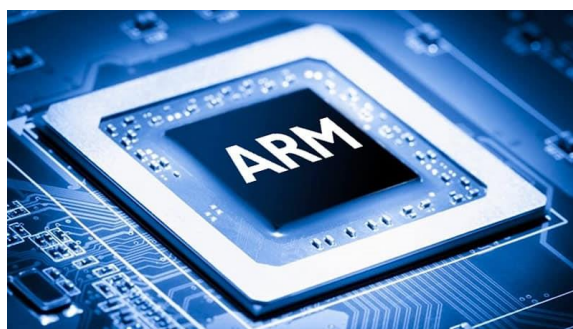


به نام خدا



دانشگاه تهران  
پردیس دانشکده‌های فنی  
دانشکده برق و کامپیوتر



آزمایشگاه معماری کامپیوتر

گزارش دستور کار شماره 2

محمد صالح عرفاتی

810197543

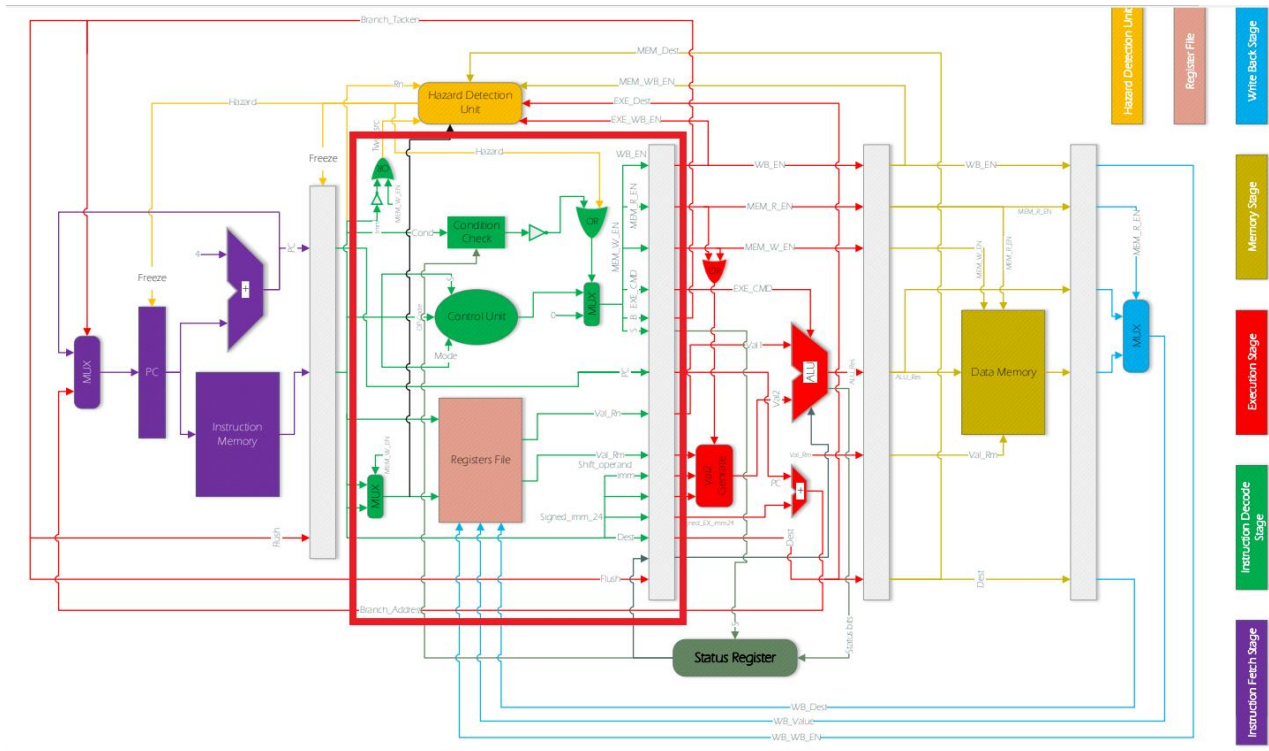
علی پادیاو

810199388

بهار 1402

## مقدمه

در این جلسه بخش Instruction Decode را تکمیل می‌کنیم.



