Programmable Processors

Datapath operation

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 $d_7d_6d_5d_4d_3d_2d_1d_0$ $r_3r_2r_1r_0$**

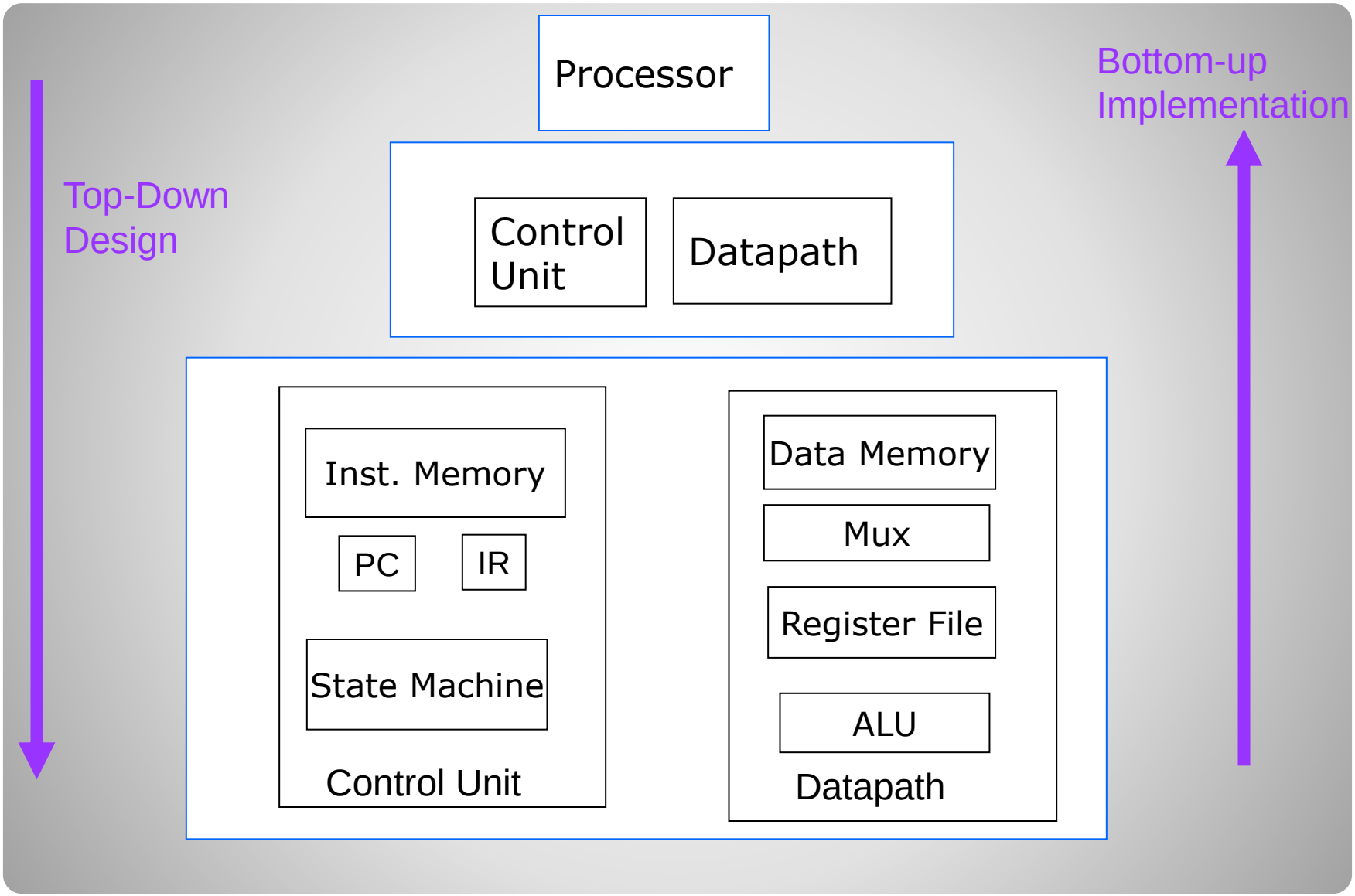
ADD instruction – **0011 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$**

