معماری کامپیوتر - دکتر عطارزاده

تمرین سری ۶

سوال ۱ - الف)

برای این که کدهای موجود در مرجع هریس کار کنند، نیاز داریم تا دو ماژول ALU و RegFile را اضافه کنیم.

کد SystemVerilog مربوط به ماژول ALU به صورت زیر است.

```
module alu(input logic [31:0] a, b,
           input logic [2:0] control,
           output logic [31:0] z,
           output logic zero);

    always_comb
      case (control)
        3'b000: z = a + b;
        3'b001: z = a - b;
        3'b010: z = a & b;
        3'b011: z = a | b;
        3'b101: z = a < b ? 1 : 0;
        default: z = 32'bx;
      endcase

    assign zero = z == 0;

endmodule
```

در بالا، خروجی‌ها از روی ورودی control مشخص شده‌اند. همچنین خروجی zero نیز از روی مقدار محاسبه شده توسط ALU مشخص شده است.

همچنین کد SystemVerilog ماژول RegFile نیز به صورت زیر است.

```
module regfile(input logic clk,
    input logic WE3,
    input logic [4:0] A1, A2, A3,
    input logic [31:0] WD3,
    output logic [31:0] RD1,
    output logic [31:0] RD2);

    logic [31:0] regMem [31:0];

    assign RD1 = A1 == 0 ? 0 : regMem[A1[4:0]];
    assign RD2 = A2 == 0 ? 0 : regMem[A2[4:0]];

    always_ff @ (posedge clk) begin
      if (WE3) regMem[A3[4:0]] <= WD3;
    end

endmodule
```

برای ذخیره محتویات رجیستر فایل، از یک RAM استفاده شده است. مقدار خوانده شده از رجیستر فایل (که روی خروجی‌های RD1 و RD2 قرار می‌گیرد) از روی RAM خوانده می‌شود؛ مگر آن که مقدار رجیستر zero را بخواهیم؛ که در این صورت مقدار صفر روی خروجی‌ها قرار داده می‌شود و از RAM مقداری خوانده نمی‌شود. همچنین عمل نوشتن به صورت سنکرون و در لبه بالارونده کلاک رخ می‌دهد. اگر WE3 فعال باشد، نوشتن رخ می‌دهد.

در نهایت، برای این که در شبیه‌سازی به مشکل نخوریم، آدرس فایل riscvtest.txt را در ماژول imem به صورت زیر اصلاح می‌کنیم (و از آدرس‌دهی مطلق استفاده می‌کنیم).

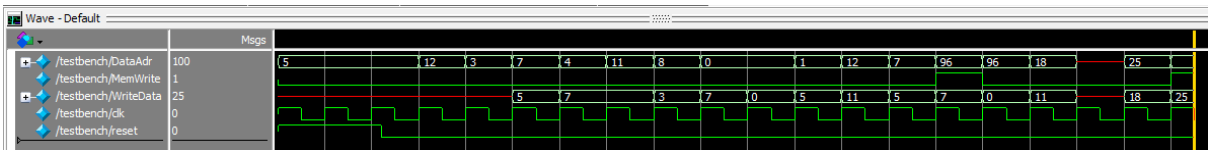
```
module imem(input logic [31:0] a,
    output logic [31:0] rd);

    logic [31:0] RAM[63:0];
    initial
       $readmemh("C:\\Users\\Mans\\Documents\\CA HW\\riscv-processor\\riscvtest.txt",RAM);

    assign rd = RAM[a[31:2]]; // word aligned
endmodule
```

لازم به ذکر است که کد ماژول‌های دیگر تغییری نمی‌کند و دقیقا مطابق کد ارائه شده در مرجع هریس است.

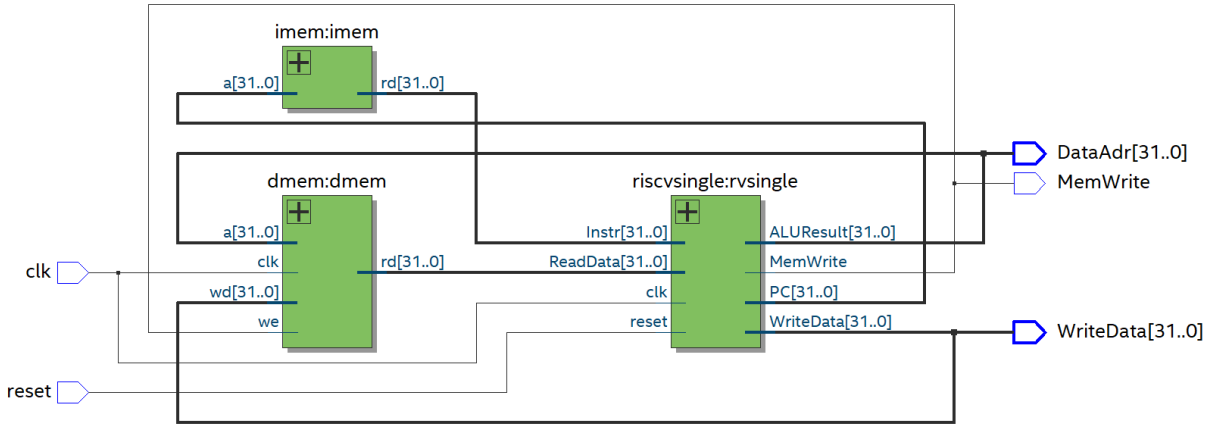
حال شبیه‌سازی را اجرا می‌کنیم. شکل موج به صورت زیر خواهد بود.



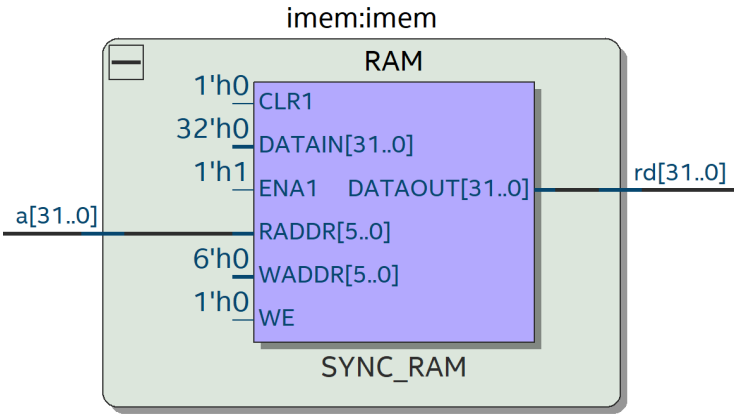
همچنین چاپ شدن پیام زیر، صحت عملکرد پردازنده را در شبیه‌سازی نشان می‌دهد.

```
Transcript
VSIM 9> run -all
# Simulation succeeded
# ** Note: $stop    : C:/Users/Mans/Documents/CA HW/riscv-processor/testbench.sv(26)
#    Time: 195 ps  Iteration: 1  Instance: /testbench
# Break in Module testbench at C:/Users/Mans/Documents/CA HW/riscv-processor/testbench.sv line 26
```

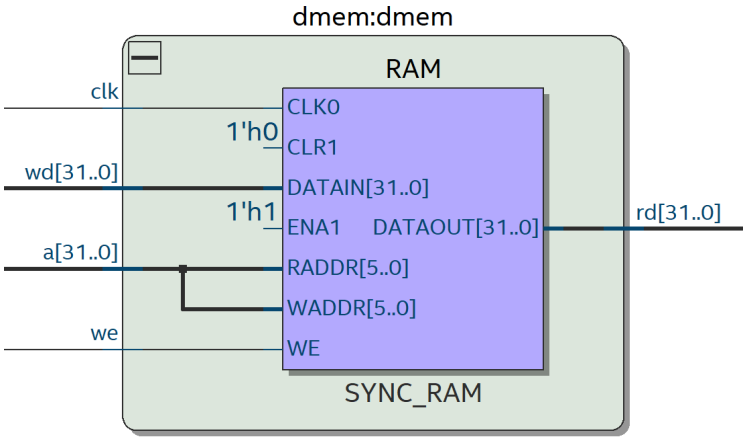
خروجی RTL-View نیز به صورت زیر است (ماژول top).