ماژول‌ها:

## 1) Register File:

کد:

```

1  module RegisterFile (
2      input clk, rst,
3      input [3:0] WB_Dest, Rs1, Rs2,
4      input [31:0] WB_Value,
5      input WB_WB_EN,
6      output [31:0] Val_Rn, Val_Rm
7  );
8
9      reg [31:0] RegFile [0:14];
10
11     integer i;
12
13     initial
14     begin
15         for (i = 0; i < 15 ; i = i + 1 )
16             RegFile[i] = i;
17     end
18
19     assign Val_Rn = RegFile[Rs1];
20     assign Val_Rm = RegFile[Rs2];
21
22     always @(negedge clk, posedge rst)
23     begin
24         if (rst)
25             for (i = 0; i < 15 ; i = i + 1 )
26                 RegFile[i] <= i;
27         else if (WB_WB_EN)
28             RegFile[WB_Dest] <= WB_Value;
29     end
30
31 endmodule
32

```

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

## 2) Control Unit:

کد:

```

1 module ControlUnit (
2     input [3:0] opcode,
3     input [1:0] mode,
4     input S_in,
5     output reg [3:0] EXE_CMD,
6     output reg WB_EN, MEM_R_EN, MEM_W_EN, B, S_out
7 );
8
9 `define MOV 4'b1101
10 `define MVN 4'b1111
11 `define ADD 4'b0100
12 `define ADC 4'b0101
13 `define SUB 4'b0010
14 `define SBC 4'b0110
15 `define AND 4'b0000
16 `define ORR 4'b1100
17 `define EOR 4'b0001
18 `define CMP 4'b1010
19 `define TST 4'b1000
20 `define LDR 4'b0100
21 `define STR 4'b0100
22
23 always @(mode, opcode, S_in)
24 begin
25     {EXE_CMD, WB_EN, MEM_R_EN, MEM_W_EN, B, S_out} = 9'b0;
26
27     case(mode)
28     2'b00:
29         begin
30             S_out = S_in;
31             case(opcode)
32             `MOV:
33                 begin
34                     EXE_CMD = 4'b0001;
35                     WB_EN = 1'b1;
36                 end
37             `MVN:
38                 begin
39                     EXE_CMD = 4'b1001;
40                     WB_EN = 1'b1;
41                 end
42             `ADD:
43                 begin
44                     EXE_CMD = 4'b0010;
45                     WB_EN = 1'b1;
46                 end
47             `ADC:
48                 begin
49                     EXE_CMD = 4'b0011;
50                     WB_EN = 1'b1;
51                 end
52             `SUB:
53                 begin
54                     EXE_CMD = 4'b0100;
55                     WB_EN = 1'b1;
56                 end
57             `SBC:
58                 begin
59                     EXE_CMD = 4'b0101;
60                     WB_EN = 1'b1;
61                 end
62             `AND:
63                 begin
64                     EXE_CMD = 4'b0110;
65                     WB_EN = 1'b1;
66                 end
67             `ORR:
68                 begin
69                     EXE_CMD = 4'b1100;
70                     WB_EN = 1'b1;
71                 end
72             `EOR:
73                 begin
74                     EXE_CMD = 4'b0001;
75                     WB_EN = 1'b1;
76                 end
77             `CMP:
78                 begin
79                     EXE_CMD = 4'b1010;
80                     WB_EN = 1'b1;
81                 end
82             `TST:
83                 begin
84                     EXE_CMD = 4'b1000;
85                     WB_EN = 1'b1;
86                 end
87             `LDR:
88                 begin
89                     EXE_CMD = 4'b0100;
90                     WB_EN = 1'b1;
91                     MEM_R_EN = 1'b1;
92                     MEM_W_EN = 0'b0;
93                     B = 1'b1;
94                 end
95             `STR:
96                 begin
97                     EXE_CMD = 4'b0100;
98                     WB_EN = 1'b1;
99                     MEM_R_EN = 0'b0;
100                    MEM_W_EN = 1'b1;
101                    B = 1'b1;
102                end
103            default:
104                EXE_CMD = 4'b0000;
105            endcase
106        end
107    endcase
108 end

