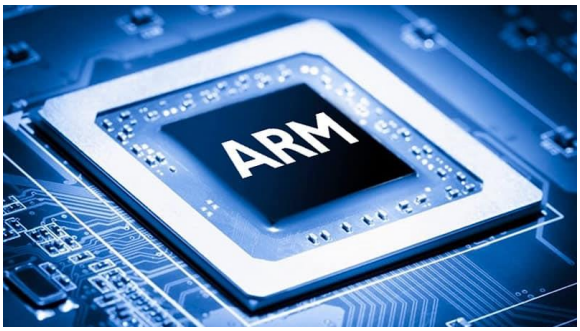
به نام خدا

دانشگاه تهران
پردیس دانشکده‌های فنی
دانشکده برق و کامپیوتر



آزمایشگاه معماری کامپیوتر

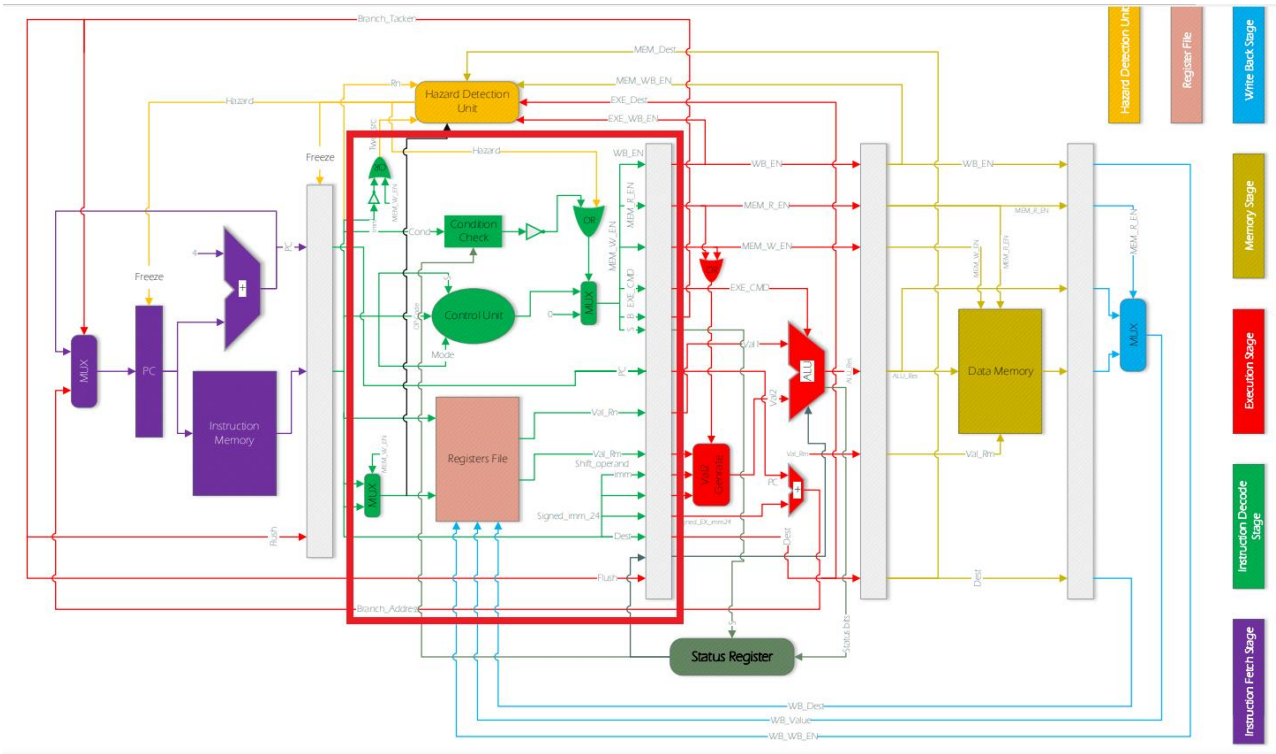
گزارش دستور کار شماره 2

محمد صالح عرفاتی
810197543
علی پادیاو
810199388

بهار 1402

مقدمه

در این جلسه بخش Instruction Decode را تکمیل می‌کنیم.