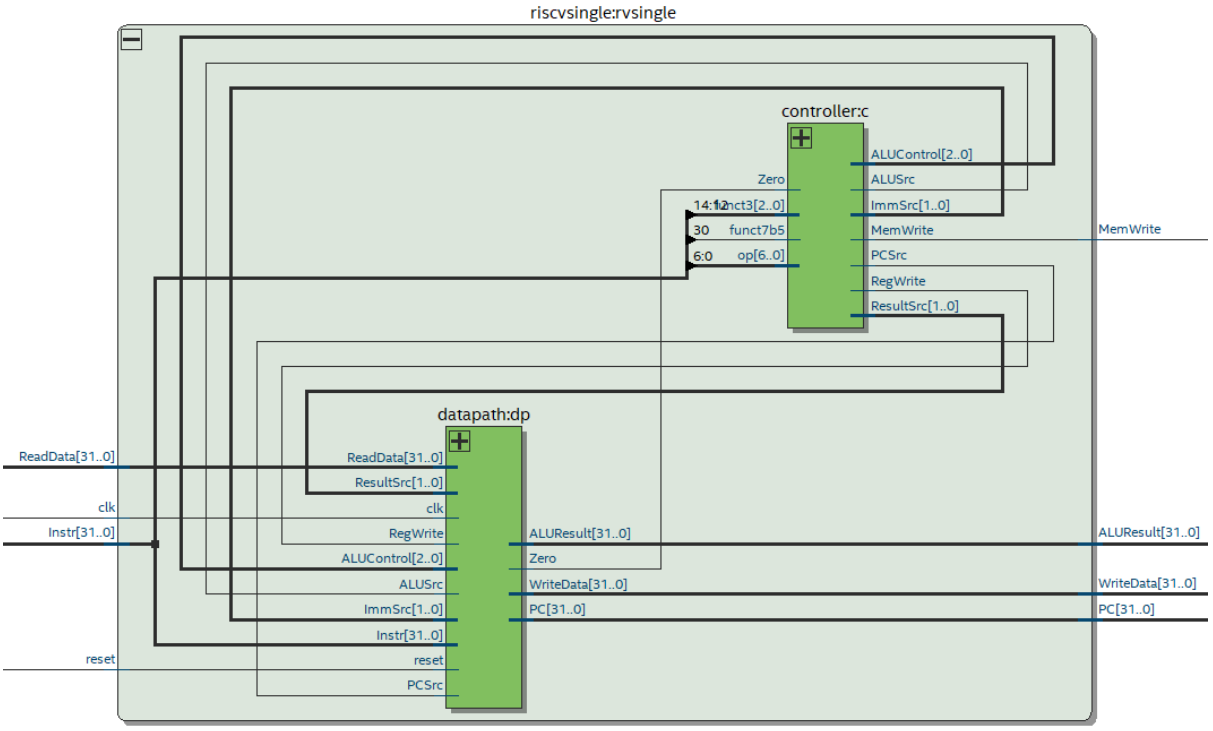
(ماژول imem)



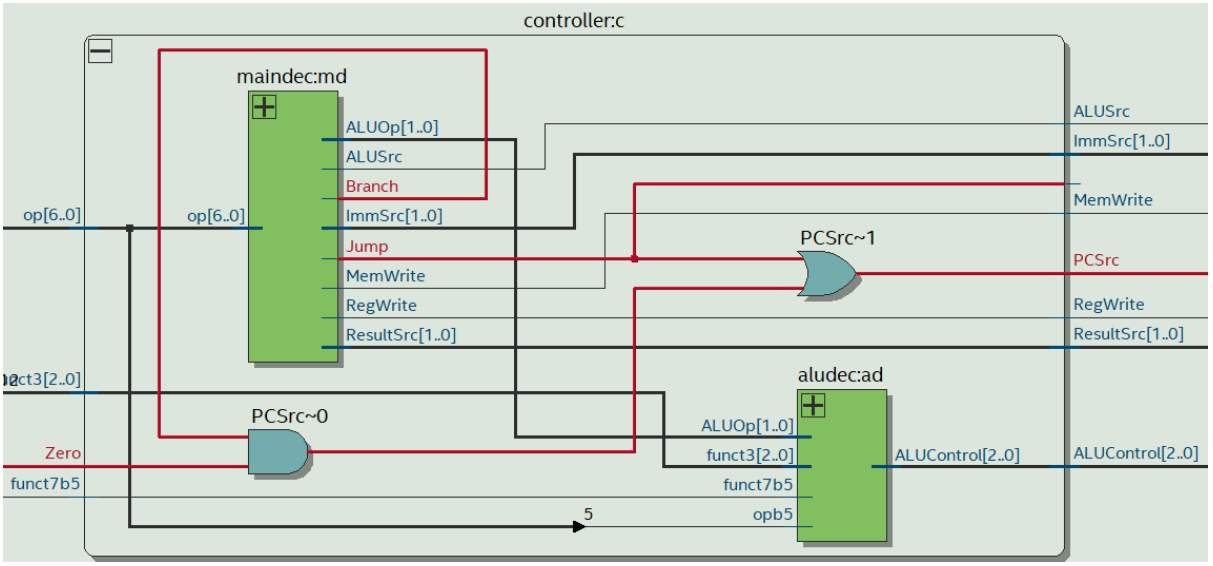
(ماژول dmem)



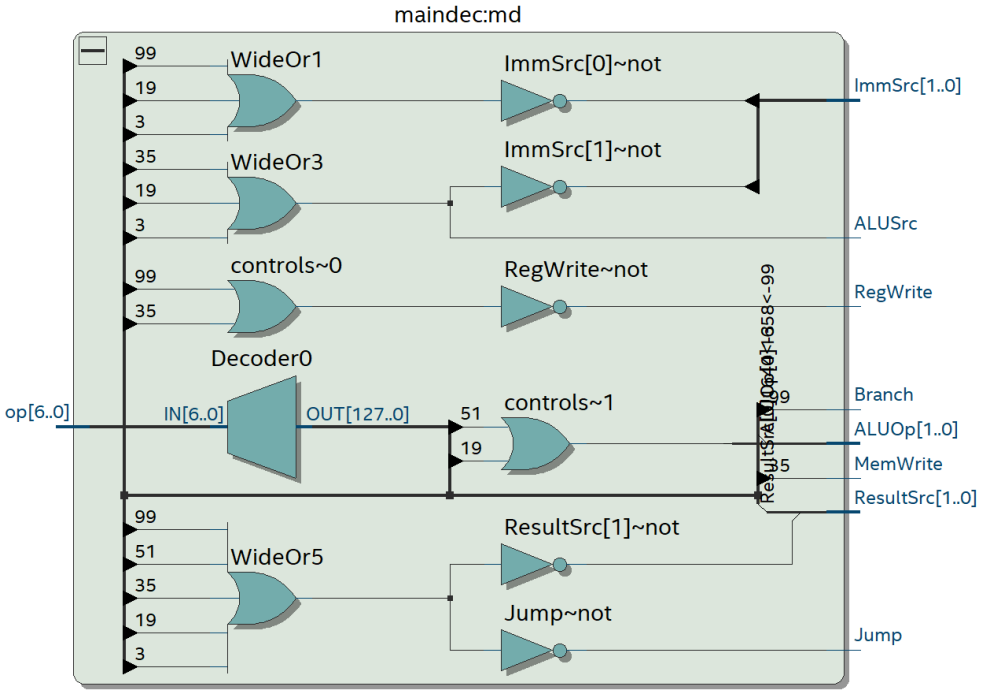
(ماژول riscvsingle)