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

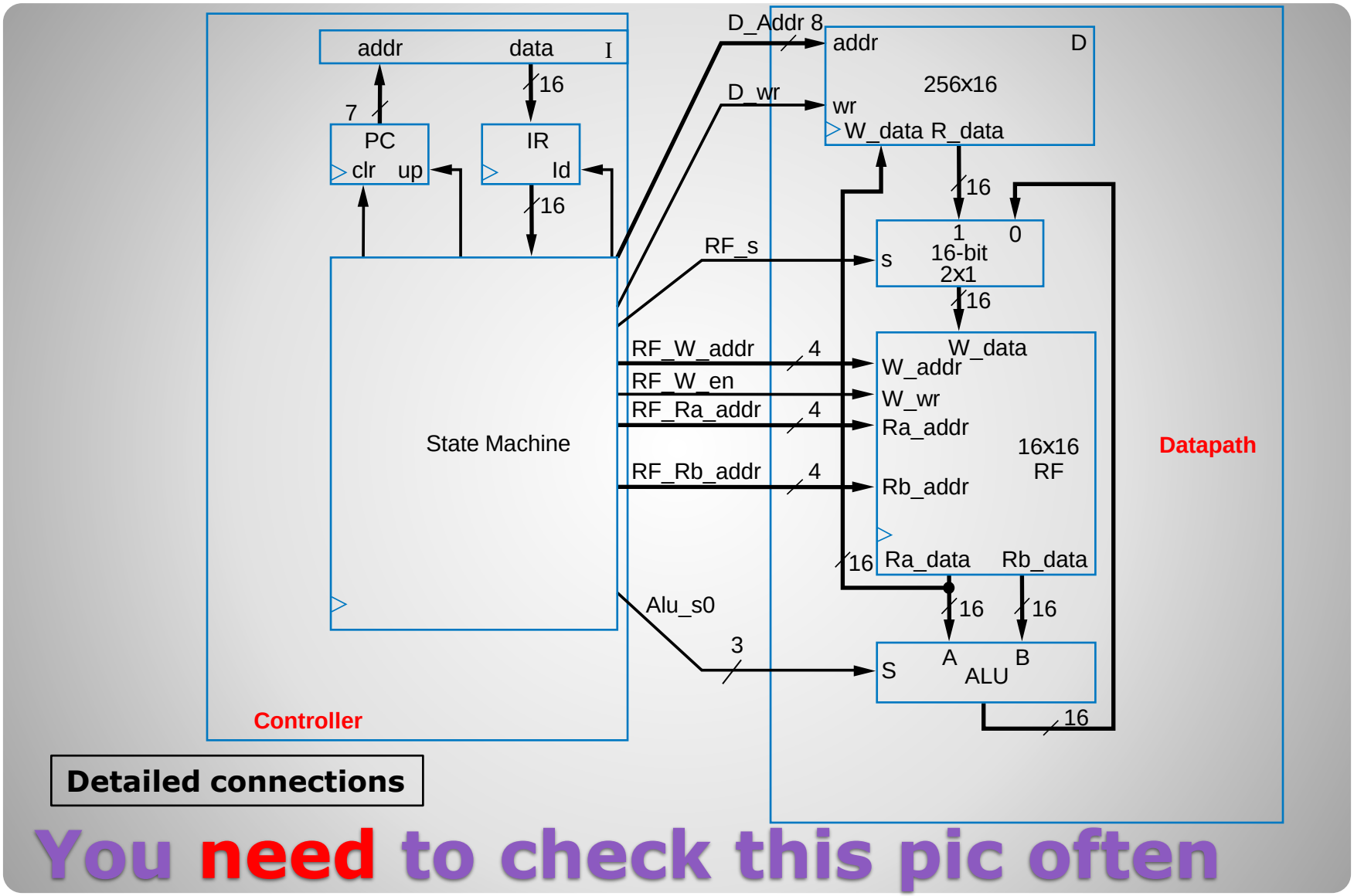
HALT instruction – **0101 0000 0000 0000**

