

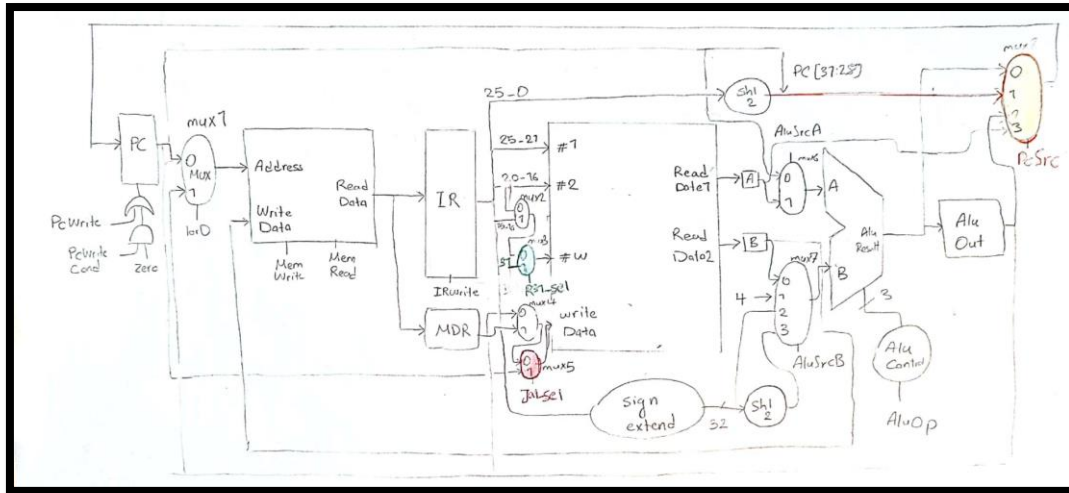
به نام خدا

## پروژه ۳ - پیاده سازی چندمرحله ای پردازنده MIPS

اولدوز نیساری (۸۱۰۱۹۹۵۰۵) - ثمر نیک فرجاد (۸۱۰۱۹۹۵۰۸)