ماژول‌ها:

1) Register File:

کد:

```
module RegisterFile (
    input clk, rst,
    input [3:0] WB_Dest, Rs1, Rs2,
    input [31:0] WB_Value,
    input WB_WB_EN,
    output [31:0] Val_Rn, Val_Rm
  );

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

  integer i;

  initial
  begin
    for (i = 0; i < 15 ; i = i + 1 )
      RegFile[i] = i;
  end

  assign Val_Rn = RegFile[Rs1];
  assign Val_Rm = RegFile[Rs2];

  always @(negedge clk, posedge rst)
  begin
    if (rst)
      for (i = 0; i < 15 ; i = i + 1 )
        RegFile[i] <= i;
    else if (WB_WB_EN)
      RegFile[WB_Dest] <= WB_Value;
  end

endmodule

```

این فایل 15 رجیستر دارد که مقدار هرکدام برابر اندیسش است. به صورت همزمان و async میشود از 2 رجیستر خواند و به صورت sync بر روی یک رجیستر نوشت.

2) Control Unit:

کد:

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

`define MOV 4'b1101
`define MVN 4'b1111
`define ADD 4'b0100
`define ADC 4'b0101
`define SUB 4'b0010
`define SBC 4'b0110
`define AND 4'b0000
`define ORR 4'b1100
`define EOR 4'b0001
`define CMP 4'b1010
`define TST 4'b1000
`define LDR 4'b0100
`define STR 4'b0100

  always @(mode, opcode, S_in)
  begin
    {EXE_CMD, WB_EN, MEM_R_EN, MEM_W_EN, B, S_out} = 9'b0;

    case(mode)
      2'b00:
      begin
        S_out = S_in;
        case(opcode)
          `MOV:
          begin
            EXE_CMD = 4'b0001;
            WB_EN = 1'b1;
          end

          `MVN:
          begin
            EXE_CMD = 4'b1001;
            WB_EN = 1'b1;
          end

          `ADD:
          begin
            EXE_CMD = 4'b0010;
            WB_EN = 1'b1;
          end

          `ADC:
          begin
            EXE_CMD = 4'b0011;
            WB_EN = 1'b1;
          end

          `SUB:
          begin
            EXE_CMD = 4'b0100;
            WB_EN = 1'b1;
          end

          `SBC:
          begin
            EXE_CMD = 4'b0101;
            WB_EN = 1'b1;
          end

          `AND:
          begin
            EXE_CMD = 4'b0110;
            WB_EN = 1'b1;
          end

```

```

          `ORR:
          begin
            EXE_CMD = 4'b0111;
            WB_EN = 1'b1;
          end

          `EOR:
          begin
            EXE_CMD = 4'b1000;
            WB_EN = 1'b1;
          end

          `CMP:
          begin
            EXE_CMD = 4'b0100;
          end

          `TST:
          begin
            EXE_CMD = 4'b0110;
          end

          default:
            EXE_CMD = 4'b0000;
        endcase
      end


      2'b01:
      begin
        EXE_CMD = 4'b0010;
        MEM_R_EN = S_in;
        MEM_W_EN = !S_in;
        WB_EN = S_in;
      end


      2'b10:
      begin
        B = 1'b1;
      end

      default:;
    endcase

  end

endmodule

```

در این ماژول بر اساس mode و opcode سیگنال‌ها تنظیم میشوند. در تصویر مشخص است که سه دستور CMP، TSTو STR سیگنال WB_EN فعال نمی‌باشد. همینطور در تمام دستورات محاسباتی (mode=00) S_out برابر S_in می‌شود.

3) Condition Check:

کد:

```
module ConditionCheck (
    input [3:0] cond,
    Status_R,   // N Z C V
    output reg Is_Valid
  );

