

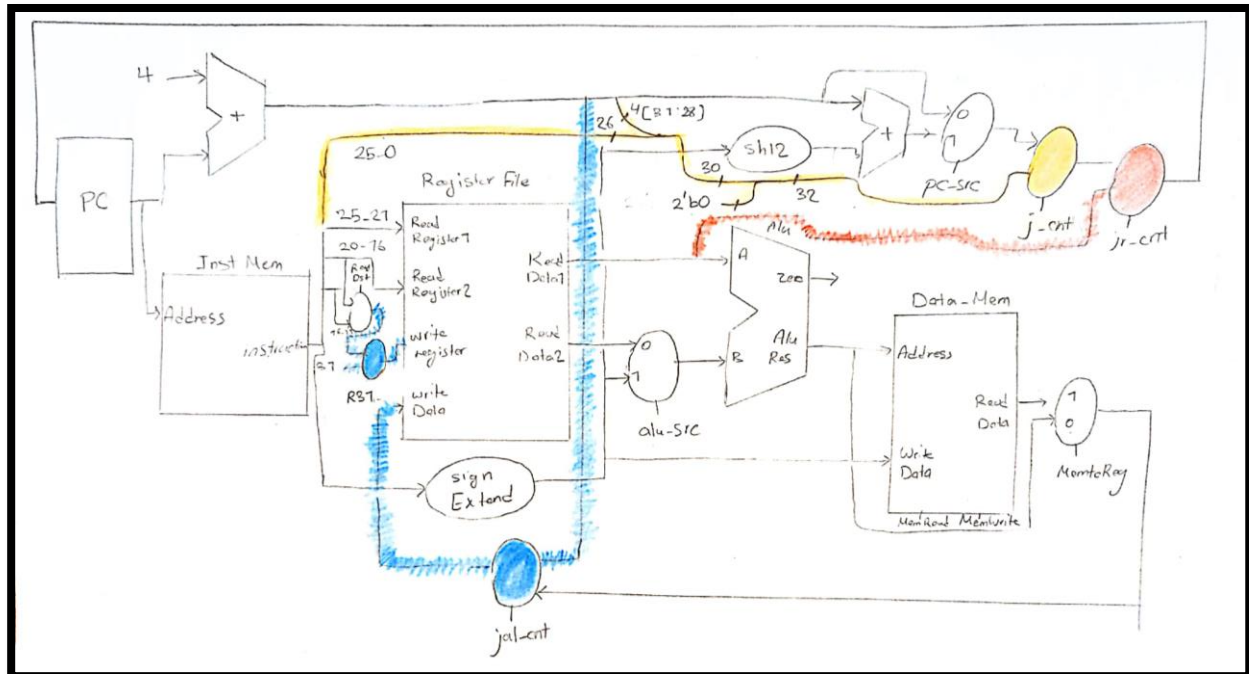
به نام خدا

پروژه ۲ – پیاده‌سازی تک‌مرحله‌ای پردازنده MIPS

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

زمستان ۱۴۰۱

● تصویر مسیر داده



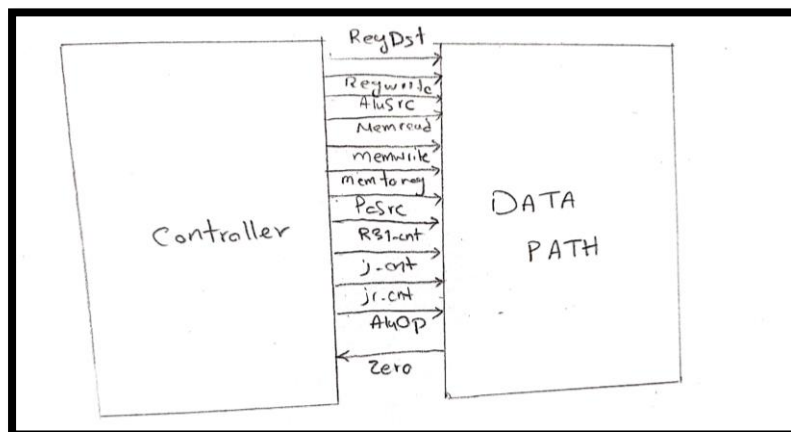
*توضیحات: مطابق آنچه که در درس بیان شده است. مسیر داده پیاده سازی تک مرحله ای پردازنده MIPS را در نظر گرفتیم و با افزودن تعدادی مالتی پلکسر مسیر داده را برای اجرای دستورات jal, jr, j آماده کردیم. (واحد های اضافه شده و مسیر هر دستور با رنگ مشخص آن دستور مشخص شده است.) لازم به ذکر است که به جای کانکت کردن بیت های صفر به سمت راست، در واقع می توان یک shl2 در مسیر داده قرار داد.