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

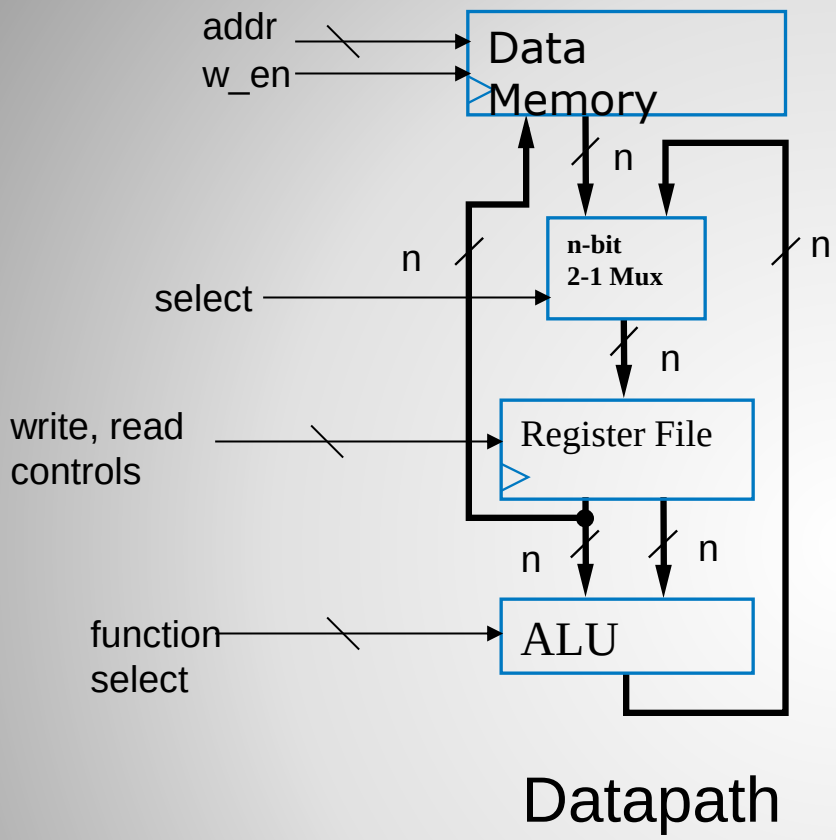
Refresh – 6-Instruction Processor



Pic of Processor Modules



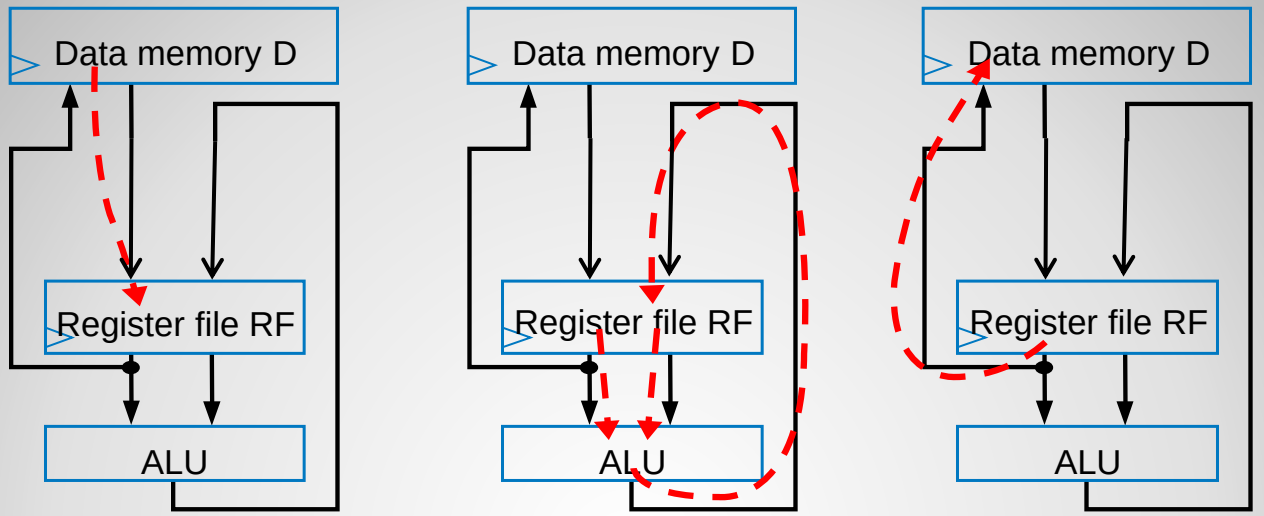
You need to check this pic often when connecting multi-modules



Datapath

- Takes data from Data Memory
- Stores it in Register File
- Operates on it in ALU (puts it back into the Register File)
- Can store Register File data back into Data Memory
- Note the dual output Register File

Datapath Side



LOAD operation

ALU operation (like an ADD)