  always @ (cond, Status_R)
  case (cond)
    // Z set
    4'b0000:
      Is_Valid = Status_R[2];
    // Z clear
    4'b0001:
      Is_Valid = !Status_R[2];
    // C set
    4'b0010:
      Is_Valid = Status_R[1];
    // C clear
    4'b0011:
      Is_Valid = !Status_R[1];
    // N set
    4'b0100:
      Is_Valid = Status_R[3];
    // N clear
    4'b0101:
      Is_Valid = !Status_R[3];
    // V set
    4'b0110:
      Is_Valid = Status_R[0];
    // V clear
    4'b0111:
      Is_Valid = !Status_R[0];
    // C set and Z clear
    4'b1000:
      Is_Valid = Status_R[1] && !Status_R[2];
    // C clear or Z set
    4'b1001:
      Is_Valid = !Status_R[1] || Status_R[2];
    // N == V
    4'b1010:
      Is_Valid = Status_R[3] == Status_R[0];
    // N != V
    4'b1011:
      Is_Valid = Status_R[3] != Status_R[0];
    // Z==0, N==V
    4'b1100:
      Is_Valid = !Status_R[2] && (Status_R[3] == Status_R[0]);
    // Z==1, N!=V
    4'b1101:
      Is_Valid = Status_R[2] && (Status_R[3] != Status_R[0]);
    // Always
    4'b1110:
      Is_Valid = 1'b1;
    // Never
    4'b1111:
      Is_Valid = 1'b0;
  endcase

endmodule

```

این ماژول cond را از instruction می‌گیرد و با توجه به Status Register می‌بیند که آیا شرط آن دستور برقرار است یا نه.

Status Register شامل 4 بیت N Z C V است که مخفف negative zero carry overflow می‌باشد.

4) ID_Stage

```
wire [3:0] cond = Ins[31:28];
 wire [1:0] mode = Ins[27:26];
 wire I = Ins[25];
 wire [3:0] opcode = Ins[24:21];
 wire S_in = Ins[20];
 wire [3:0] Rn = Ins[19:16];
 wire [3:0] Rd = Ins[15:12];
 wire [3:0] Rm = MEM_W_EN ? Rd : Ins[3:0];
 assign shift_operand = Ins[11:0];
 assign signed_imm_24 = Ins[23:0];

 ConditionCheck CC (
                  .cond(cond),
                  .Status_R(Status_R),

                  .Is_Valid(Is_Valid)
                );

 RegisterFile RF (
                .clk(clk),
                .rst(rst),
                .Rs1(Rn),
                .Rs2(Rm),
                .WB_WB_EN(WB_WB_EN),
                .WB_Dest(WB_Dest),
                .WB_Value(WB_Value),

                .Val_Rn(Val_Rn),
                .Val_Rm(Val_Rm)
              );

 ControlUnit CU (
               .opcode(opcode),
               .mode(mode),
               .S_in(S_in),

               .EXE_CMD(EXE_CMD_CU),
               .WB_EN(WB_EN_CU),
               .MEM_R_EN(MEM_R_EN_CU),
               .MEM_W_EN(MEM_W_EN_CU),
               .B(B_CU),
               .S_out(S_CU)
             );

 assign imm = I;
 assign Dest = Rd;
 assign stop = hazard || !Is_Valid;
 assign {EXE_CMD, WB_EN, MEM_R_EN, MEM_W_EN, B, S} = stop ? 9'b0 : {EXE_CMD_CU, WB_EN_CU, MEM_R_EN_CU, MEM_W_EN_CU, B_CU, S_CU};
 assign Two_src = MEM_W_EN || !I;
```

ابتدا بخش‌های مختلف `Instruction` را مشخص کردیم. بعد از آن از ماژول های ساخته شده در `ID_Stage` اینستنس گرفتیم و درنهایت `output` های باقی‌مانده را `assign` کردیم.

5) ID_Reg

```
module IF_Reg (
    input clk, rst, freeze, flush,
    input [31:0] PC_in, instruction_in,
    output reg [31:0] PC_out, instruction_out
  );

  always @(posedge clk, posedge rst)
  begin
    if (rst)
    begin
      PC_out <= 0;
      instruction_out <= 0;
    end
    else if (flush)
    begin
      PC_out <= 0;
      instruction_out <= 0;
    end
    else if (~freeze)
    begin
      PC_out <= PC_in;
      instruction_out <= instruction_in;
    end
  end

endmodule

```

دستورات برنامه:

کد:

```
module Ins_Mem (
    input [31:0] in,
    output [31:0] out
  );

  reg [31:0] mem[31:0];

  initial
  begin
    mem[0] = 32'b1110_00_1_1101_0_0000_0000_000000010100;
    mem[1] = 32'b1110_00_1_1101_0_0000_0001_101000000001;
    mem[2] = 32'b1110_00_1_1101_0_0000_0010_000100000011;
    mem[3] = 32'b1110_00_0_0100_1_0010_0011_000000000010;
    mem[4] = 32'b1110_00_0_0101_0_0000_0100_000000000000;
    mem[5] = 32'b1110_00_0_0010_0_0100_0101_000100000100;
    mem[6] = 32'b1110_00_0_0110_0_0000_0110_000010100000;
    mem[7] = 32'b1110_00_0_1100_0_0101_0111_000101000010;
    mem[8] = 32'b1110_00_0_0000_0_0111_1000_000000000011;
    mem[9] = 32'b1110_00_0_1111_0_0000_1001_000000000110;
    mem[10] = 32'b1110_00_0_0001_0_0100_1010_000000000101;
    mem[11] = 32'b1110_00_0_1010_1_1000_0000_000000000110;
    mem[12] = 32'b0001_00_0_0100_0_0001_0001_000000000001;
    mem[13] = 32'b1110_00_0_1000_1_1001_0000_000000001000;
    mem[14] = 32'b0000_00_0_0100_0_0010_0010_000000000010;
    mem[15] = 32'b1110_00_1_1101_0_0000_0000_101100000001;
    mem[16] = 32'b1110_01_0_0100_0_0000_0001_000000000000;
    mem[17] = 32'b1110_01_0_0100_1_0000_1011_000000000000;
  end

  assign out = mem[in>>2];

endmodule
```

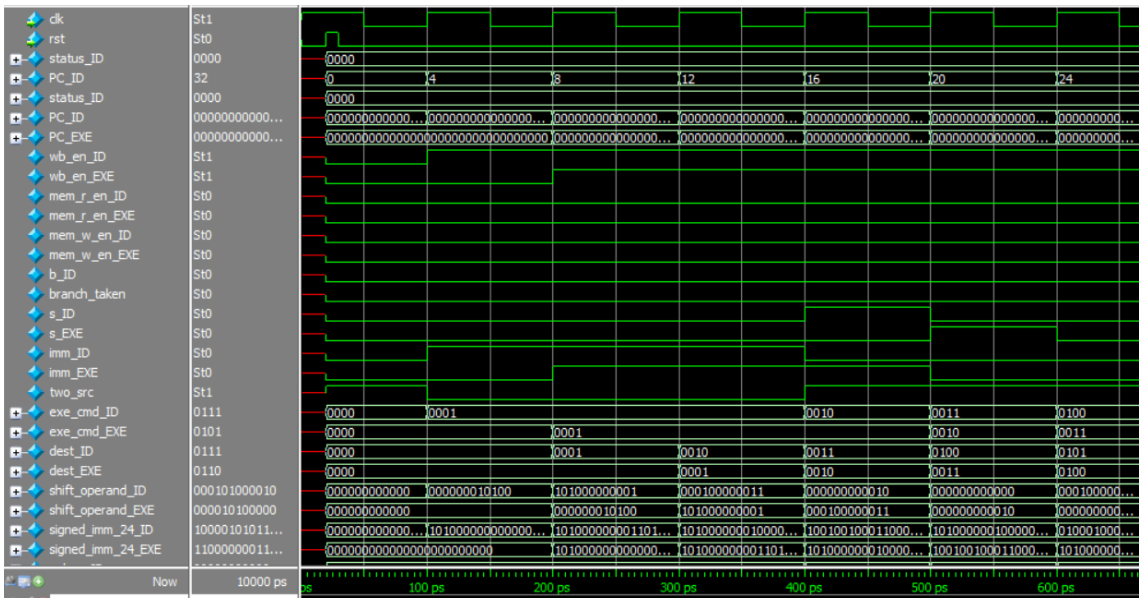
18 دستور اول برنامه محک را در Instruction Mem قرار دادیم.

SIMULATION:

برای چک کردن خروجی‌ها کد تست‌بنچ زیر را نوشتیم و در modelsim اجرا کردیم.