```

```

1
2     `ORR:
3     begin
4         EXE_CMD = 4'b0111;
5         WB_EN = 1'b1;
6     end
7
8     `EOR:
9     begin
10        EXE_CMD = 4'b1000;
11        WB_EN = 1'b1;
12    end
13
14    `CMP:
15    begin
16        EXE_CMD = 4'b0100;
17    end
18
19    `TST:
20    begin
21        EXE_CMD = 4'b0110;
22    end
23
24    default:
25        EXE_CMD = 4'b0000;
26    endcase
27 end
28
29
30 2'b01:
31 begin
32     EXE_CMD = 4'b0010;
33     MEM_R_EN = S_in;
34     MEM_W_EN = !S_in;
35     WB_EN = S_in;
36 end
37
38
39 2'b10:
40 begin
41     B = 1'b1;
42 end
43
44 default:;
45 endcase
46
47 end
48
49 endmodule
50

```

در این مازول بر اساس mode و opcode سیگنال‌ها تنظیم میشوند. در تصویر مشخص است که سه دستور CMP، TST و STR سیگنال WB\_EN فعال نمی‌باشد. همینطور در تمام دستورات محاسباتی S\_out برابر S\_in (mode=00) می‌شود.

## 3) Condition Check:

کد:

```

1 module ConditionCheck (
2     input [3:0] cond,
3     Status_R,    // N Z C V
4     output reg Is_Valid
5 );
6
7 always @ (cond, Status_R)
8 case (cond)
9     // Z set
10    4'b0000:
11        Is_Valid = Status_R[2];
12    // Z clear
13    4'b0001:
14        Is_Valid = !Status_R[2];
15    // C set
16    4'b0010:
17        Is_Valid = Status_R[1];
18    // C clear
19    4'b0011:
20        Is_Valid = !Status_R[1];
21    // N set
22    4'b0100:
23        Is_Valid = Status_R[3];
24    // N clear
25    4'b0101:
26        Is_Valid = !Status_R[3];
27    // V set
28    4'b0110:
29        Is_Valid = Status_R[0];
30    // V clear
31    4'b0111:
32        Is_Valid = !Status_R[0];
33    // C set and Z clear
34    4'b1000:
35        Is_Valid = Status_R[1] && !Status_R[2];
36    // C clear or Z set
37    4'b1001:
38        Is_Valid = !Status_R[1] || Status_R[2];
39    // N == V
40    4'b1010:
41        Is_Valid = Status_R[3] == Status_R[0];
42    // N != V
43    4'b1011:
44        Is_Valid = Status_R[3] != Status_R[0];
45    // Z==0, N==V
46    4'b1100:
47        Is_Valid = !Status_R[2] && (Status_R[3] == Status_R[0]);
48    // Z==1, N!=V
49    4'b1101:
50        Is_Valid = Status_R[2] && (Status_R[3] != Status_R[0]);
51    // Always
52    4'b1110:
53        Is_Valid = 1'b1;
54    // Never
55    4'b1111:
56        Is_Valid = 1'b0;
57 endcase
58
59 endmodule
60

```

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

```

1  wire [3:0] cond = Ins[31:28];
2  wire [1:0] mode = Ins[27:26];
3  wire I = Ins[25];
4  wire [3:0] opcode = Ins[24:21];
5  wire S_in = Ins[20];
6  wire [3:0] Rn = Ins[19:16];
7  wire [3:0] Rd = Ins[15:12];
8  wire [3:0] Rm = MEM_W_EN ? Rd : Ins[3:0];
9  assign shift_operand = Ins[11:0];
10 assign signed_imm_24 = Ins[23:0];
11
12 ConditionCheck CC (
13     .cond(cond),
14     .Status_R(Status_R),
15
16     .Is_Valid(Is_Valid)
17 );
18
19 RegisterFile RF (
20     .clk(clk),
21     .rst(rst),
22     .Rs1(Rn),
23     .Rs2(Rm),
24     .WB_WB_EN(WB_WB_EN),
25     .WB_Dest(WB_Dest),
26     .WB_Value(WB_Value),
27
28     .Val_Rn(Val_Rn),
29     .Val_Rm(Val_Rm)
30 );
31
32 ControlUnit CU (
33     .opcode(opcode),
34     .mode(mode),
35     .S_in(S_in),
36
37     .EXE_CMD(EXE_CMD_CU),
38     .WB_EN(WB_EN_CU),
39     .MEM_R_EN(MEM_R_EN_CU),
40     .MEM_W_EN(MEM_W_EN_CU),
41     .B(B_CU),
42     .S_out(S_CU)
43 );
44
45 assign imm = I;
46 assign Dest = Rd;
47 assign stop = hazard || !Is_Valid;
48 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};
49 assign Two_src = MEM_W_EN || !I;

```

ابتدا بخش‌های مختلف Instruction را مشخص کردیم. بعد از آن از ماژول‌های ساخته شده در ID\_Stage اینستنس گرفتیم و در نهایت output‌های باقی‌مانده را assign کردیم.