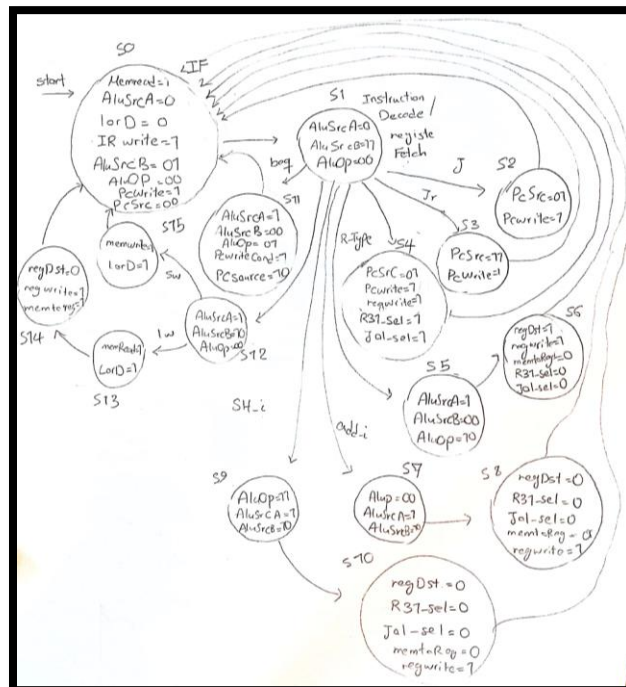
زمستان ۱۴۰۱

## • تصویر مسیر داده



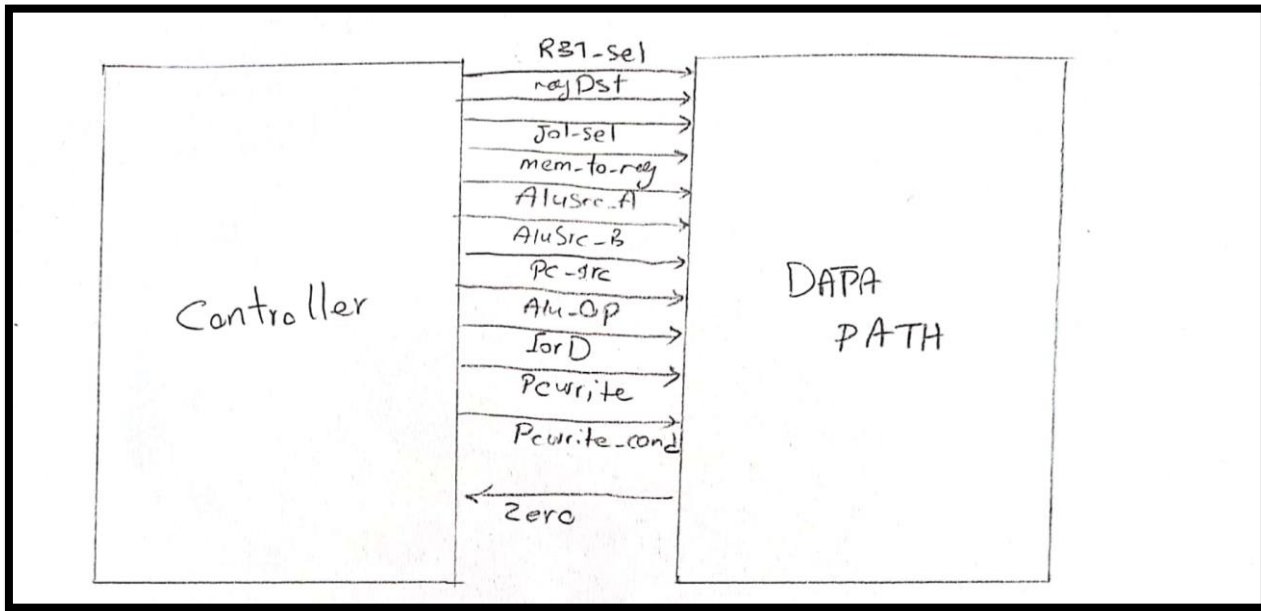
\*توضیحات : تغییرات مسیر داده نسبت به حالت اولیه، شامل تغییر مولتی پلکسر ۲ و اضافه کردن مولتی پلکسرهای ۵ و ۳ در تصویر است. به این صورت مسیر داده‌ی تدریس شده در درس را برای اجرای دستورات `jal`, `jr`, `jalr`, `jalr` آماده کردیم. ( واحد های اضافه شده و مسیر هر دستور با رنگ مشخص آن دستور مشخص شده است )

## • تصویر کنترلر



\*توضیحات : استیت های `S0`, `S1` برای تمام دستورات لازم هستند پس از این استیت ها بر اساس نوع دستوری که داریم در یکی از مسیر های کنترلر پیش می رویم. همانطور که در شکل مشخص شده، استیت هایی برای اجرای دستورات `jr`, `jalr`, `jal`, `jalr` اضافه شده. دستور های `beq`, `jr`, `jal` به 3 استیت و دستورهای `addi`, `sw`, `jalr` به 4 استیت و دستور `lw` به 5 استیت نیاز است.