```
module TB_DE2 ();

  reg clk, rst;

  ARM arm (
        .clk(clk),
        .rst(rst)
      );

  initial
  begin
    clk = 1;
    repeat (200)
    begin
      #50;
      clk = ~clk;
    end
  end

  initial
  begin
    rst = 0;
    #20 rst = 1;
    #10 rst = 0;
  end

endmodule

```

ابتدا سیگنال‌های خروجی `ID Stage` و `ID Reg` را مقایسه میکنیم. همانطور که در تصویر مشخص است، خروجی‌های `ID Reg` یک کلاک از `ID Stage` عقب‌ترند.



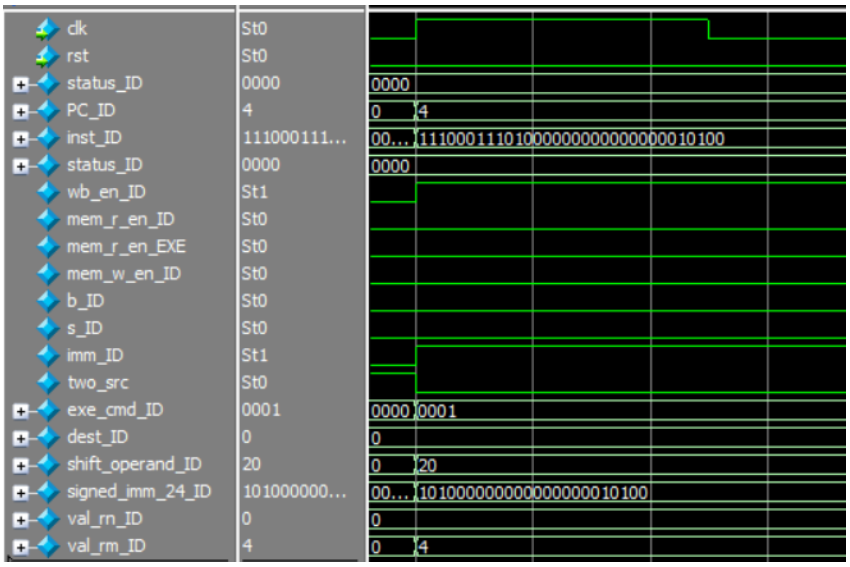
حالا 18 دستور اول برنامه محک را بررسی می‌کنیم.

دستور 1:

```
//MOV       R0, #20
1110_00_1_1101_0_0000_0000_000000010100
```

در این دستور wb_en_ID و imm_ID 1 و exe_cmd_ID 0001 میشوند که در تصویر مشخص است.

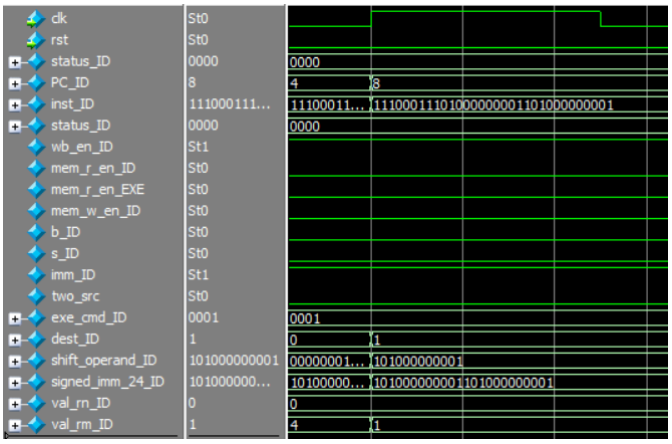
همچنین مقادیر shift_operand_ID، و Dest نیز با اینستراکشن‌ها همخوانی دارد.



دستور 2:

```
//MOV       R1, #4096

b1110_00_1_1101_0_0000_0001_101000000001
```

رجیستر مقصد و عدد قطعی تغییر کردند. عدد قطعی 101000000001 است که 8 بیت کم ارزش آن immed_8 و 4 بیت باقی‌مانده rotate_imm هستند. بنابراین عدد بالا برابر 4096 میباشد.

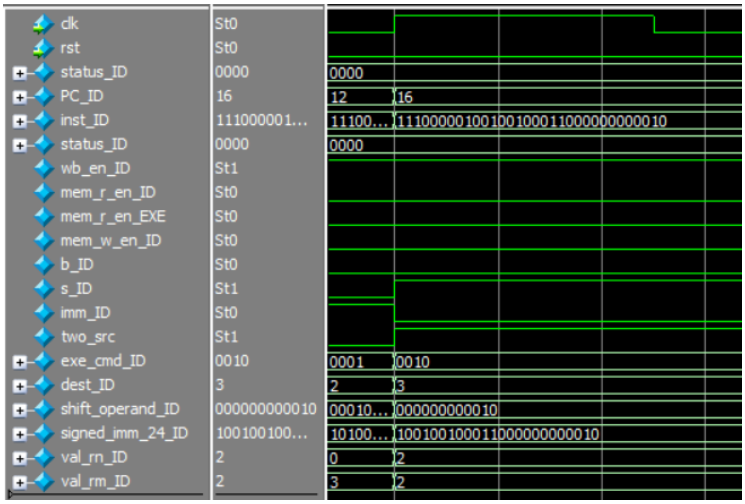


دستور 4:

```
//ADDS      R3,R2,R2

1110_00_0_0100_1_0010_0011_000000000010
```

بیت S در این دستور 1 شده و چون بیت imm 0 میشود، two_src از 0 به 1 تغییر می‌کند.

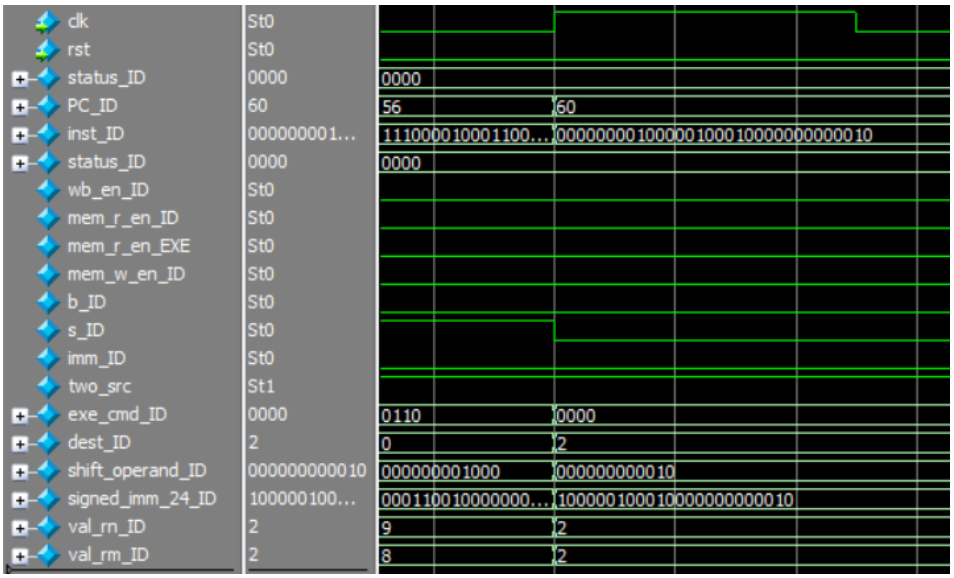
دستور Add است و command آن 0010 می‌باشد که در تصویر مشخص است. dest برابر 3 و Rn و Rm هر دو برابر 2 می‌باشند.



دستور 15:

```
//ADDEQ      R2,R2,R2

0000_00_0_0100_0_0010_0010_000000000010
```

Cond 0000 بیت Z را چک میکند. از آنجایی که Status Register ما در اینجای کار همواره 0 است، بنابراین شرط برقرار نمیشود و خروجی‌های کنترلر همه برابر 0 میشوند.

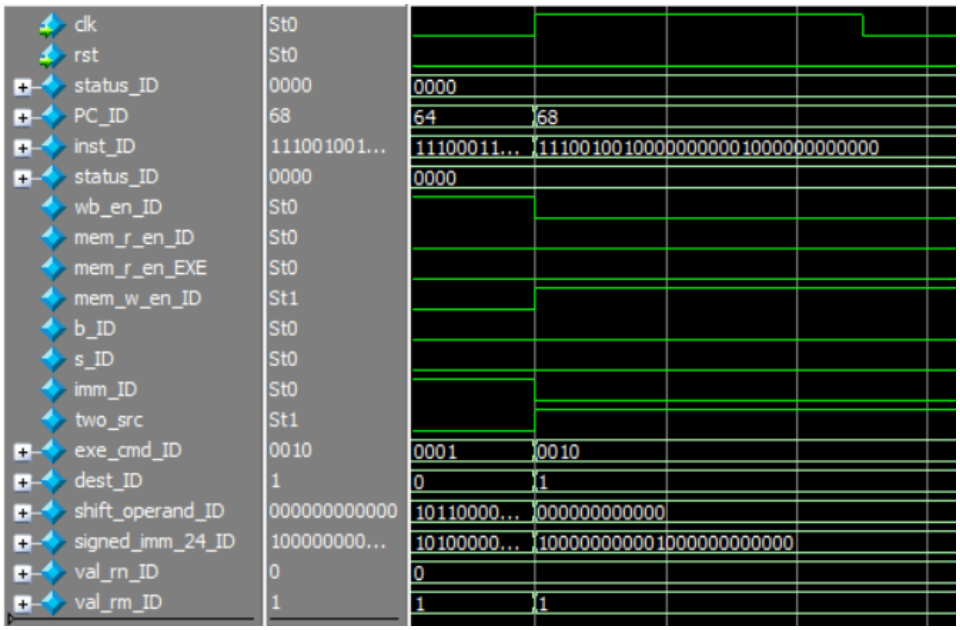


دستور 17:

```
//STR       R1,[R0],#0

1110_01_0_0100_0_0000_0001_000000000000
```

ازآنجایی که opcode دستور STR و LDR با دستور Add یکسان است، exe_cmd آنها نیز مثل ADD 0010 می‌باشد.

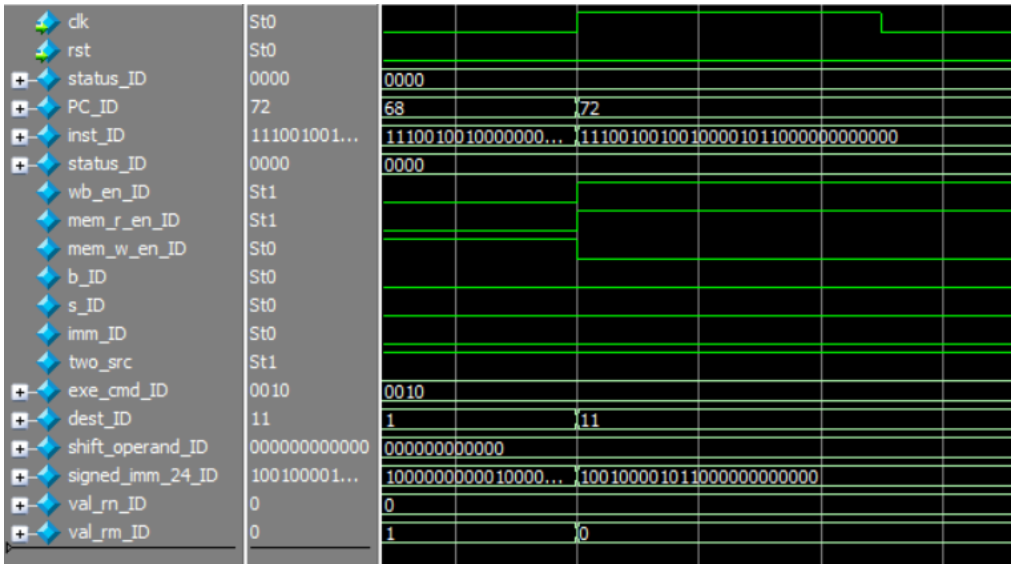
val_rn برابر 0 است. در دستور STR خروجی دوم رجیسترفایل برابر rd می‌باشد که 1 است.



دستور 18:

```
//LDR       R11,[R0],#0

b1110_01_0_0100_1_0000_1011_000000000000
```

دستور LDR تنها دستوری است که سیگنال mem_r_en در آن فعال می‌شود.



Quartus:

Flow Summary	
Flow Status	Successful - Mon Apr 03 00:12:41 2023
Quartus II 64-Bit Version	13.0.1 Build 232 06/12/2013 SP 1 SJ Web Edition
Revision Name	EXP2
Top-level Entity Name	DE2
Family	Cyclone II
Device	EP2C35F672C6
Timing Models	Final
Total logic elements	0 / 33,216 (0 %)
Total combinational functions	0 / 33,216 (0 %)
Dedicated logic registers	0 / 33,216 (0 %)
Total registers	0
Total pins	418 / 475 (88 %)
Total virtual pins	0
Total memory bits	0 / 483,840 (0 %)
Embedded Multiplier 9-bit elements	0 / 70 (0 %)
Total PLLs	0 / 4 (0 %)