## 5) ID\_Reg

```
1  module IF_Reg (  
2      input clk, rst, freeze, flush,  
3      input [31:0] PC_in, instruction_in,  
4      output reg [31:0] PC_out, instruction_out  
5  );  
6  
7  always @(posedge clk, posedge rst)  
8  begin  
9      if (rst)  
10     begin  
11         PC_out <= 0;  
12         instruction_out <= 0;  
13     end  
14     else if (flush)  
15     begin  
16         PC_out <= 0;  
17         instruction_out <= 0;  
18     end  
19     else if (~freeze)  
20     begin  
21         PC_out <= PC_in;  
22         instruction_out <= instruction_in;  
23     end  
24 end  
25  
26 endmodule  
27
```

دستورات برنامه:

کد:

```

1  module Ins_Mem (
2      input [31:0] in,
3      output [31:0] out
4  );
5
6      reg [31:0] mem[31:0];
7
8      initial
9      begin
10         mem[0] = 32'b1110_00_1_1101_0_0000_0000_000000010100;
11         mem[1] = 32'b1110_00_1_1101_0_0000_0001_101000000001;
12         mem[2] = 32'b1110_00_1_1101_0_0000_0010_000100000011;
13         mem[3] = 32'b1110_00_0_0100_1_0010_0011_000000000010;
14         mem[4] = 32'b1110_00_0_0101_0_0000_0100_000000000000;
15         mem[5] = 32'b1110_00_0_0010_0_0100_0101_000100000100;
16         mem[6] = 32'b1110_00_0_0110_0_0000_0110_000010100000;
17         mem[7] = 32'b1110_00_0_1100_0_0101_0111_000101000010;
18         mem[8] = 32'b1110_00_0_0000_0_0111_1000_000000000011;
19         mem[9] = 32'b1110_00_0_1111_0_0000_1001_000000000110;
20         mem[10] = 32'b1110_00_0_0001_0_0100_1010_000000000101;
21         mem[11] = 32'b1110_00_0_1010_1_1000_0000_000000000110;
22         mem[12] = 32'b0001_00_0_0100_0_0001_0001_000000000001;
23         mem[13] = 32'b1110_00_0_1000_1_1001_0000_000000001000;
24         mem[14] = 32'b0000_00_0_0100_0_0010_0010_000000000010;
25         mem[15] = 32'b1110_00_1_1101_0_0000_0000_101100000001;
26         mem[16] = 32'b1110_01_0_0100_0_0000_0001_000000000000;
27         mem[17] = 32'b1110_01_0_0100_1_0000_1011_000000000000;
28     end
29
30     assign out = mem[in>>2];
31
32 endmodule
33