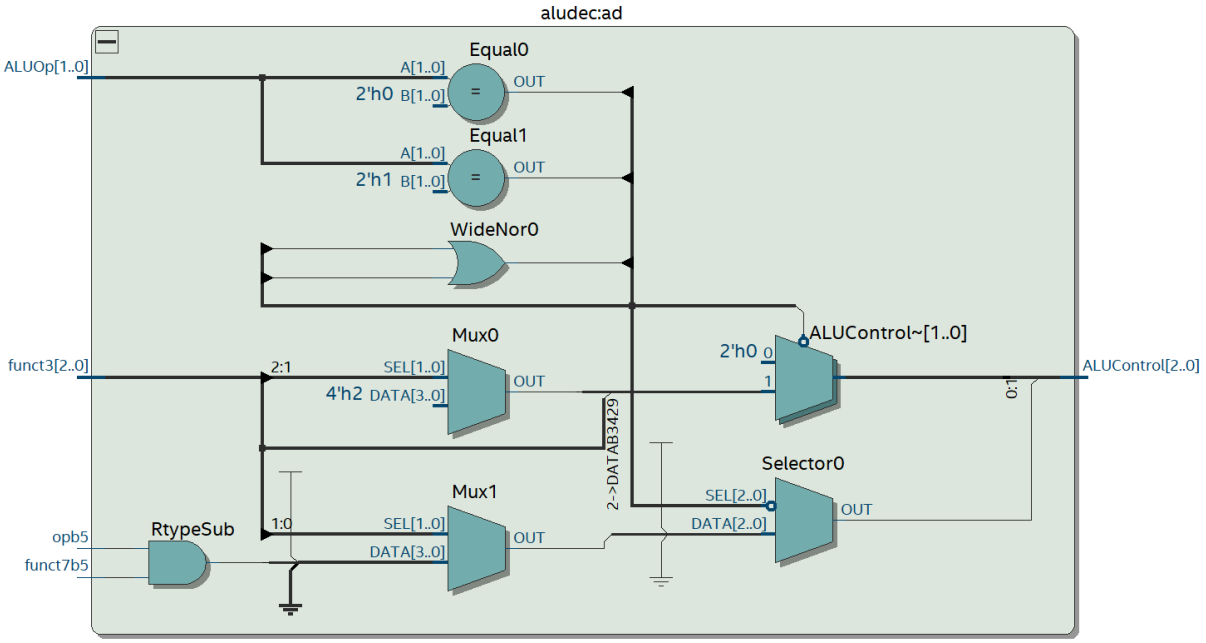
(ماژول controller)



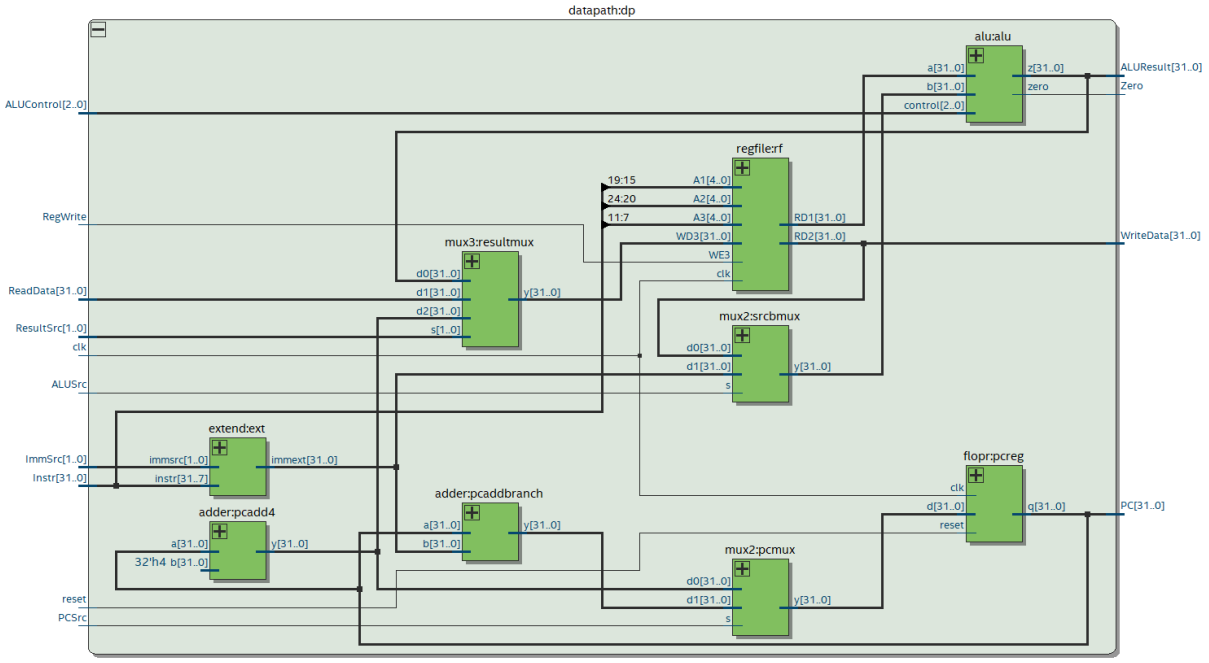
(ماژول maindec)



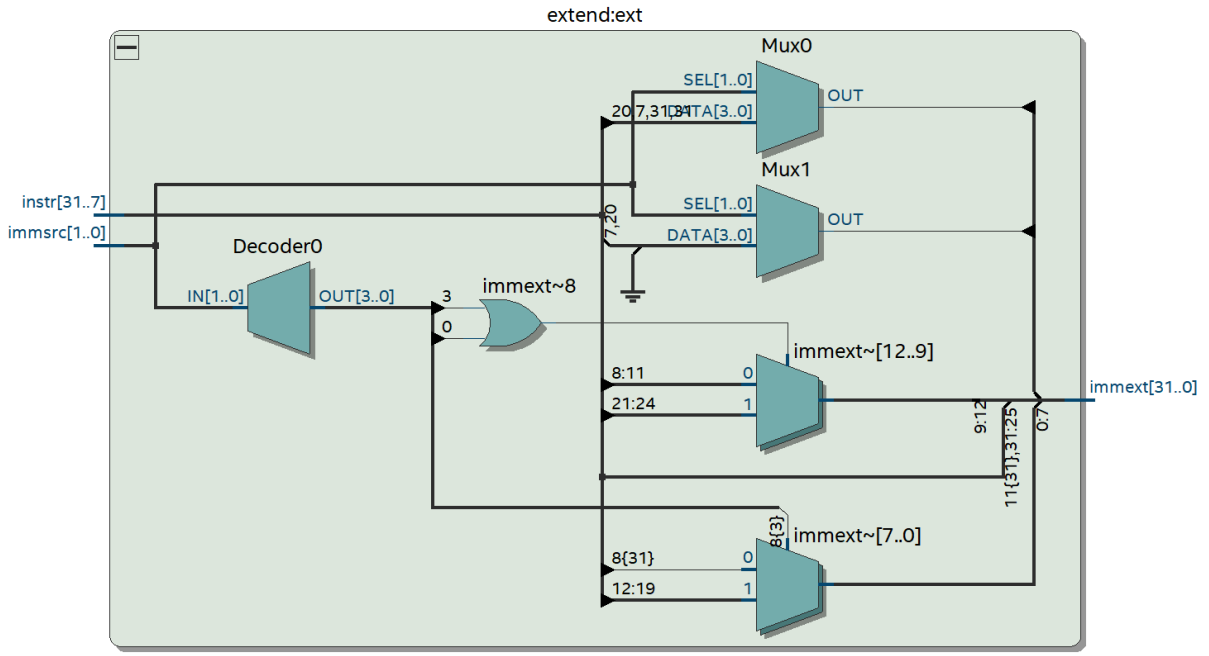
(ماژول aludec)



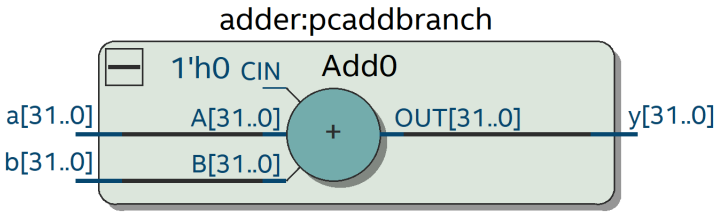
(ماژول datapath)