- تصویر رابطه ی بین دیتاپس و کنترلر



- تصویر کدهای وریلاگ (با توجه به طولانی بودن، صرفا ورودی ها و خروجی ها آورده شده)

- مسیر داده :

```

11 module datapath (clk, rst,
12 pc_id, IorD, IRwrite, reg_dst, mem_to_reg, alu_src_A, alu_src_B, alu_ctrl, reg_write, pc_src,
13 zero,
14 R31_sel, jal_sel,
15 mem_out,
16 inst_out,
17 inst_data_adr, mem_data_in
18 );
19 input clk, rst;
20 output zero;
21
22 input pc_id, IorD, IRwrite, reg_dst, mem_to_reg, alu_src_A, reg_write;
23 input [1:0] alu_src_B, pc_src;
24 input [2:0] alu_ctrl;
25
26 input R31_sel, jal_sel;
27
28 input [31:0] mem_out;
29
30 output [31:0] inst_out;
31
32 output [31:0] inst_data_adr, mem_data_in;
33
34
35
36
  
```

- کنترلر :

```

17 module controller ( opcode, func, zero,
18 IorD, IRwrite, R31_sel, jal_sel,
19 reg_dst, mem_to_reg, alu_src_A, alu_src_B, operation, reg_write,
20 pc_src, pc_id, mem_read, mem_write, clk, rst
21 );
22
23
24 input [5:0] opcode;
25 input [5:0] func;
26 input zero, clk, rst;
27 output reg IorD, IRwrite, R31_sel, jal_sel, reg_dst, mem_to_reg, alu_src_A, reg_write,
28 pc_id, mem_read, mem_write;
29 output [2:0] operation;
30 output reg [1:0] pc_src, alu_src_B;
31
32 reg [4:0] ps, ns;
33 reg [1:0] alu_Op;
34
35 reg pcwrite, pcwrite_cond;
36
  
```

- پردازنده :

```

19 module mips(rst, clk, mem_out, mem_read, mem_write, inst_data_adr, mem_data_in);
20
21
22 input rst, clk;
23
24 input [31:0] mem_out;
25
26 output mem_read, mem_write;
27
28 output [31:0] inst_data_adr;
29 output [31:0] mem_data_in;
30
31
  
```

## - مموری :

```

2
3 module memory (inst_data_addr, mem_data_in, mrd, mwr, clk, mem_out, min_value, min_index);
4     input [31:0] inst_data_addr;
5     input [31:0] mem_data_in;
6     input mrd, mwr, clk;
7     output [31:0] mem_out;
8     output [31:0] min_value;
9     output [31:0] min_index;
10
11     reg [7:0] mem[0:65535];
12
13

```

## - تست بنچ :

```

5 module multicycle_mips_tb;
6
7     wire [31:0] inst_data_addr, mem_out, mem_data_in, min_value, min_index;
8     wire mem_read, mem_write;
9     reg clk, rst;
10
11     mips CPU (rst, clk, mem_out, mem_read, mem_write, inst_data_addr, mem_data_in);
12
13     memory MEM (inst_data_addr, mem_data_in, mem_read, mem_write, clk, mem_out, min_value, min_index);
14
15     initial
16     begin
17         rst = 1'b1;
18         clk = 1'b0;
19         #9 rst = 1'b0;
20         #7201 $stop;
21     end
22
23     always
24     begin
25         #7 clk = ~clk;
26     end
27
28 endmodule

```

\*توضیحات: برای تست کردن پردازنده، برنامه ای مطابق آنچه در شرح پروژه آمده است به زبان اسمبلی نوشتیم. همچنین 20 عدد تصادفی را از آدرس 1000 در فایل حافظه نوشتیم تا برای تست از آن ها استفاده کنیم.

## • برنامه به زبان اسمبلی

```

19
20 //0 //PRGM: addi R1,R0,1000          ///R1: starting address
21 //4 // lw R2,0(R1)                ///R2: min element value
22 //8 // addi R3,R0,0                ///R3: index(i)
23 //12 // addi R4,R0,0                ///R4: min element index
24 //16 // addi R5,R0,19                ///R5: number of iterations
25
26 //20 //LOOP: beq R3,R5,END_LOOP
27 //24 // addi R1,R1,4
28 //28 // addi R3,R3,1
29 //32 // lw R6,0(R1)
30 //36 // slt R7,R6,R2
31 //40 // beq R7,R0,LOOP
32 //44 // addi R4,R3,0
33 //48 // addi R2,R6,0
34 //52 // j LOOP
35
36 //56 //END_LOOP: sw R2,2000(R0)
37 //60 // sw R4,2004(R0)
38