• جدول سیگنال‌های کنترلی

	RegDst	R31Ld	RegWrite	ALUSrc	Memread	Memwrite	MemtoReg	RSrc	j_cnt	jal_cnt	jr_cnt	ALOP
R-Type	1	0	1	0	0	0	0	0	0	0	0	10
lw	0	0	1	1	1	0	1	0	0	0	0	00
sw	x	0	0	1	0	1	x	0	0	0	0	00
beq	x	0	0	0	0	0	x	1	0	0	0	01
addi	0	0	1	1	0	0	0	0	0	0	0	00
slli	0	0	1	1	0	0	0	0	0	0	0	11
j	x	0	0	x	0	0	x	x	1	0	0	x
jr	x	0	0	x	0	0	x	x	0	1	0	x
jal	x	1	1	x	0	0	x	x	0	0	1	x

*توضیحات : از آنجایی که در پیاده سازی تک مرحله ای تنها از یک سیکل ساعت برای اجرای دستورات استفاده می شود، برای کنترلر کافی است سیگنال های خروجی کنترلر به ازای هر یک از دستورات را مشخص کنیم.

• 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 :

```

1 module datapath ( clk, rst, inst_adr, inst,
2   data_adr, data_out, data_in,
3   reg_dst, mem_to_reg, alu_src, pc_src, alu_ctrl, reg_write,
4   zero, j_cnt, jr_cnt, jal_cnt, R31
5 );
6 input clk, rst;
7 output [31:0] inst_adr;
8 input [31:0] inst;
9 output [31:0] data_adr;
10 output [31:0] data_out;
11 input [31:0] data_in;
12 input reg_dst, mem_to_reg, alu_src, pc_src, reg_write;
13 input [2:0] alu_ctrl;
14 output zero;
15 input j_cnt, jr_cnt, jal_cnt, R31;

```

- کنترلر :

```

1 module controller ( opcode, func, zero, reg_dst, mem_to_reg, reg_write,
2   alu_src, mem_read, mem_write, pc_src, operation,
3   j_cnt, jr_cnt, jal_cnt, R31
4 );
5
6 input [5:0] opcode;
7 input [5:0] func;
8 input zero;
9 output reg_dst, mem_to_reg, reg_write, alu_src,
10   mem_read, mem_write, pc_src;
11 reg reg_dst, mem_to_reg, reg_write,
12   alu_src, mem_read, mem_write;
13 output [2:0] operation;
14 output j_cnt, jr_cnt, jal_cnt, R31;
15 reg j_cnt, jr_cnt, jal_cnt, R31;
16
17 reg [1:0] alu_op;
18 reg branch;
19

```

- پردازنده :

```

1 module mips_single_cycle (rst, clk, inst_adr, inst, data_adr, data_in, data_out, mem_read, mem_write);
2 input rst, clk;
3 output [31:0] inst_adr;
4 input [31:0] inst;
5 output [31:0] data_adr;
6 input [31:0] data_in;
7 output [31:0] data_out;
8 output mem_read, mem_write;

```

- مموری دستور :

```

1 module inst_mem (adr, d_out);
2 input [31:0] adr;
3 output [31:0] d_out;
4

```

- مموری داده :

```

1 module data_mem (adr, d_in, mrd, mwr, clk, d_out, min_value, min_index);
2     input [31:0] adr;
3     input [31:0] d_in;
4     input mrd, mwr, clk;
5     output [31:0] d_out;
6     output [31:0] min_value;
7     output [31:0] min_index;
8
9     reg [7:0] mem[0:65535];
10

```

- تست بنچ :

```

1 module mips_tb;
2
3     wire [31:0] inst_adr, inst, data_adr, data_in, data_out, min_value, min_index;
4     wire mem_read, mem_write;
5     reg clk, rst;
6
7     mips_single_cycle CPU(rst, clk, inst_adr, inst, data_adr, data_out, data_in, mem_read, mem_write);
8     inst_mem IM (inst_adr, inst);
9     data_mem DM (data_adr, data_in, mem_read, mem_write, clk, data_out, min_value, min_index);
10
11     initial
12     begin
13         rst = 1'b1;
14         clk = 1'b0;
15         #9 rst = 1'b0;
16         #1887 $stop;
17     end
18
19     always
20     begin
21         #7 clk = ~clk;
22     end
23