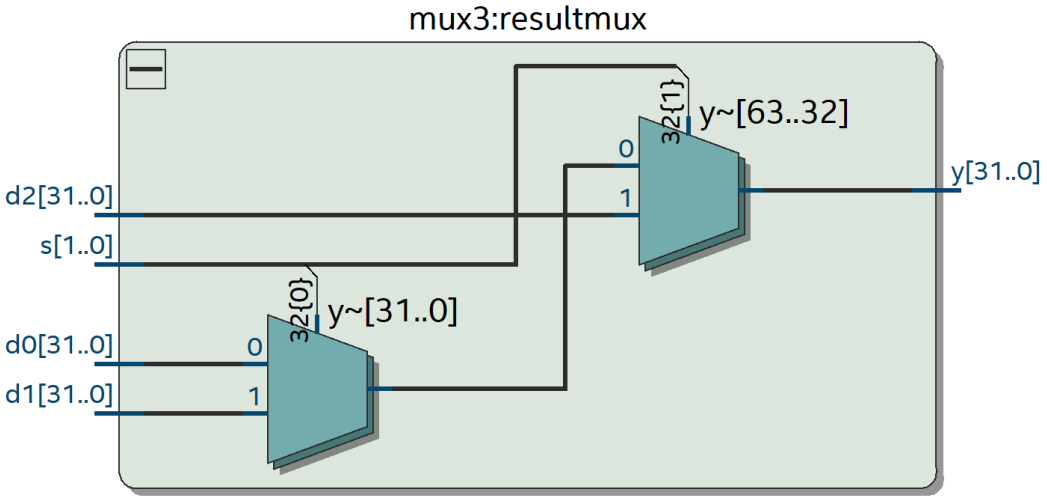
(ماژول extend)



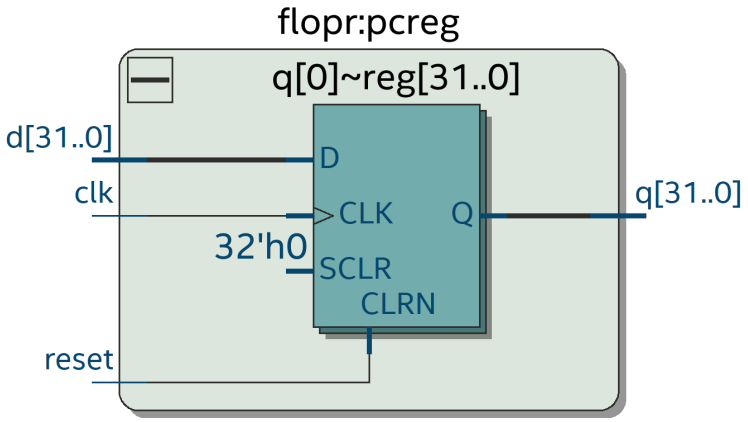
(ماژول adder)



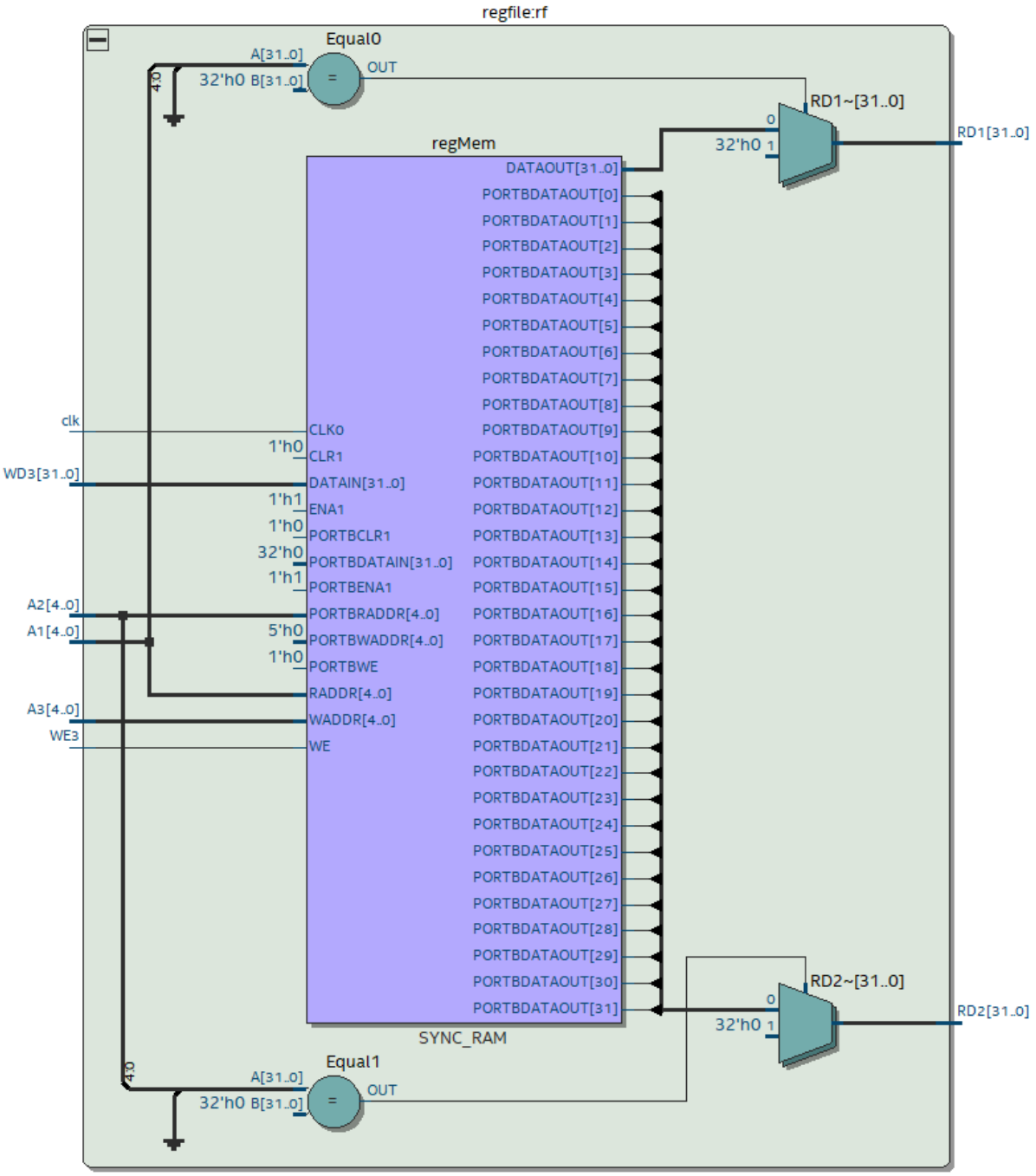
(ماژول mux3)



(ماژول flopr)



(ماژول regfile)



سوال ۱ - ب)

با استفاده از گزارش Fmax summary، حداکثر فرکانس ممکن مدار را به دست می‌آوریم.

Slow 1200mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	52.25 MHz	52.25 MHz	clk	

حداکثر فرکانس، برابر 52.25MHz خواهد بود.

همچنین مسیر بحرانی از گزارش Path در Quartus به دست می‌آید.

Slow 1200mV 85C Model

Command Info | Summary of Paths

	Delay	From Node	To Node
1	20.497	riscvsingle:rvsingle|datapath:dp|flopr:pcreg|q[5]	DataAdr[0]

مسیر بحرانی به صورت زیر است.