```

## • برنامه به زبان صفر و یک

```

39 // {mem[3], mem[2], mem[1], mem[0]} = {6'b001001, 5'd0, 5'd1, 16'd1000};
40 // {mem[7], mem[6], mem[5], mem[4]} = {6'b100011, 5'd1, 5'd2, 16'd0};
41 // {mem[11], mem[10], mem[9], mem[8]} = {6'b001001, 5'd0, 5'd3, 16'd0};
42 // {mem[15], mem[14], mem[13], mem[12]} = {6'b001001, 5'd0, 5'd4, 16'd0};
43 // {mem[19], mem[18], mem[17], mem[16]} = {6'b001001, 5'd0, 5'd5, 16'd19};
44 //
45 // {mem[23], mem[22], mem[21], mem[20]} = {6'b000100, 5'd3, 5'd5, 16'd8};
46 // {mem[27], mem[26], mem[25], mem[24]} = {6'b001001, 5'd1, 5'd1, 16'd4};
47 // {mem[31], mem[30], mem[29], mem[28]} = {6'b001001, 5'd3, 5'd3, 16'd1};
48 // {mem[35], mem[34], mem[33], mem[32]} = {6'b100011, 5'd1, 5'd6, 16'd0};
49 // {mem[39], mem[38], mem[37], mem[36]} = {6'b000000, 5'd6, 5'd2, 5'd7, 5'd0, 6'b101010};
50 // {mem[43], mem[42], mem[41], mem[40]} = {6'b000100, 5'd7, 5'd0, -16'd6};
51 // {mem[47], mem[46], mem[45], mem[44]} = {6'b001001, 5'd3, 5'd4, 16'd0};
52 // {mem[51], mem[50], mem[49], mem[48]} = {6'b001001, 5'd6, 5'd2, 16'd0};
53 // {mem[55], mem[54], mem[53], mem[52]} = {6'b000010, 26'd5};
54 //
55 // {mem[59], mem[58], mem[57], mem[56]} = {6'b101011, 5'd0, 5'd2, 16'd2000};
56 // {mem[63], mem[62], mem[61], mem[60]} = {6'b101011, 5'd0, 5'd4, 16'd2004};
57 //

```

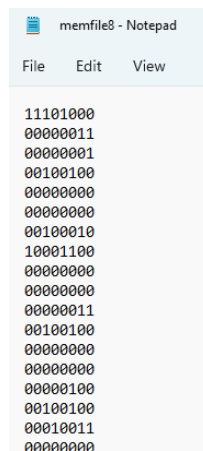
- داده‌ها به زبان صفر و یک

```

58 // {mem[1003], mem[1002], mem[1001], mem[1000]} = 32'd12;
59 // {mem[1007], mem[1006], mem[1005], mem[1004]} = 32'd13;
60 // {mem[1011], mem[1010], mem[1009], mem[1008]} = 32'd21;
61 // {mem[1015], mem[1014], mem[1013], mem[1012]} = 32'd31;
62 // {mem[1019], mem[1018], mem[1017], mem[1016]} = 32'd44;
63 // {mem[1023], mem[1022], mem[1021], mem[1020]} = 32'd53;
64 // {mem[1027], mem[1026], mem[1025], mem[1024]} = 32'd19;
65 // {mem[1031], mem[1030], mem[1029], mem[1028]} = 32'd2;
66 // {mem[1035], mem[1034], mem[1033], mem[1032]} = (-32'd11);
67 // {mem[1039], mem[1038], mem[1037], mem[1036]} = 32'd49;
68 // {mem[1043], mem[1042], mem[1041], mem[1040]} = 32'd52;
69 // {mem[1047], mem[1046], mem[1045], mem[1044]} = 32'd13;
70 // {mem[1051], mem[1050], mem[1049], mem[1048]} = 32'd27;
71 // {mem[1055], mem[1054], mem[1053], mem[1052]} = 32'd36;
72 // {mem[1059], mem[1058], mem[1057], mem[1056]} = 32'd45;
73 // {mem[1063], mem[1062], mem[1061], mem[1060]} = 32'd51;
74 // {mem[1067], mem[1066], mem[1065], mem[1064]} = 32'd71;
75 // {mem[1071], mem[1070], mem[1069], mem[1068]} = 32'd62;
76 // {mem[1075], mem[1074], mem[1073], mem[1072]} = 32'd93;
77 // {mem[1079], mem[1078], mem[1077], mem[1076]} = (-32'd84);
78

