به نام خدا

آزمایشگاه معماری کامپیوتر (CALab)

گزارش جلسه دوم

ایمان رسولی پرتو 810199425 & پارسا حداد منفرد 810198380

ID Stage

Register File ←

ماژول Reg_File دارای 15 رجیستر همه منظوره هست که برای خوندن و نوشتن استفاده میشه که عملیات نوشتن با negedge clk انجام میشه. کد این بخش به شکل زیر خواهد بود:

```
module RegisterFile(clk, rst, src1, src2, WB_DEST, WB_Value, WB_WB_EN, reg1, reg2);
  input clk, rst;
  input [3:0] src1, src2, WB_DEST;
  input [31:0] WB_Value;
  input WB_WB_EN;
  output reg [31:0] reg1, reg2;

  reg [31:0] registerfile[14:0];

  assign reg1 = registerfile[src1];
  assign reg2 = registerfile[src2];

  always @(posedge clk)begin
    if(rst)begin
      registerfile[0] = 32'b0000_0000_0000_0000_0000_0000_0000_0000;
      registerfile[1] = 32'b0000_0000_0000_0000_0000_0000_0000_0001;
      registerfile[2] = 32'b0000_0000_0000_0000_0000_0000_0000_0010;
      registerfile[3] = 32'b0000_0000_0000_0000_0000_0000_0000_0011;
      registerfile[4] = 32'b0000_0000_0000_0000_0000_0000_0000_0100;
      registerfile[5] = 32'b0000_0000_0000_0000_0000_0000_0000_0101;
      registerfile[6] = 32'b0000_0000_0000_0000_0000_0000_0000_0110;
      registerfile[7] = 32'b0000_0000_0000_0000_0000_0000_0000_0111;
      registerfile[8] = 32'b0000_0000_0000_0000_0000_0000_0000_1000;
      registerfile[9] = 32'b0000_0000_0000_0000_0000_0000_0000_1001;
      registerfile[10] = 32'b0000_0000_0000_0000_0000_0000_0000_1010;
      registerfile[11] = 32'b0000_0000_0000_0000_0000_0000_0000_1011;
      registerfile[12] = 32'b0000_0000_0000_0000_0000_0000_0000_1100;
      registerfile[13] = 32'b0000_0000_0000_0000_0000_0000_0000_1101;
      registerfile[14] = 32'b0000_0000_0000_0000_0000_0000_0000_1110;
    end
  end

  always @(negedge clk)begin
    if(rst)
      registerfile[WB_DEST] = registerfile[WB_DEST];
    else
      registerfile[WB_DEST] = WB_WB_EN ? WB_Value : registerfile[WB_DEST];
  end

endmodule
```

Fig1. Reg_File Verilog description

Control Unit ←

کنترلر مجموعه با توجه به mode و opcode دستور عملیات لازم برای تولید سیگنال‌های کنترلی رو انجام میده.

```
module Control_Unit(mode, opcode, S_in, EXE_CMD, MEM_R_EN, MEM_W_EN, WB_EN, B, S_out);
  input [1:0] mode;
  input [3:0] opcode;
  input S_in;
  output reg [3:0] EXE_CMD;
  output reg MEM_R_EN, MEM_W_EN, WB_EN, B, S_out;

  always@(opcode)begin
    {EXE_CMD, MEM_R_EN, MEM_W_EN, WB_EN, B, S_out} = 8'b0;
    if(mode == 2'b00)begin
      case(opcode)
        4'b0000: EXE_CMD = 4'b0000;
        4'b1101: EXE_CMD = 4'b0001;
        4'b1111: EXE_CMD = 4'b1001;
        4'b0100: EXE_CMD = 4'b0010;
        4'b0101: EXE_CMD = 4'b0011;
        4'b0010: EXE_CMD = 4'b0100;
        4'b0110: EXE_CMD = 4'b0101;
        4'b0000: EXE_CMD = 4'b0110;
        4'b1100: EXE_CMD = 4'b0111;
        4'b0001: EXE_CMD = 4'b1000;
        4'b1010: EXE_CMD = 4'b0100;
        4'b1000: EXE_CMD = 4'b0110;
        default: EXE_CMD = 4'b0000;
      endcase
    end
    else if(mode == 2'b01)begin
      if(opcode == 4'b0100)begin
        EXE_CMD = 4'b0010;
        if(S_in == 1'b1)begin
            S_out = 1'b1;
            MEM_R_EN = 1'b1;
        end
        else begin
            MEM_W_EN = 1;
        end
      end
    end
    else if(mode == 2'b10)
        B = 1'b1;
  end

endmodule
```