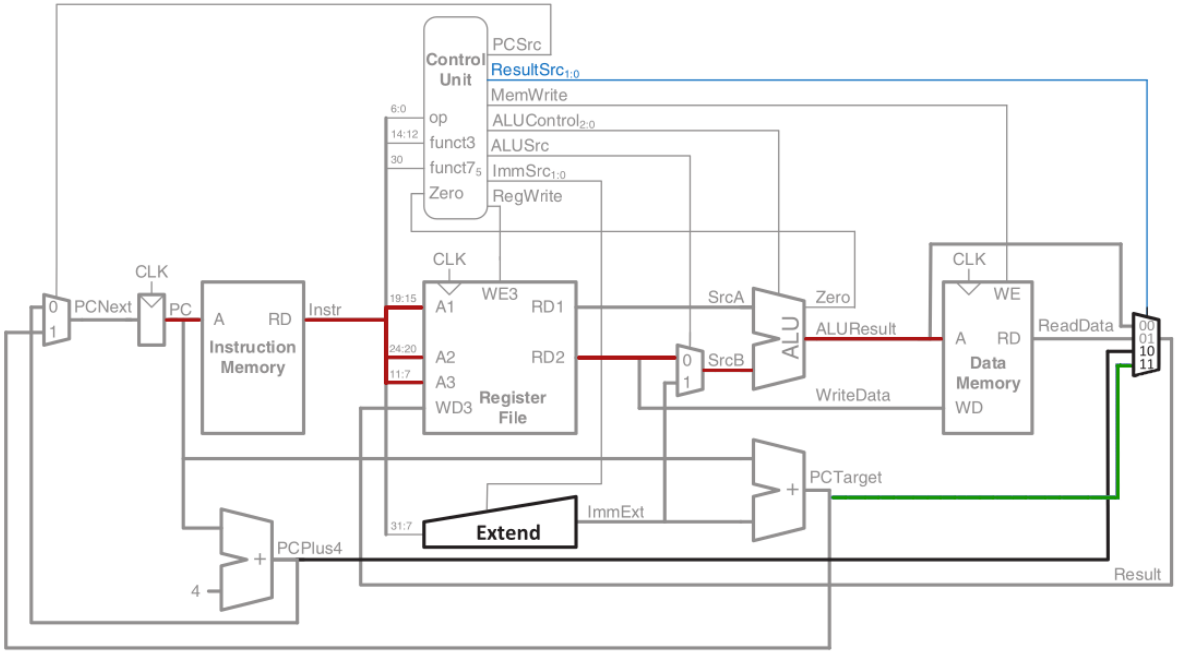
Path #1: Delay is 20.497

Path Summary | Statistics | Data Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	⌄ 20.497	20.497					data path
1	0.000	0.000			1	FF_X55_Y26_N23	riscvsingle:rvsingle|datapath:dp|flopr:pcreg|q[5]
2	0.000	0.000	FF	CELL	40	FF_X55_Y26_N23	rvsingle|dp|pcreg|q[5]|q
3	0.488	0.488	FF	IC	1	LCCOMB_X55_Y26_N16	imem|RAM~10|dataa
4	0.912	0.424	FF	CELL	2	LCCOMB_X55_Y26_N16	imem|RAM~10|combout
5	1.163	0.251	FF	IC	1	LCCOMB_X55_Y26_N2	imem|RAM~11|datad
6	1.288	0.125	FF	CELL	234	LCCOMB_X55_Y26_N2	imem|RAM~11|combout
7	2.756	1.468	FF	IC	1	LCCOMB_X50_Y23_N10	rvsingle|dp|rf|regMem~1432|dataa
8	3.180	0.424	FF	CELL	1	LCCOMB_X50_Y23_N10	rvsingle|dp|rf|regMem~1432|combout
9	3.940	0.760	FF	IC	1	LCCOMB_X50_Y24_N14	rvsingle|dp|rf|regMem~1433|datab
10	4.344	0.404	FF	CELL	1	LCCOMB_X50_Y24_N14	rvsingle|dp|rf|regMem~1433|combout
11	7.341	2.997	FF	IC	1	LCCOMB_X55_Y27_N26	rvsingle|dp|rf|regMem~1434|datab
12	7.745	0.404	FF	CELL	1	LCCOMB_X55_Y27_N26	rvsingle|dp|rf|regMem~1434|combout
13	7.980	0.235	FF	IC	1	LCCOMB_X55_Y27_N4	rvsingle|dp|rf|regMem~1437|datac
14	8.241	0.261	FR	CELL	1	LCCOMB_X55_Y27_N4	rvsingle|dp|rf|regMem~1437|combout
15	8.641	0.400	RR	IC	1	LCCOMB_X56_Y27_N2	rvsingle|dp|rf|RD2[20]~20|datab
16	9.043	0.402	RR	CELL	67	LCCOMB_X56_Y27_N2	rvsingle|dp|rf|RD2[20]~20|combout
17	10.348	1.305	RR	IC	1	LCCOMB_X60_Y26_N14	rvsingle|dp|srcbmux|y[20]~11|datad
18	10.503	0.155	RR	CELL	2	LCCOMB_X60_Y26_N14	rvsingle|dp|srcbmux|y[20]~11|combout
19	11.286	0.783	RR	IC	1	LCCOMB_X60_Y24_N8	rvsingle|dp|alu|LessThan0~41|datab
20	11.758	0.472	RR	CELL	1	LCCOMB_X60_Y24_N8	rvsingle|dp|alu|LessThan0~41|cout
21	11.758	0.000	RR	IC	1	LCCOMB_X60_Y24_N10	rvsingle|dp|alu|LessThan0~43|cin
22	11.824	0.066	RF	CELL	1	LCCOMB_X60_Y24_N10	rvsingle|dp|alu|LessThan0~43|cout
23	11.824	0.000	FF	IC	1	LCCOMB_X60_Y24_N12	rvsingle|dp|alu|LessThan0~45|cin
24	11.890	0.066	FR	CELL	1	LCCOMB_X60_Y24_N12	rvsingle|dp|alu|LessThan0~45|cout
25	11.890	0.000	RR	IC	1	LCCOMB_X60_Y24_N14	rvsingle|dp|alu|LessThan0~47|cin
26	11.956	0.066	RF	CELL	1	LCCOMB_X60_Y24_N14	rvsingle|dp|alu|LessThan0~47|cout

27	11.956	0.000	FF	IC	1	LCCOMB_X60_Y24_N16	rvsingle|dp|alu|LessThan0~49|cin
28	12.022	0.066	FR	CELL	1	LCCOMB_X60_Y24_N16	rvsingle|dp|alu|LessThan0~49|cout
29	12.022	0.000	RR	IC	1	LCCOMB_X60_Y24_N18	rvsingle|dp|alu|LessThan0~51|cin
30	12.088	0.066	RF	CELL	1	LCCOMB_X60_Y24_N18	rvsingle|dp|alu|LessThan0~51|cout
31	12.088	0.000	FF	IC	1	LCCOMB_X60_Y24_N20	rvsingle|dp|alu|LessThan0~53|cin
32	12.154	0.066	FR	CELL	1	LCCOMB_X60_Y24_N20	rvsingle|dp|alu|LessThan0~53|cout
33	12.154	0.000	RR	IC	1	LCCOMB_X60_Y24_N22	rvsingle|dp|alu|LessThan0~55|cin
34	12.220	0.066	RF	CELL	1	LCCOMB_X60_Y24_N22	rvsingle|dp|alu|LessThan0~55|cout
35	12.220	0.000	FF	IC	1	LCCOMB_X60_Y24_N24	rvsingle|dp|alu|LessThan0~57|cin
36	12.286	0.066	FR	CELL	1	LCCOMB_X60_Y24_N24	rvsingle|dp|alu|LessThan0~57|cout
37	12.286	0.000	RR	IC	1	LCCOMB_X60_Y24_N26	rvsingle|dp|alu|LessThan0~59|cin
38	12.352	0.066	RF	CELL	1	LCCOMB_X60_Y24_N26	rvsingle|dp|alu|LessThan0~59|cout
39	12.352	0.000	FF	IC	1	LCCOMB_X60_Y24_N28	rvsingle|dp|alu|LessThan0~61|cin
40	12.418	0.066	FR	CELL	1	LCCOMB_X60_Y24_N28	rvsingle|dp|alu|LessThan0~61|cout
41	12.418	0.000	RR	IC	1	LCCOMB_X60_Y24_N30	rvsingle|dp|alu|LessThan0~62|cin
42	12.954	0.536	RR	CELL	1	LCCOMB_X60_Y24_N30	rvsingle|dp|alu|LessThan0~62|combout
43	13.364	0.410	RR	IC	1	LCCOMB_X59_Y24_N20	rvsingle|dp|alu|Add0~7|datac
44	13.651	0.287	RR	CELL	3	LCCOMB_X59_Y24_N20	rvsingle|dp|alu|Add0~7|combout
45	16.287	2.636	RR	IC	1	IOOBUF_X67_Y0_N23	DataAdr[0]~output|i
46	20.497	4.210	RR	CELL	1	IOOBUF_X67_Y0_N23	DataAdr[0]~output|o
47	20.497	0.000	RR	CELL	0	PIN_AA15	DataAdr[0]