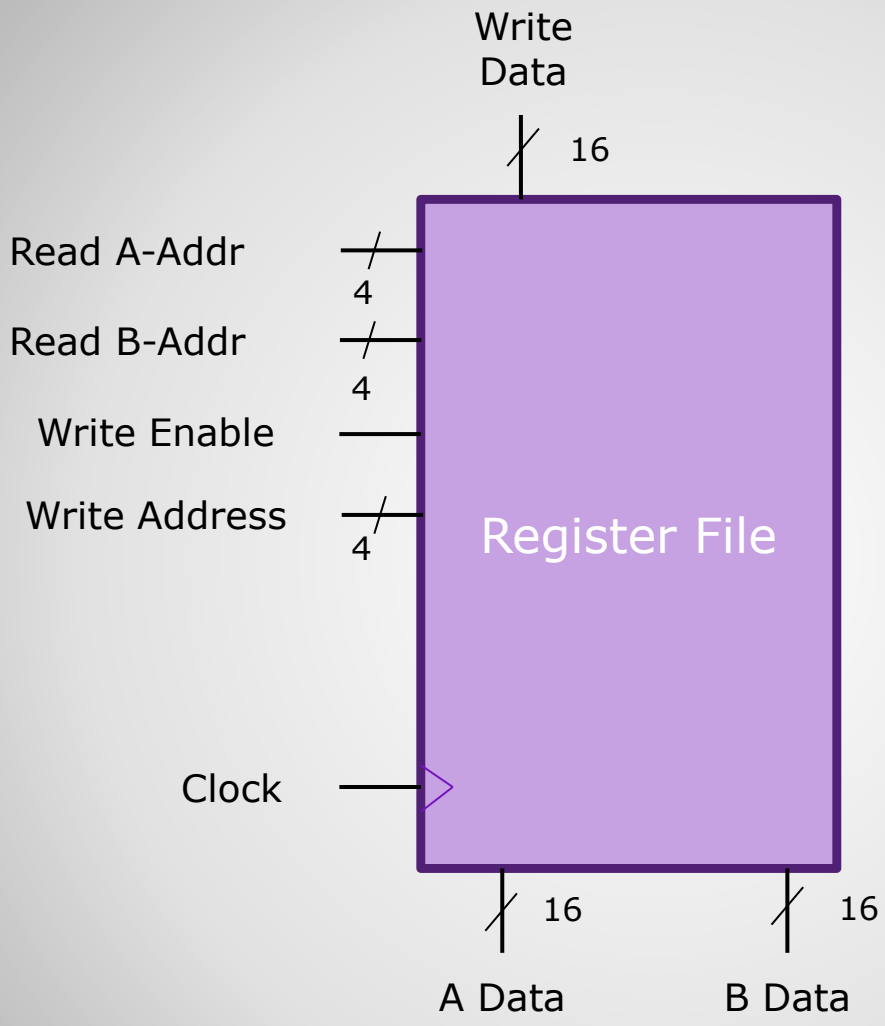
STORE operation

D[9] = D[0] + D[1] – requires a sequence of four datapath operations:

LOAD 0: RF[0] = D[0]
LOAD 1: RF[1] = D[1]
ADD 2: RF[2] = RF[0] + RF[1]
STORE 3: D[9] = RF[2]

Each line here represents one instruction; so it takes four instructions to do this add

Basic Datapath Operations



The Register File

```
// This is a Verilog description for an 8 x 16 register file
module regfile8x16a
   (input clk,                  // system clock
    input write,                // write enable
    input [2:0] wrAddr,         // write address
    input [15:0] wrData,        // write data
    input [2:0] rdAddrA,        // A-side read address
    output [15:0] rdDataA,      // A-side read data
    input [2:0] rdAddrB,        // B-side read address
    output [15:0] rdDataB );    // B-sdie read data

    logic [15:0] regfile [0:7];  // 8 registers, each 16-bit

    // read the registers
     …….                        // put your code here


    // write the registers
     …….                        // put your code here

endmodule
```

Register File (for reference)

```
module registerfile (Read1,Read2,WriteReg,WriteData,RegWrite,
Data1,Data2,clock);

   input [5:0] Read1,Read2,WriteReg; // the register numbers
to read or write
   input [31:0] WriteData; // data to write
   input RegWrite, // the write control
     clock; // the clock to trigger write
   output [31:0] Data1, Data2; // the register values read
   reg [31:0] RF [31:0]; // 32 registers each 32 bits long

   assign Data1 = RF[Read1];
   assign Data2 = RF[Read2];

   always begin
      // write the register with new value if Regwrite is
high
      @(posedge clock) if (RegWrite) RF[WriteReg] <=
WriteData;
   end
endmodule
```