Fig2. Controller Verilog description

Check Condition ←

این ماژول با توجه به شرطی که از دستور میگیره (cond) و status register برقرار بودن شرط دستور رو بررسی میکنه؛ اگر شرط برقرار نباشه دستور اجرا نخواهد شد.

```
module Condition_Check(cond, flag, status);
  input [3:0] cond, flag;
  output reg status;

  always@(cond)begin
    case(cond)
      4'b0000: status = flag[2] ? 1 : 0;
      4'b0001: status = ~flag[2] ? 1 : 0;
      4'b0010: status = flag[1] ? 1 : 0;
      4'b0011: status = ~flag[1] ? 1 : 0;
      4'b0100: status = flag[3] ? 1 : 0;
      4'b0101: status = ~flag[1] ? 1 : 0;
      4'b0110: status = flag[0] ? 1 : 0;
      4'b0111: status = ~flag[0] ? 1 : 0;
      4'b1000: status = (flag[1] == 1 && flag[2] == 0) ? 1 : 0;
      4'b1001: status = (flag[1] == 0 && flag[2] == 1) ? 1 : 0;
      4'b1010: status = ~(flag[2] ^ flag[0]) ? 1 : 0;
      4'b1011: status = (flag[2] ^ flag[0]) ? 1 : 0;
      4'b1100: status = ((flag[2] == 0 && flag[3] == 1 && flag[0] == 1) ||
                         flag[3] == 0 && flag[0] == 0) ? 1 : 0;
      4'b1101: status = ((flag[2] == 1 && flag[3] == 1 && flag[0] == 0) ||
                         flag[3] == 0 && flag[0] == 1) ? 1 : 0;
      4'b1110: status = 1'b1;
    endcase
  end

endmodule
```

Fig3. Check Condition module Verilog description

← وصل کردن module های ساخته شده و طراحی بخش ID stage

سه ماژول ساخته شده در بخش ID بهم متصل میشن. در کنار اونها mux های لازم، or gate ها و not gate های لازم هم به شکل implement behavioral خواهند شد.

```
module ID_stage(clk, rst, Instruction, WB_Value, WB_WB_EN, WB_DEST, hazard, SR,
        WB_EN, MEM_R_EN, MEM_W_EN, B, S, EXE_CMD, Val_Rn, Val_Rm, imm,
        Shift_operand, Signed_imm_24, Dest, src1, src2, Two_src);
  input clk, rst;
  input [31:0] Instruction;
  input [31:0] WB_Value;
  input WB_WB_EN;
  input [3:0] WB_DEST;
  input hazard;
  input [3:0] SR;

  output WB_EN, MEM_R_EN, MEM_W_EN, B, S;
  output [3:0] EXE_CMD;
  output [31:0] Val_Rn, Val_Rm;
  output imm;
  output [11:0] Shift_operand;
  output [23:0] Signed_imm_24;
  output [3:0] Dest;
  output [3:0] src1, src2;
  output Two_src;

  wire wb_en, mem_read, mem_write, b, s_in;
  wire [3:0] exe_cmd;

  wire [3:0] opcode, cond, dest;
  wire [1:0] mode;
  wire s_out, cond_check_out, ctrl_out_sel;

  assign opcode = Instruction[24:21];
  assign cond = Instruction[31:28];
  assign mode = Instruction[27:26];
  assign s_in = Instruction[20];

  assign src1 = Instruction[19:16];
  assign src2 = (mem_read) ? Instruction[15:12] : Instruction[3:0];

  assign ctrl_out_sel = !cond_check_out || hazard;

  RegisterFile RF(clk, rst, src1, src2, WB_DEST, WB_Value, WB_WB_EN, Val_Rn, Val_Rm);

  Condition_Check Cond_Check(cond, SR, cond_check_out);

  Control_Unit CU(mode, opcode, s_in, exe_cmd, mem_read, mem_write, wb_en, b, s_out);

  assign imm = Instruction[25];
  assign Dest = Instruction[15:12];
  assign Signed_imm_24 = Instruction[23:0];
  assign Shift_operand = Instruction[11:0];

  assign WB_EN = ctrl_out_sel ? 1'b0 : wb_en;
  assign MEM_R_EN = ctrl_out_sel ? 1'b0 : mem_read;
  assign MEM_W_EN = ctrl_out_sel ? 1'b0 : mem_write;
  assign EXE_CMD = ctrl_out_sel ? 4'b0 : exe_cmd;
  assign B = ctrl_out_sel ? 1'b0 : b;
  assign S = ctrl_out_sel ? 1'b0 : s_out;

  assign Two_src = !imm || mem_write;

endmodule
```

Fig4. ID stage Verilog description

برای قابل فهم تر شدن برنامه از wire های اضافی نظیر opcode, cond استفاده کردیم و برای هنگامی که شرط دستور برقرار نباشد، از wire های کمکی wb_en, mem_read, mem_write, … استفاده کردیم.