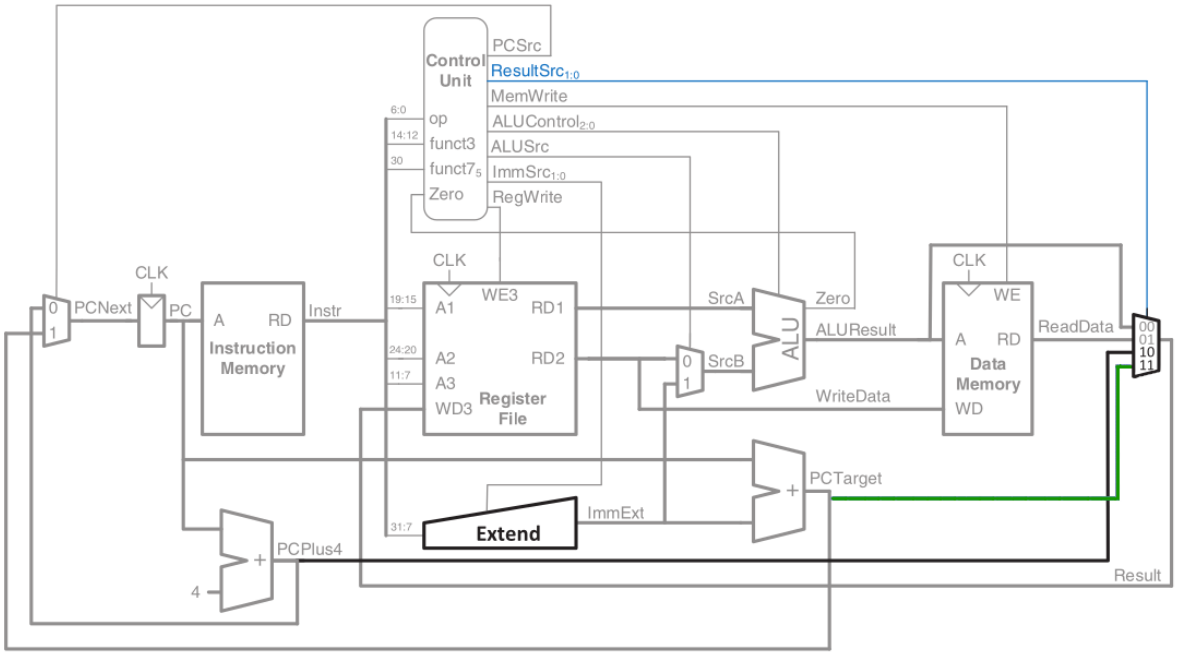
این مسیر، با خط قرمز روی شکل زیر نشان داده شده است.



این مسیر، وقتی استفاده می‌شود که بخواهیم با محتویات Data Memory کار کنیم. در نتیجه هنگام فراخوانی دستورات lw و sw، این مسیر استفاده می‌شود.

سوال ۱ - ج)

از تمرین قبلی، می‌دانیم که تغییرات روی datapath و جدول درستی main decoder و واحد extend به صورت زیر است.



ImmSrc	ImmExt
000	{{20{Instr[31]}}, Instr[31:20]}
001	{{20{Instr[31]}}, Instr[31:25], Instr[11:7]}
010	{{20{Instr[31]}}, Instr[7], Instr[30:25], Instr[11:8], 1'b0}
011	{{12{Instr[31]}}, Instr[19:12], Instr[20], Instr[30:21], 1'b0}
100	**{Instr[31:12], 12'b0}**

Instruction	Opcode	RegWrite	ImmSrc	ALUSrc	MemWrite	ResultSrc	Branch	ALUOp	Jump
lw	0000011	1	000	1	0	01	0	00	0
sw	0100011	0	001	1	1	xx	0	00	0
R-type	0110011	1	xxx	0	0	00	0	10	0
beq	1100011	0	010	0	0	xx	1	01	0
I-type ALU	0010011	1	000	1	0	00	0	10	0
jal	1101111	1	011	x	0	10	0	xx	1
auipc	**0010111**	**1**	**100**	**x**	**0**	**11**	**0**	**xx**	**0**

حال تغییرات بالا را روی کد اعمال می‌کنیم. ابتدا یک ماژول mux4 درست می‌کنیم، که یک multiplexer با ۴ ورودی است. کد این ماژول به صورت زیر است.

```
module mux4 #(parameter WIDTH = 8)
   (input logic [WIDTH-1:0] d0, d1, d2, d3,
    input logic [1:0] s,
    output logic [WIDTH-1:0] y);

   assign y = s[1] ? (s[0] ? d3 : d2) : (s[0] ? d1 : d0);
endmodule
```

همچنین برای اضافه کردن ردیف جدید به جدول درستی واحد extend، کافی‌ست ردیف جدیدی به قسمت case این ماژول اضافه کنیم (این تغییرات در خط ۱۹ و ۲۰ انجام شده‌اند.)

```
module extend(input logic [31:7] instr,
    input logic [2:0] immsrc,
    output logic [31:0] immext);

    always_comb
       case(immsrc)
          // Itype
          3'b000: immext = {{20{instr[31]}}, instr[31:20]};

          // Stype (stores)
          3'b001: immext = {{20{instr[31]}}, instr[31:25], instr[11:7]};

          // Btype (branches)
          3'b010: immext = {{20{instr[31]}}, instr[7], instr[30:25], instr[11:8], 1'b0};

          // Jtype (jal)
          3'b011: immext = {{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0};

          // Upper Immediate
          3'b100: immext = {instr[31:12], 12'b0};

          default: immext = 32'bx; // undefined
       endcase
endmodule
```

همین کار را برای اضافه کردن ردیف جدید به جدول درستی واحد main decoder انجام می‌دهیم (خط ۲۲).

```
module maindec(input logic [6:0] op,
    output logic [1:0] ResultSrc,
    output logic MemWrite,
    output logic Branch, ALUSrc,
    output logic RegWrite, Jump,
    output logic [2:0] ImmSrc,
    output logic [1:0] ALUOp);

    logic [11:0] controls;

    assign {RegWrite, ImmSrc, ALUSrc, MemWrite,
       ResultSrc, Branch, ALUOp, Jump} = controls;
    always_comb
       case(op)
          // RegWrite_ImmSrc_ALUSrc_MemWrite_ResultSrc_Branch_ALUOp_Jump
          7'b0000011: controls = 12'b1_000_1_0_01_0_00_0; // lw
          7'b0100011: controls = 12'b0_001_1_1_00_0_00_0; // sw
          7'b0110011: controls = 12'b1_xxx_0_0_00_0_10_0; // Rtype
          7'b1100011: controls = 12'b0_010_0_0_00_1_01_0; // beq
          7'b0010011: controls = 12'b1_000_1_0_00_0_10_0; // Itype ALU
          7'b1101111: controls = 12'b1_011_0_0_10_0_00_1; // jal
          7'b0010111: controls = 12'b1_100_x_0_11_0_xx_0; // auipc
          default: controls = 11'bx_xx_x_x_xx_x_xx_x; // ???
       endcase
endmodule
```

سپس ۳ بیتی شدن ImmSrc را در همه ماژول‌های مربوطه اعمال می‌کنیم. به طور خاص، ماژول‌های زیر تغییر می‌کنند:

- Extend
- maindec
- datapath
- controller
- riscvsingle

حال تغییرات مشخص شده در شکل را در کد datapath اعمال می‌کنیم (خط ۳۱).

```
1  module datapath(input logic clk, reset,
2      input logic [1:0] ResultSrc,
3      input logic PCSrc, ALUSrc,
4      input logic RegWrite,
5      input logic [2:0] ImmSrc,
6      input logic [2:0] ALUControl,
7      output logic Zero,
8      output logic [31:0] PC,
9      input logic [31:0] Instr,
10     output logic [31:0] ALUResult, WriteData,
11     input logic [31:0] ReadData);
12
13     logic [31:0] PCNext, PCPlus4, PCTarget;
14     logic [31:0] ImmExt;
15     logic [31:0] SrcA, SrcB;
16     logic [31:0] Result;
17
18     // next PC logic
19     flopr #(32) pcreg(clk, reset, PCNext, PC);
20     adder pcadd4(PC, 32'd4, PCPlus4);
21     adder pcaddbranch(PC, ImmExt, PCTarget);
22     mux2 #(32) pcmux(PCPlus4, PCTarget, PCSrc, PCNext);
23
24     // register file logic
25     regfile rf(clk, RegWrite, Instr[19:15], Instr[24:20], Instr[11:7], Result, SrcA, WriteData);
26     extend ext(Instr[31:7], ImmSrc, ImmExt);
27
28     // ALU logic
29     mux2 #(32) srcbmux(WriteData, ImmExt, ALUSrc, SrcB);
30     alu alu(SrcA, SrcB, ALUControl, ALUResult, Zero);
31     mux4 #(32) resultmux(ALUResult, ReadData, PCPlus4, PCTarget, ResultSrc, Result);
32 endmodule
```