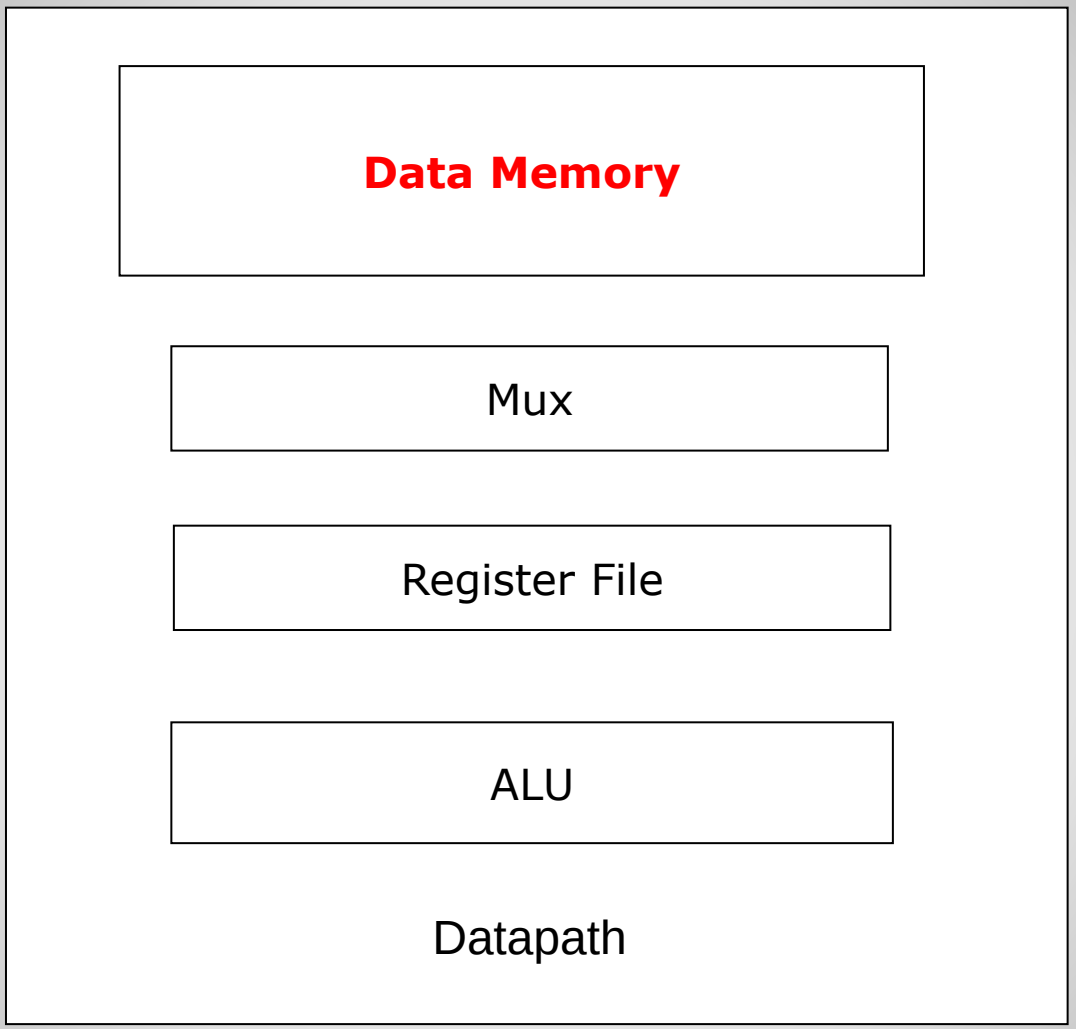
read the registers

Write to the registers

FIGURE B.8.11 A MIPS register file written in behavioral Verilog.
This register file writes on the rising clock edge.

A MIPS register file (372 Textbook)

How to test the register file?