← طراحی ID stage register

این رجیستر PC رو از IF stage rgister دریافت میکنه، همچنین سیگنال های مورد نیاز رو در صورت برقرار بودن شرط رجیستر میکنه و اونها رو برای بخش EXE آماده میکنه.

```
85  module ID_stage_reg(clk, rst, flush, WB_EN_IN, MEM_R_EN_IN, MEM_W_EN_IN, B_IN, S_IN,
86                      EXE_CMD_IN, PC_IN, Val_Rn_IN, Val_Rm_IN, imm_IN,Shift_operand_IN,
87                      Signed_imm_24_IN, Dest_IN, WB_EN, MEM_R_EN, MEM_W_EN, B, S, EXE_CMD,
88                      PC, Val_Rn, Val_Rm, imm, Shift_operand, Signed_imm_24, Dest);
89    input clk, rst, flush;
90    input WB_EN_IN, MEM_R_EN_IN, MEM_W_EN_IN, B_IN, S_IN;
91    input [3:0] EXE_CMD_IN;
92    input [31:0] PC_IN;
93    input [31:0] Val_Rn_IN, Val_Rm_IN;
94    input imm_IN;
95    input [11:0] Shift_operand_IN;
96    input [23:0] Signed_imm_24_IN;
97    input [3:0] Dest_IN;
98
99    output reg WB_EN, MEM_R_EN, MEM_W_EN, B, S;
100   output reg [3:0] EXE_CMD;
101   output reg [31:0] PC;
102   output reg [31:0] Val_Rn, Val_Rm;
103   output reg imm;
104   output reg [11:0] Shift_operand;
105   output reg [23:0] Signed_imm_24;
106   output reg [3:0] Dest;
107
108   always@(posedge clk, posedge rst)begin
109     if(rst) begin
110       WB_EN <= 1'b0;
111       MEM_R_EN <= 1'b0;
112       MEM_W_EN <= 1'b0;
113       B <= 1'b0;
114       S <= 1'b0;
115       EXE_CMD <= 4'b0;
116       PC <= 32'b0;
117       Val_Rn <= 32'b0;
118       Val_Rm <= 32'b0;
119       imm <= 1'b0;
120       Shift_operand <= 12'b0;
121       Signed_imm_24 <= 24'b0;
122       Dest <= 4'b0;
123     end
124     else begin
125       WB_EN <= WB_EN_IN;
126       MEM_R_EN <= MEM_R_EN_IN;
127       MEM_W_EN <= MEM_W_EN_IN;
128       B <= B_IN;
129       S <= S_IN;
130       EXE_CMD <= EXE_CMD_IN;
131       PC <= PC_IN;
132       Val_Rn <= Val_Rn_IN;
133       Val_Rm <= Val_Rm_IN;
134       imm <= imm_IN;
135       Shift_operand <= Shift_operand_IN;
136       Signed_imm_24 <= Signed_imm_24_IN;
137       Dest <= Dest_IN;
138     end
139   end
140
141 endmodule
```

Fig5. ID stage register Verilog description

← متصل کردن بخش های IF, ID و تست بخشی از benchmark

ماژول های IF, ID به همراه رجیسترهاشون رو بهم متصل میکنیم.

```
module ARM_ID(clk, rst, WB_Value, WB_WB_EN, WB_DEST, SR);
  input clk, rst;
  input [31:0] WB_Value;
  input WB_WB_EN;
  input [3:0] WB_DEST;
  wire hazard;
  input [3:0] SR;

  wire freeze, Branch_taken;
  wire [31:0] BranchAddr, IF_PC, IF_Instruction;

  assign freeze = 1'b0;
  assign Brach_taken = 1'b0;
  assign BranchAddr = 32'b0;

  wire flush;
  wire [31:0] IF_Reg_PC, IF_Reg_Instruction;

  assign flush = 1'b0;
  assign hazard = 1'b0;

  wire WB_EN_IN, MEM_R_EN_IN, MEM_W_EN_IN, B_IN, S_IN;
  wire [3:0] EXE_CMD_IN;
  wire [31:0] Val_Rn_IN, Val_Rm_IN;
  wire imm_IN;
  wire [11:0] Shift_operand_IN;
  wire [23:0] Signed_imm_24_IN;
  wire [3:0] Dest_IN;
  wire [3:0] src1, src2;
  wire Two_src;

  wire WB_EN, MEM_R_EN, MEM_W_EN, B, S;
  wire [3:0] EXE_CMD;
  wire [31:0] Val_Rn, Val_Rm;
  wire imm;
  wire [11:0] Shift_operand;
  wire [23:0] Signed_imm_24;
  wire [3:0] Dest;

  IF_stage IF(clk, rst, freeze, Branch_taken, BranchAddr, IF_PC, IF_Instruction);

  IF_stage_reg IF_Reg(clk, rst, freeze, flush, IF_PC, IF_Instruction, IF_Reg_PC, IF_Reg_Istruction);

  ID_stage ID(clk, rst, IF_Reg_Instruction, WB_Value, WB_WB_EN, WB_DEST, hazard, SR,
          WB_EN_IN, MEM_R_EN_IN, MEM_W_EN_IN, B_IN, S_IN, EXE_CMD_IN, Val_Rn_IN,
          Val_Rm_IN, imm_IN, Shift_operand_IN, Signed_imm_24_IN, Dest_IN, src1, src2, Two_src);

  ID_stage_reg ID_Reg(clk, rst, flush, WB_EN_IN, MEM_R_EN_IN, MEM_W_EN_IN, B_IN, S_IN,
            EXE_CMD_IN, PC_IN, Val_Rn_IN, Val_Rm_IN, imm_IN, Shift_operand_IN,
            Signed_imm_24_IN, Dest_IN, WB_EN, MEM_R_EN, MEM_W_EN, B, S, EXE_CMD,
            PC, Val_Rn, Val_Rm, imm, Shift_operand, Signed_imm_24, Dest);
endmodule
```