```

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



SIMULATION

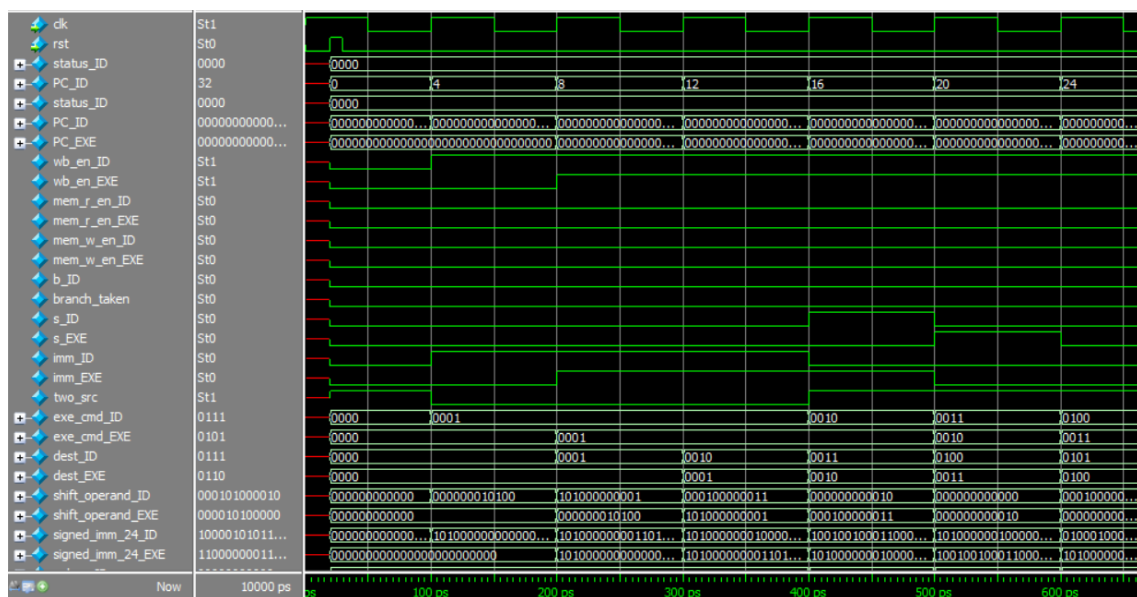
برای چک کردن خروجی‌ها کد تست‌بنچ زیر را نوشتیم و در modelsim اجرا کردیم.

```

1  module TB_DE2 ();
2
3      reg clk, rst;
4
5      ARM arm (
6          .clk(clk),
7          .rst(rst)
8      );
9
10     initial
11     begin
12         clk = 1;
13         repeat (200)
14         begin
15             #50;
16             clk = ~clk;
17         end
18     end
19
20     initial
21     begin
22         rst = 0;
23         #20 rst = 1;
24         #10 rst = 0;
25     end
26
27 endmodule
28

```

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.

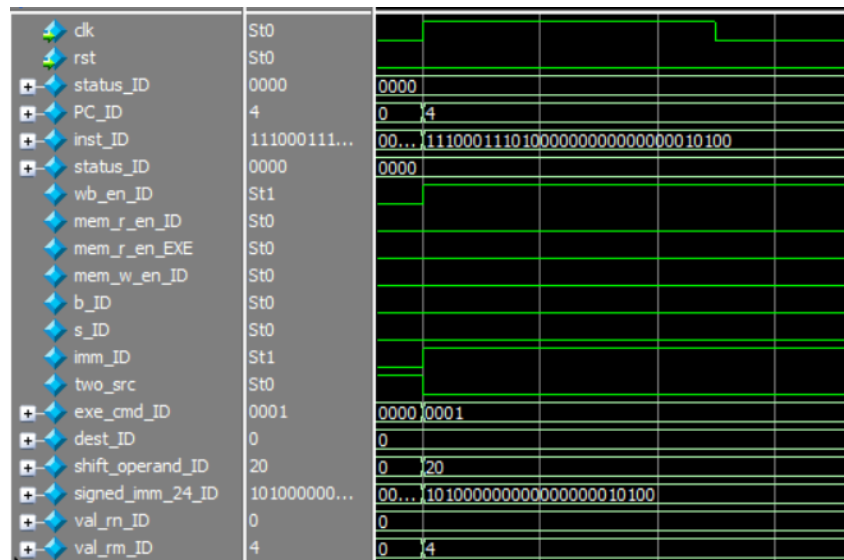
دستور 1:

```
//MOV      R0, #20
```

```
1110_00_1_1101_0_0000_0000_0000000010100
```

در این دستور wb\_en\_ID و imm\_ID و 1 exe\_cmd\_ID و 0001 میشوند که در تصویر مشخص است.

همچنین مقادیر shift\_operand\_ID، و Dest نیز با اینستراکشن‌ها همخوانی دارد.