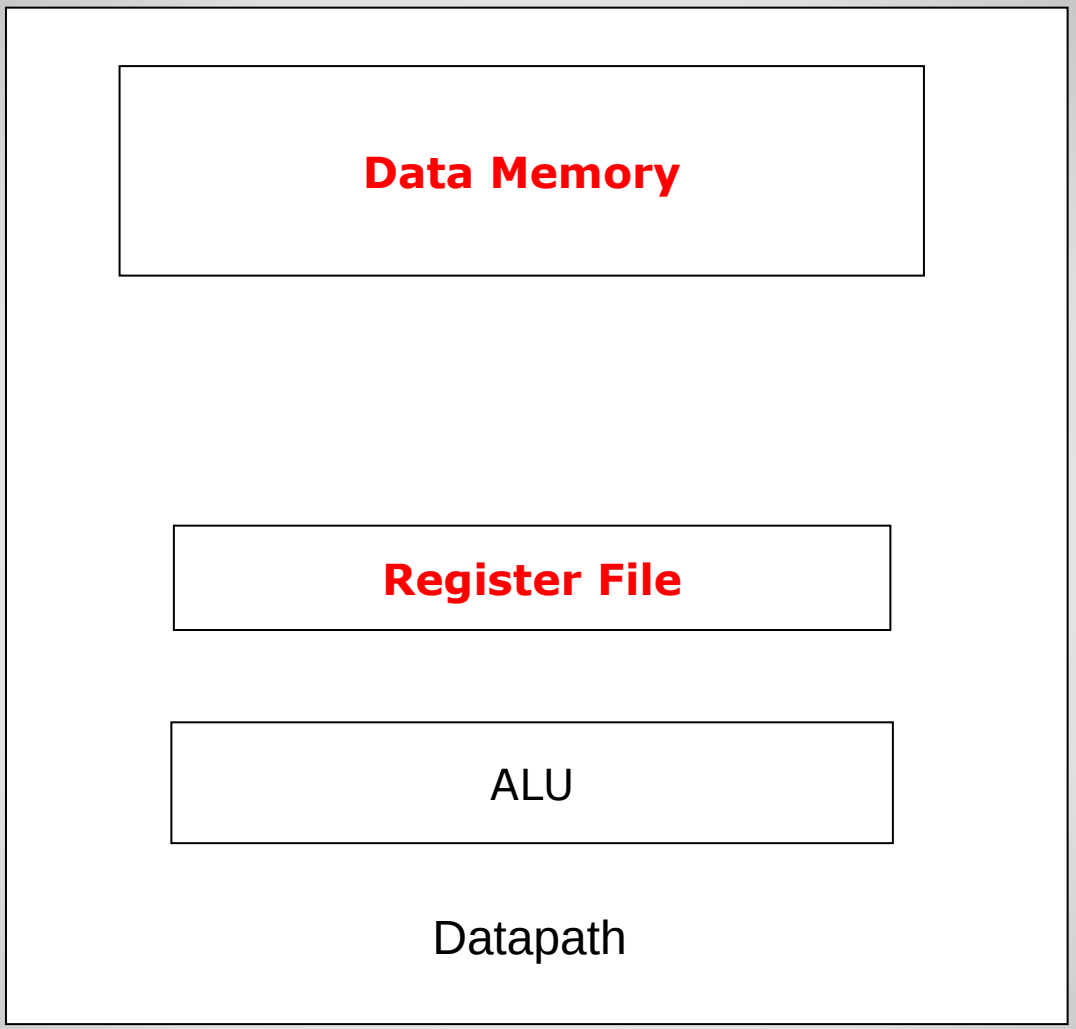


Along the Datapath side

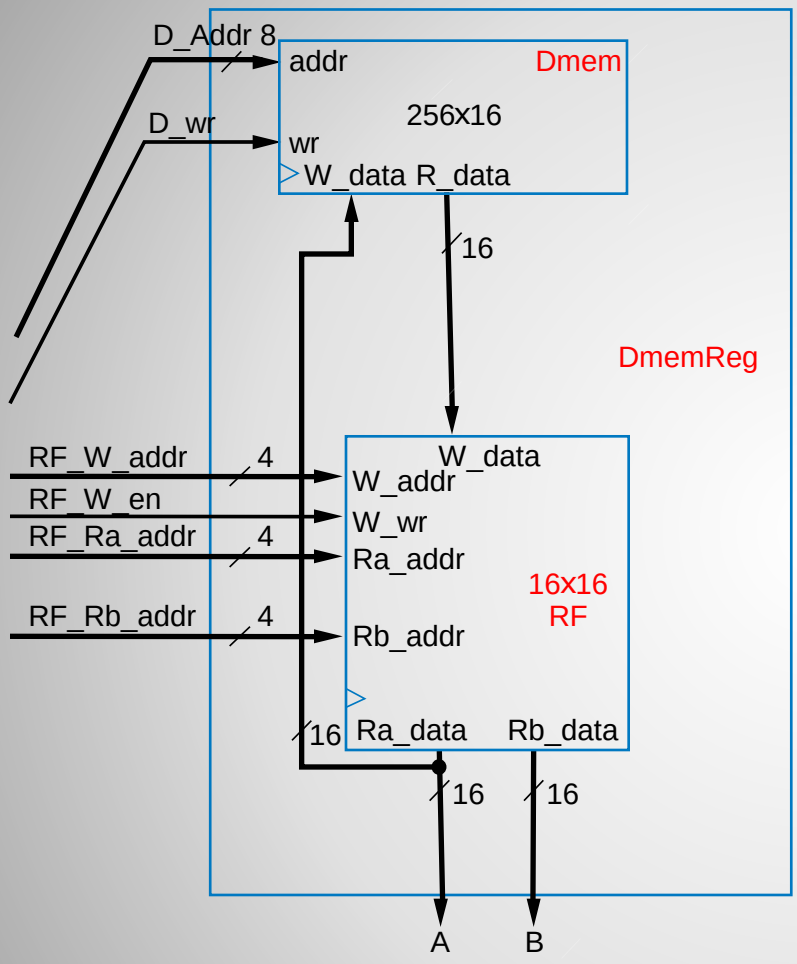
Data Memory Module

- Follow instructions:
  - Under Quartus, create a new .mif file, name it say, `D.mif`. Later manually type into specific locations numbers according to the given requirements.
  - Use Altera Library and build a 1-port RAM module, say, `DataMemory.v`. Make sure during the configuration procedure you have associated `DataMemory.v` with `D.mif`.