```

- فایل مموری (چند خط ابتدایی)



```

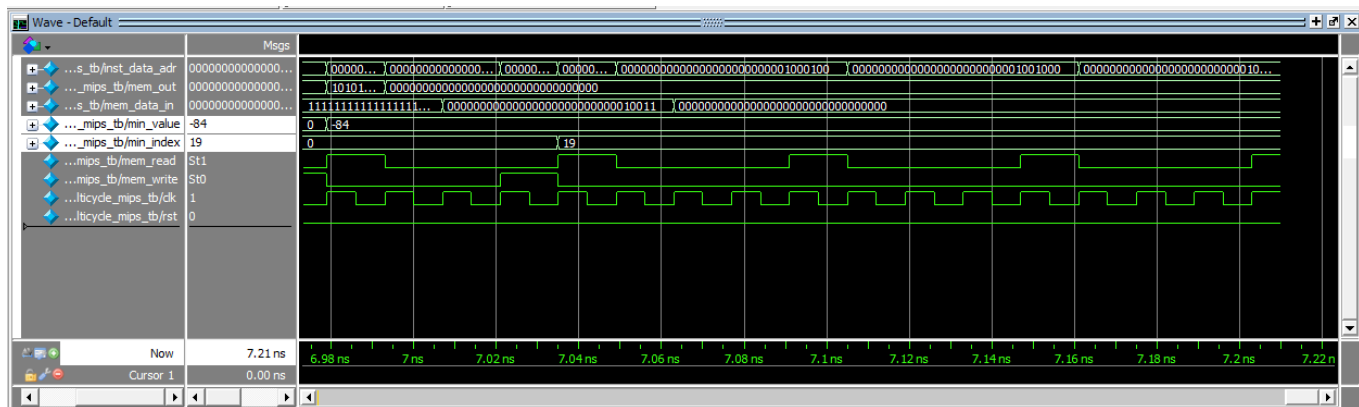
memfile8 - Notepad
File Edit View

11101000
00000011
00000001
00100100
00000000
00000000
00100010
10001100
00000000
00000000
00000011
00100100
00000000
00000000
00000100
00100100
00010011
00000000

```

\*توضیحات: در هر خط فایل مموری ۸ بین وجود دارد و موقع خواندن یا نوشتن نیز هر چهار خط به صورت یک مجموعه‌ی ۳۲بیتی خوانده یا نوشته می‌شود. دستورات از ابتدای فایل نوشته شده و داده‌ها از خط ۱۰۰۱ که در واقع با شروع از صفر همان خط ۱۰۰۰ است نوشته شده است. همه‌ی دستورات و داده‌ها ۳۲بیتی هستند. حد فاصل این دو بخش نیز در مموری صفر وارد شده است.

## • تصویر شکل موج خروجی



\* همانطور که مشخص است، شماره‌ی درایه‌ی کمینه و مقدار آن در خانه‌های مربوطه نوشته شده و نشان داده شده. این مقادیر توسط واسط فایل مموری و پردازنده (ماژول memory) نشان داده می‌شوند.