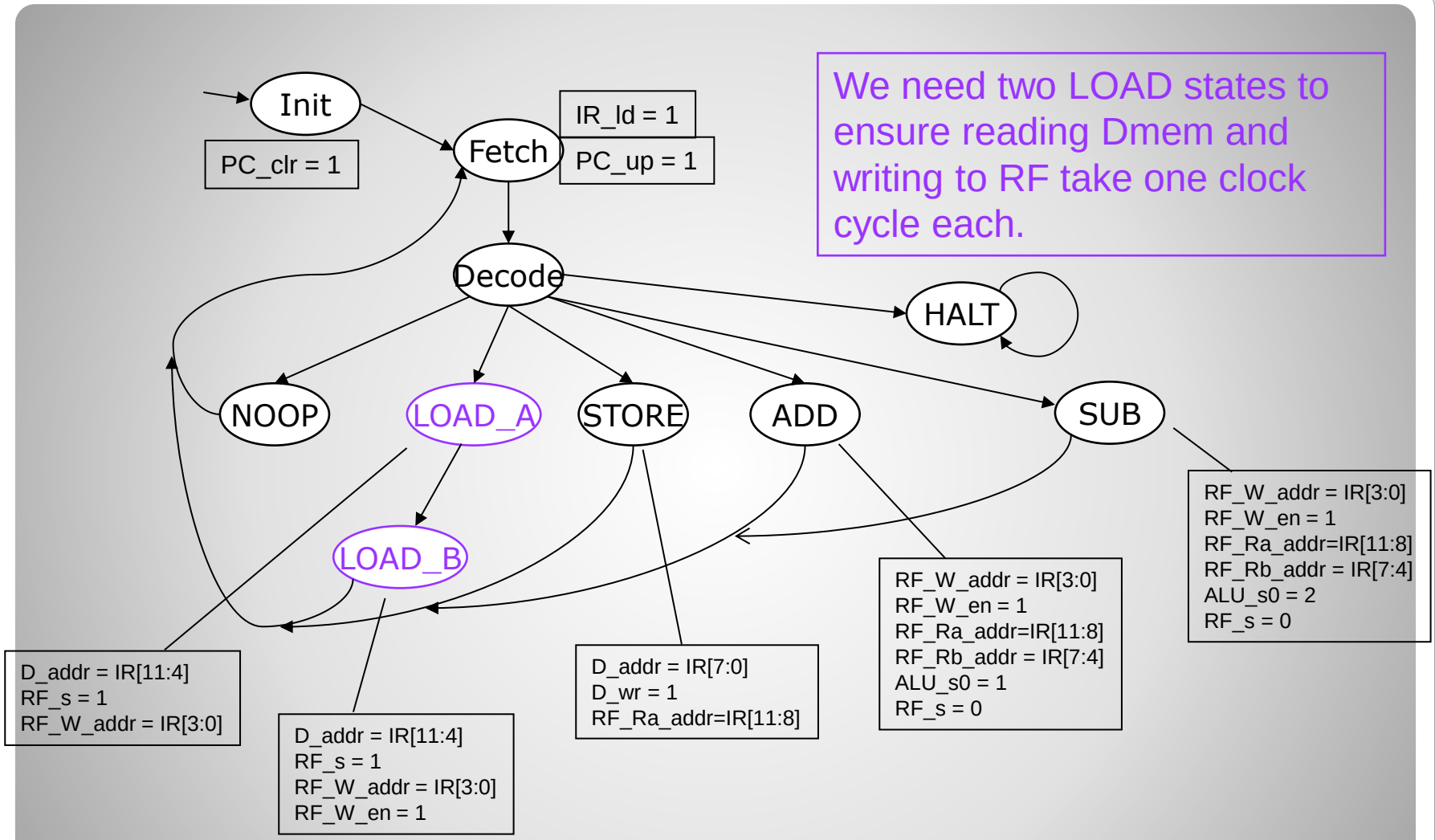
Along the Control Unit side ...



Control Lines

Control	Size
PC_clr	1
PC_up	1
IR_ld	1
D_addr	8
D_wr	1
RF_s	1
RF_W_addr	4
RF_W_en	1
RF_Ra_addr	4
RF_Rb_addr	4
Alu_s0	3

Original State Machine



Controller FSM (stick with this design)

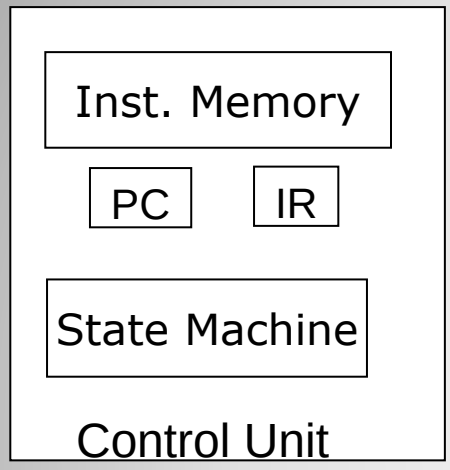
State Machine Output Definitions

Variable	Meaning
PC_clr	Program counter (PC) clear command
IR_ld	Instruction load command
PC_up	PC increment command
D_addr	Data memory address (8 bits)
D_wr	Data memory write enable
RF_s	Mux select line
RF_Ra_addr	Register file A-side read address (4 bits)
RF_Rb_addr	Register file B-side read address (4 bits)
RF_W_en	Register file write enable
RF_W_Addr	Register file write address (4 bits)
ALU_s0	ALU function select (3 bits)

State Machine State Outputs

State	Non-Zero Outputs
Init	PC_clr = 1
Fetch	IR_ld = 1, PC_up =1
Decode	Check the opcode
LoadA	D_addr = IR[11:4], RF_s = 1, RF_W_addr = IR[3:0]
LoadB	D_addr = IR[11:4], RF_s = 1, RF_W_addr = IR[3:0], RF_W_en = 1
Store	D_addr = IR[7:0], D_wr = 1, RF_Ra_addr = IR[11:8]
Add, Sub	RF_Ra_addr = IR[11:8], RF_Rb_addr = IR[7:4], RF_W_addr = IR[3:0], RF_W_en = 1, ALU_s0 = alu function (1 for Add or 2 for Sub), RF_s = 0
Halt	

How to test the State Machine?



Control Unit module should instantiate the following submodules:

```
PC.sv
IR.sv
InstMemory.v
FSM.sv
```

Required input signals: Clock, ResetN (Notice here the ResetN signal is required for FSM; it is active low!)

Required output signals: PC_Out, IR_Out, OutState, NextState, D_Addr, D_Wr,RF_s, RF_W_en, RF_Ra_Addr, RF_Rb_Addr, RF_W_Addr, ALU_s0

Build Control Unit Module and Test it!