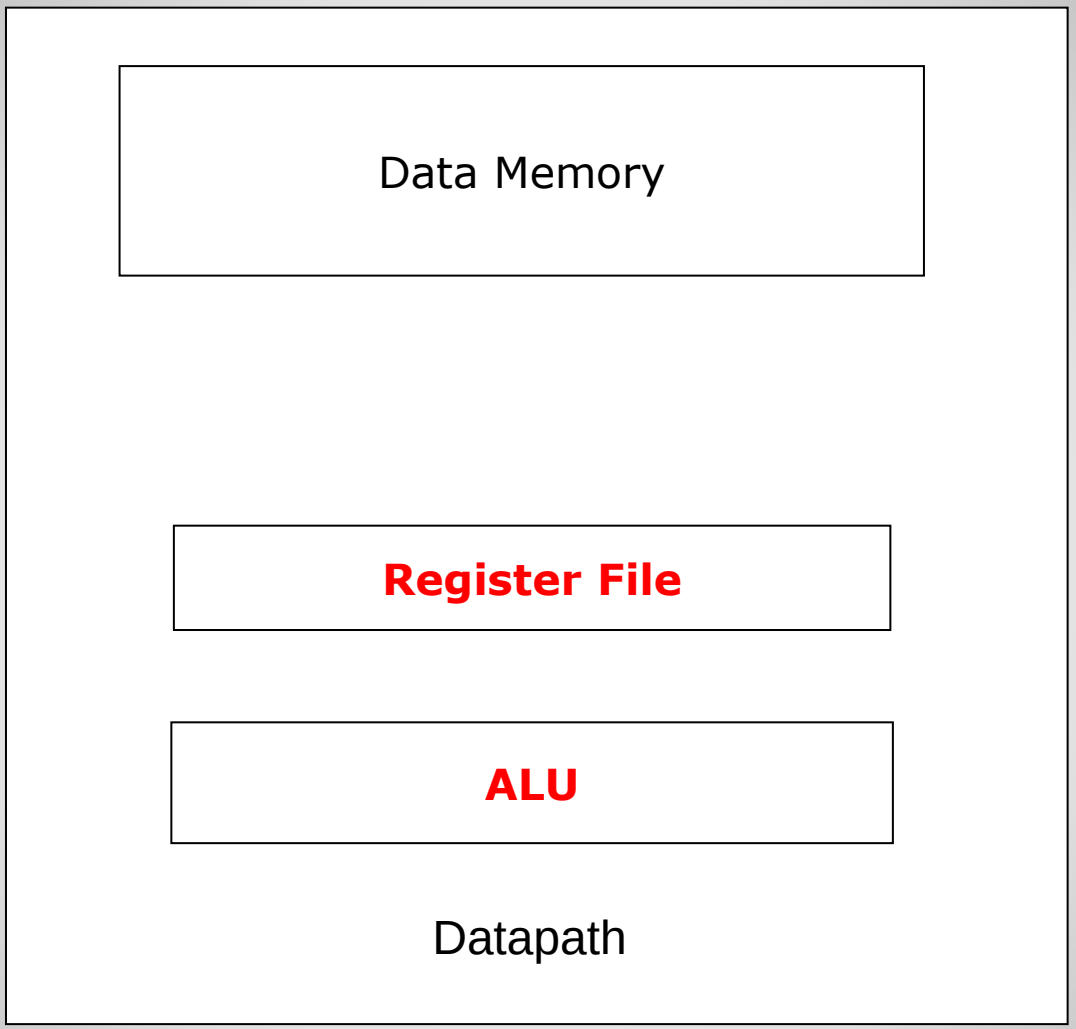
How to test the Data Memory?