حال تست‌بنچ را نیز تغییر می‌دهیم تا دستور اضافه شده را نیز تست کند. کد جدید به صورت زیر خواهد بود.

```
main:
addi x2, x0, 5
addi x3, x0, 12
addi x7, x3, -9
or x4, x7, x2
and x5, x3, x4
add x5, x5, x4
beq x5, x7, end
slt x4, x3, x4
beq x4, x0, around
addi x5, x0, 0
around:
slt x4, x7, x2
add x7, x4, x5
sub x7, x7, x2
```

```
sw x7, 84(x3)
lw x2, 96(x0)
add x9, x2, x5
jal x3, end
addi x2, x0, 1
end:
auipc x2, 1  # x2 = 0x1000 + PC = 0x1000 + 0x48 = 0x1048
sw x2, 0x20(x3)
done:
beq x2, x2, done
```

خط مشخص شده، با دستور auipc جایگزین شده است. در صورت کارکرد درست، این دستور باید مقدار 0x1048 را همانطور که در کامنت بالا مشخص شده در x2 قرار دهد.

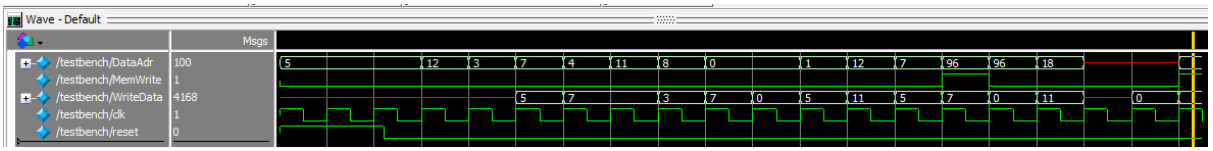
کد ماشین دستور اضافه شده نیز با توجه به ضمیمه مرجع هریس برابر 0x00001117 است. این کد را در فایل riscvtest.txt قرار می‌دهیم.

شرط درستی تست‌بنچ را نیز با توجه به توضیحات بالا تغییر می‌دهیم.

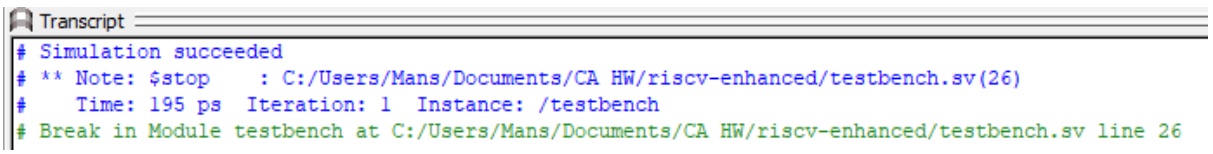
```
if(MemWrite) begin
    if(DataAdr === 100 & WriteData === 'h1048) begin
        $display("Simulation succeeded");
        $stop;
    end else if (DataAdr !== 96) begin
        $display("Simulation failed");
        $stop;
    end
end
```

سوال ۱ - د)

شبیه‌سازی را اجرا می‌کنیم. شکل موج حاصل به صورت زیر است.



همچنین چاپ شدن پیام زیر نشانه درستی کارکرد مدار است.



با استفاده از گزارش Fmax summary، حداکثر فرکانس ممکن مدار را به دست می‌آوریم.

Slow 1200mV 85C Model

	Fmax	Restricted Fmax	Clock Name	Note
1	56.77 MHz	56.77 MHz	clk	

مقدار حداکثر فرکانس برابر 56.77MHz خواهد بود.

همچنین مسیر بحرانی، از گزارش Path در Quartus به دست می‌آید.

Slow 1200mV 85C Model

Command Info | Summary of Paths

	Delay	From Node	To Node
1	21.767	riscvsingle:rvsingle|datapath:dp|flopr:pcreg|q[4]	DataAdr[0]

با توجه به نتیجه گزارش، مسیر بحرانی به صورت زیر خواهد بود.