```

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

• برنامه به زبان اسمبلی

```

//0 //PRGM: addi R1,R0,1000          ///R1: starting address
//4 //      lw R2,0(R1)              ///R2: min element value
//8 //      addi R3,R0,0              ///R3: index(i)
//12 //     addi R4,R0,0              ///R4: min element index
//16 //     addi R5,R0,19             ///R5: number of iterations

//20 //LOOP: beq R3,R5,END_LOOP
//24 //      addi R1,R1,4
//28 //      addi R3,R3,1
//32 //      lw R6,0(R1)
//36 //      slt R7,R6,R2
//40 //      beq R7,R0,LOOP
//44 //      addi R4,R3,0
//48 //      addi R2,R6,0
//52 //      j LOOP

//56 //END_LOOP: sw R2,2000(R0)
//60 //      sw R4,2004(R0)

```

• برنامه به زبان صفر و یک

```

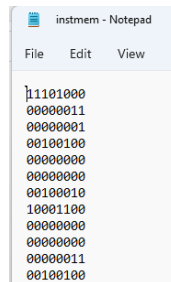
// {mem[3], mem[2], mem[1], mem[0]} = {6'b001001, 5'd0, 5'd1, 16'd1000};
// {mem[7], mem[6], mem[5], mem[4]} = {6'b100011, 5'd1, 5'd2, 16'd0};
// {mem[11], mem[10], mem[9], mem[8]} = {6'b001001, 5'd0, 5'd3, 16'd0};
// {mem[15], mem[14], mem[13], mem[12]} = {6'b001001, 5'd0, 5'd4, 16'd0};
// {mem[19],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

```

- داده‌ها به زبان صفر و یک

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

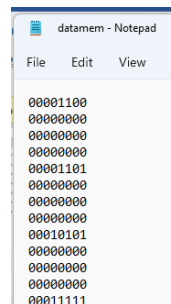
- فایل مموری دستور (چند خط ابتدایی)



```
instmem - Notepad
File Edit View

11101000
00000011
00000001
00100100
00000000
00000000
00100010
10011100
00000000
00000000
00000011
00100100
```

- فایل مموری داده (چند خط ابتدایی)

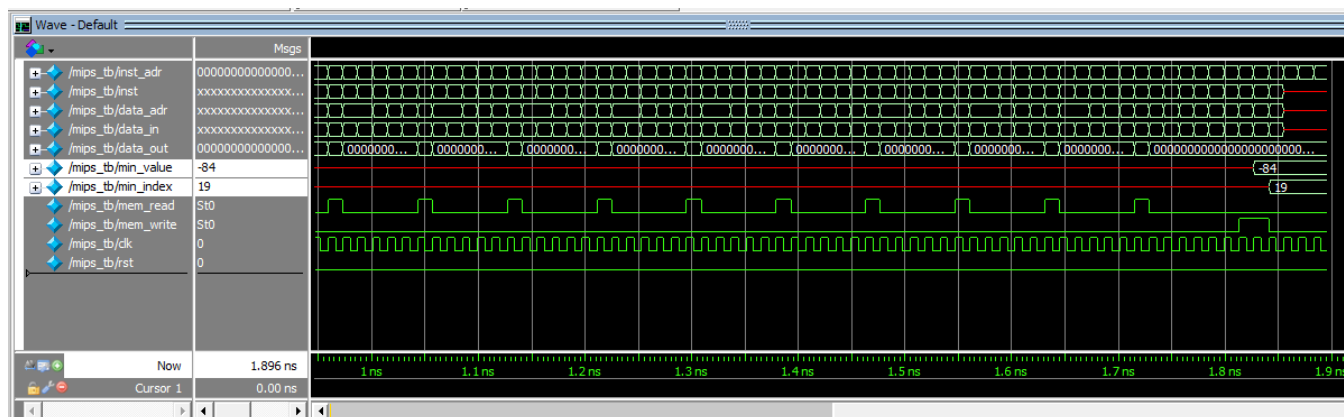


```
datamem - Notepad
File Edit View

00001100
00000000
00000000
00000000
00001101
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00011111
```

*توضیحات: در هر خط فایل‌های 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 مموری دستور نوشته شده و داده‌ها نیز از ابتدای فایل دستور نوشته شده. همه‌ی دستورات و داده‌ها ۳۲بیتی هستند. لازم به ذکر است که می‌شد داده‌ها را ۱۰۰۰ عدد پایین‌تر نوشت اما در کد فایل را از خانه‌ی صفر mem نوشت.

• 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 فایل مموری داده‌ها و پردازنده (ماژول data_mem) نشان داده می‌شوند.