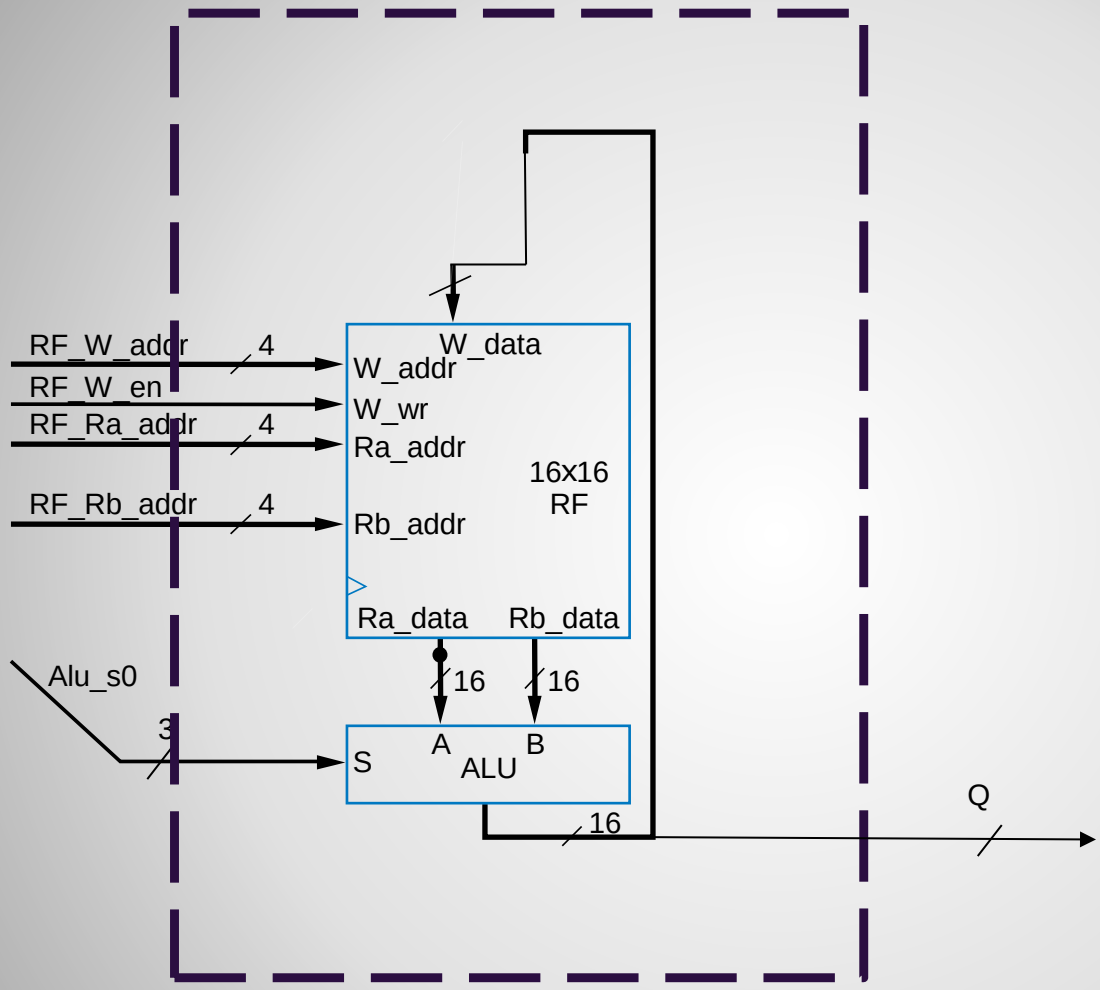
Connect Reg and Dmem & Test!



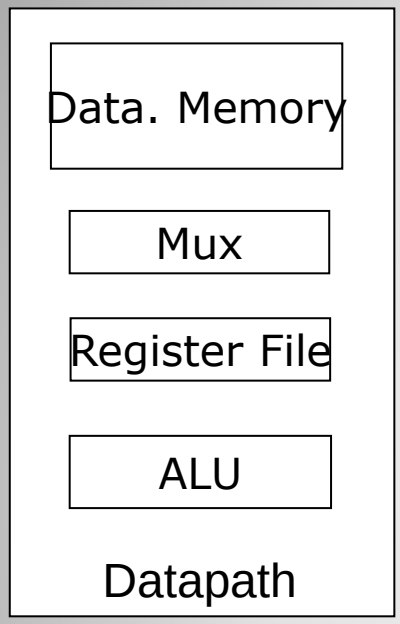
Design and Test of DmemReg



Connect Reg and ALU & Test!



Connect Reg and ALU & Test!



Datapath module should instantiate the following submodules:

```
DataMemory.v
Mux.sv
Register.sv
ALU.sv
```

Required input signals: Clock, D_Addr, D_Wr, RF_s, RF_W_Addr, RF_W_en, RF_Ra_Addr, RF_Rb_Addr, ALU_s0

Required output signals: ALU_inA, ALU_inB, ALU_out

Connect all 4 modules to build Datapath Module & Test!