Fig6. Connect IF & ID with registers

برای سیگنال هایی که هنوز پیاده سازی نشدن دو راه داریم؛ سیگنال‌هایی نظیر flush, hazard که deactive شون میکنیم و سیگنال هایی مثل SR, WB_WB_EN که برای verify کردن عملکرد مدار اونها رو به عنوان input در نظر میگیریم و در testbench به شکل دستی اونها رو مقداردهی میکنیم.

پس از اتصال ماژول ها یک testbench مینویسیم و از ARM Top module، instance میگیریم.

```
module ARM_Tb();
  reg clk, rst;
  reg [31:0] WB_Value;
  reg WB_WB_EN;
  reg [3:0] WB_DEST;
  reg [3:0] SR;

  ARM_ID ARM(clk, rst, WB_Value, WB_WB_EN, WB_DEST, SR);

  initial begin
    clk = 1'b0;
    rst = 1'b0;
    WB_Value = 32'b0;
    WB_WB_EN = 1'b0;
    WB_DEST = 4'b0;
    SR = 4'b0;
  end

  always #5 clk = ~clk;

  initial begin
    #10 rst = 1'b1;
    #20 rst = 1'b0;
    #500 $stop;
  end
endmodule
```

Fig7. ARM testbench

میبینیم که سیگنال های اشاره شده به شکل reg هستند و به شکل دستی مقداردهی میشن؛ 18 دستور اول benchmark رو در حافظه دستور قرار میدیم و پردازنده رو شبیه‌سازی میکنیم.

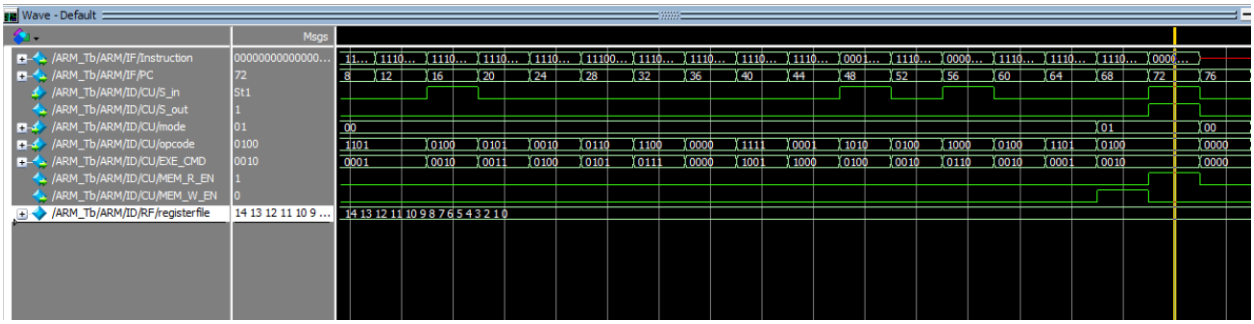

Fig9. Benchmark test on ARM

با توجه به شکل 9 میبینیم مقادیر رجیسترهای Reg_File مطابق عدد رجیستر هست؛ وقتی PC = 72 هست یعنی 18 instruction ام اجرا میشه (چون PC از 0 شروع میشه)، سیگنال mem_read = 1 هست و در بقیه موارد صفره؛ این بررسی‌ها نشون میده بخش ID پردازنده ARM به درستی طراحی شده.