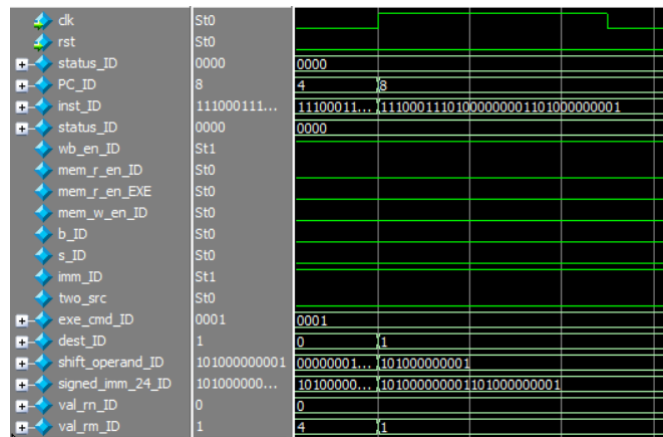


دستور 2:

//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.

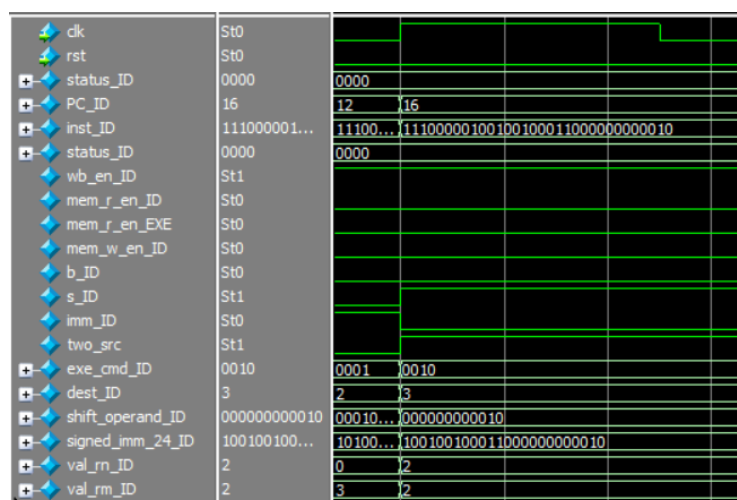


دستور 4:

//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.



```
//ADDEQ    R2, R2, R2
```

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.

Register	Value	Hex	Dec
clk	St0		
rst	St0		
status_ID	0000	0000	
PC_ID	60	56	60
inst_ID	000000001...	111000010001100...	00000000100000100010000000000010
status_ID	0000	0000	
wb_en_ID	St0		
mem_r_en_ID	St0		
mem_r_en_EXE	St0		
mem_w_en_ID	St0		
b_ID	St0		
s_ID	St0		
imm_ID	St0		
two_src	St1		
exe_cmd_ID	0000	0110	0000
dest_ID	2	0	2
shift_operand_ID	000000000010	0000000001000	0000000000010
signed_imm_24_ID	100000100...	000110010000000...	100000100010000000000010
val_rn_ID	2	9	2
val_rm_ID	2	8	2

دستور 17:

//STR R1, [R0], #0

1110\_01\_0\_0100\_0\_0000\_0001\_000000000000

از آنجایی 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.

clk	St0				
rst	St0				
status_ID	0000	0000			
PC_ID	68	64	68		
inst_ID	111001001...	11100011...	11100100100000000010000000000000		
status_ID	0000	0000			
wb_en_ID	St0				
mem_r_en_ID	St0				
mem_r_en_EXE	St0				
mem_w_en_ID	St1				
b_ID	St0				
s_ID	St0				
imm_ID	St0				
two_src	St1				
exe_cmd_ID	0010	0001	0010		
dest_ID	1	0	1		
shift_operand_ID	000000000000	10110000...	000000000000		
signed_imm_24_ID	100000000...	10100000...	10000000000010000000000000		
val_rn_ID	0	0			
val_rm_ID	1	1	1		

## 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.

Register	Value	Hex	Dec	Bin	Oct	Hex	Dec	Bin	Oct
clk	St0								
rst	St0								
status_ID	0000	0000							
PC_ID	72	68	72						
inst_ID	111001001...	1110010010000000...	11100100100010000101100000000000						
status_ID	0000	0000							
wb_en_ID	St1								
mem_r_en_ID	St1								
mem_w_en_ID	St0								
b_ID	St0								
s_ID	St0								
imm_ID	St0								
two_src	St1								
exe_cmd_ID	0010	0010							
dest_ID	11	1	11						
shift_operand_ID	0000000000000	0000000000000							
signed_imm_24_ID	100100001...	10000000000010000...	100100001011000000000000						
val_rn_ID	0	0							
val_rm_ID	0	1	0						

: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 % )