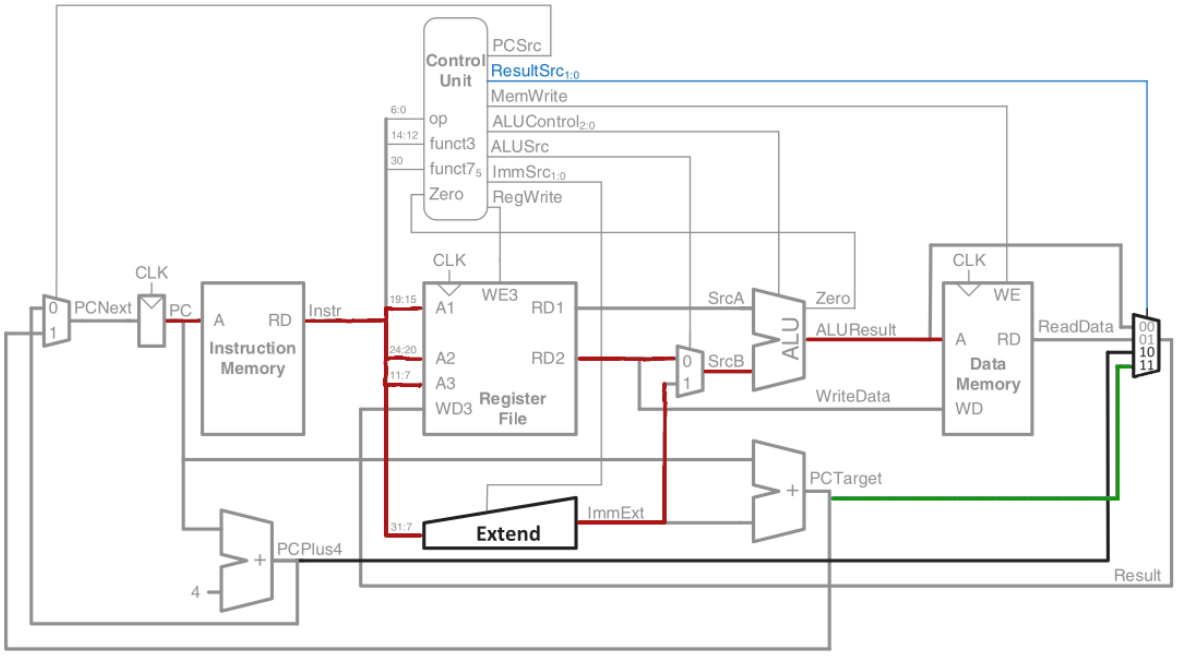
Path #1: Delay is 21.767

Path Summary | Statistics | Data Path

	Total	Incr	RF	Type	Fanout	Location	Element
1	0.000	0.000			1	FF_X60_Y22_N17	riscvsingle:rvsingle|datapath:dp|flopr:pcreg|q[4]
2	0.000	0.000	RR	CELL	61	FF_X60_Y22_N17	rvsingle|dp|pcreg|q[4]|q
3	0.806	0.806	RR	IC	1	LCCOMB_X59_Y22_N24	imem|RAM~4|dataa
4	1.234	0.428	RF	CELL	1	LCCOMB_X59_Y22_N24	imem|RAM~4|combout
5	1.461	0.227	FF	IC	1	LCCOMB_X59_Y22_N18	rvsingle|dp|ext|Mux11~8|datad
6	1.611	0.150	FR	CELL	233	LCCOMB_X59_Y22_N18	rvsingle|dp|ext|Mux11~8|combout
7	3.352	1.741	RR	IC	1	LCCOMB_X50_Y19_N6	rvsingle|dp|rf|regMem~1417|dataa
8	3.780	0.428	RF	CELL	1	LCCOMB_X50_Y19_N6	rvsingle|dp|rf|regMem~1417|combout
9	4.213	0.433	FF	IC	1	LCCOMB_X49_Y19_N26	rvsingle|dp|rf|regMem~1418|dataa
10	4.637	0.424	FF	CELL	1	LCCOMB_X49_Y19_N26	rvsingle|dp|rf|regMem~1418|combout
11	5.682	1.045	FF	IC	1	LCCOMB_X53_Y20_N26	rvsingle|dp|rf|regMem~1419|datab
12	6.038	0.356	FF	CELL	1	LCCOMB_X53_Y20_N26	rvsingle|dp|rf|regMem~1419|combout
13	7.443	1.405	FF	IC	1	LCCOMB_X63_Y23_N10	rvsingle|dp|rf|RD2[13]~13|datac
14	7.724	0.281	FF	CELL	67	LCCOMB_X63_Y23_N10	rvsingle|dp|rf|RD2[13]~13|combout
15	8.486	0.762	FF	IC	1	LCCOMB_X63_Y22_N22	rvsingle|dp|srcbmux|y[13]~38|datad
16	8.611	0.125	FF	CELL	2	LCCOMB_X63_Y22_N22	rvsingle|dp|srcbmux|y[13]~38|combout
17	10.011	1.400	FF	IC	1	LCCOMB_X65_Y22_N26	rvsingle|dp|alu|LessThan0~27|datab
18	10.520	0.509	FR	CELL	1	LCCOMB_X65_Y22_N26	rvsingle|dp|alu|LessThan0~27|cout
19	10.520	0.000	RR	IC	1	LCCOMB_X65_Y22_N28	rvsingle|dp|alu|LessThan0~29|cin
20	10.586	0.066	RF	CELL	1	LCCOMB_X65_Y22_N28	rvsingle|dp|alu|LessThan0~29|cout
21	10.586	0.000	FF	IC	1	LCCOMB_X65_Y22_N30	rvsingle|dp|alu|LessThan0~31|cin
22	10.652	0.066	FR	CELL	1	LCCOMB_X65_Y22_N30	rvsingle|dp|alu|LessThan0~31|cout
23	10.652	0.000	RR	IC	1	LCCOMB_X65_Y21_N0	rvsingle|dp|alu|LessThan0~33|cin
24	10.718	0.066	RF	CELL	1	LCCOMB_X65_Y21_N0	rvsingle|dp|alu|LessThan0~33|cout

25	10.718	0.000	FF	IC	1	LCCOMB_X65_Y21_N2	rvsingle|dp|alu|LessThan0~35|cin
26	10.784	0.066	FR	CELL	1	LCCOMB_X65_Y21_N2	rvsingle|dp|alu|LessThan0~35|cout
27	10.784	0.000	RR	IC	1	LCCOMB_X65_Y21_N4	rvsingle|dp|alu|LessThan0~37|cin
28	10.850	0.066	RF	CELL	1	LCCOMB_X65_Y21_N4	rvsingle|dp|alu|LessThan0~37|cout
29	10.850	0.000	FF	IC	1	LCCOMB_X65_Y21_N6	rvsingle|dp|alu|LessThan0~39|cin
30	10.916	0.066	FR	CELL	1	LCCOMB_X65_Y21_N6	rvsingle|dp|alu|LessThan0~39|cout
31	10.916	0.000	RR	IC	1	LCCOMB_X65_Y21_N8	rvsingle|dp|alu|LessThan0~41|cin
32	10.982	0.066	RF	CELL	1	LCCOMB_X65_Y21_N8	rvsingle|dp|alu|LessThan0~41|cout
33	10.982	0.000	FF	IC	1	LCCOMB_X65_Y21_N10	rvsingle|dp|alu|LessThan0~43|cin
34	11.048	0.066	FR	CELL	1	LCCOMB_X65_Y21_N10	rvsingle|dp|alu|LessThan0~43|cout
35	11.048	0.000	RR	IC	1	LCCOMB_X65_Y21_N12	rvsingle|dp|alu|LessThan0~45|cin
36	11.114	0.066	RF	CELL	1	LCCOMB_X65_Y21_N12	rvsingle|dp|alu|LessThan0~45|cout
37	11.114	0.000	FF	IC	1	LCCOMB_X65_Y21_N14	rvsingle|dp|alu|LessThan0~47|cin
38	11.180	0.066	FR	CELL	1	LCCOMB_X65_Y21_N14	rvsingle|dp|alu|LessThan0~47|cout
39	11.180	0.000	RR	IC	1	LCCOMB_X65_Y21_N16	rvsingle|dp|alu|LessThan0~49|cin
40	11.246	0.066	RF	CELL	1	LCCOMB_X65_Y21_N16	rvsingle|dp|alu|LessThan0~49|cout
41	11.246	0.000	FF	IC	1	LCCOMB_X65_Y21_N18	rvsingle|dp|alu|LessThan0~51|cin
42	11.312	0.066	FR	CELL	1	LCCOMB_X65_Y21_N18	rvsingle|dp|alu|LessThan0~51|cout
43	11.312	0.000	RR	IC	1	LCCOMB_X65_Y21_N20	rvsingle|dp|alu|LessThan0~53|cin
44	11.378	0.066	RF	CELL	1	LCCOMB_X65_Y21_N20	rvsingle|dp|alu|LessThan0~53|cout
45	11.378	0.000	FF	IC	1	LCCOMB_X65_Y21_N22	rvsingle|dp|alu|LessThan0~55|cin
46	11.444	0.066	FR	CELL	1	LCCOMB_X65_Y21_N22	rvsingle|dp|alu|LessThan0~55|cout
47	11.444	0.000	RR	IC	1	LCCOMB_X65_Y21_N24	rvsingle|dp|alu|LessThan0~57|cin
48	11.510	0.066	RF	CELL	1	LCCOMB_X65_Y21_N24	rvsingle|dp|alu|LessThan0~57|cout
49	11.510	0.000	FF	IC	1	LCCOMB_X65_Y21_N26	rvsingle|dp|alu|LessThan0~59|cin
50	11.576	0.066	FR	CELL	1	LCCOMB_X65_Y21_N26	rvsingle|dp|alu|LessThan0~59|cout
51	11.576	0.000	RR	IC	1	LCCOMB_X65_Y21_N28	rvsingle|dp|alu|LessThan0~61|cin
52	11.642	0.066	RF	CELL	1	LCCOMB_X65_Y21_N28	rvsingle|dp|alu|LessThan0~61|cout
53	11.642	0.000	FF	IC	1	LCCOMB_X65_Y21_N30	rvsingle|dp|alu|LessThan0~62|cin
54	12.076	0.434	FF	CELL	1	LCCOMB_X65_Y21_N30	rvsingle|dp|alu|LessThan0~62|combout
55	13.221	1.145	FF	IC	1	LCCOMB_X60_Y21_N6	rvsingle|dp|alu|Add0~7|datad
56	13.346	0.125	FF	CELL	3	LCCOMB_X60_Y21_N6	rvsingle|dp|alu|Add0~7|combout
57	17.566	4.220	FF	IC	1	IOOBUF_X115_Y22_N2	DataAdr[0]~output|i
58	21.767	4.201	FF	CELL	1	IOOBUF_X115_Y22_N2	DataAdr[0]~output|o
59	21.767	0.000	FF	CELL	0	PIN_U23	DataAdr[0]

با توجه به داده‌های بالا، مسیر بحرانی روی شکل به صورت زیر است.



مشابه سوال ۱-ب، این مسیر نیز توسط دستورات lw و sw استفاده می‌شود.

سوال 2)

این تکه کد در ابتدا، شامل دستور I-TYPE ALU INSTRUCTIONS است. دستور های I-TYPE ALU INSTRUCTIONS در 4 سیکل انجام می‌شوند که این چهار سیکل به ترتیب عبارت اند از:

1. Fetch
2. Decode
3. ExecuteI
4. ALUWB

پس انجام این 3 دستور، 12 سیکل به طول می‌انجامد.

سپس حلقه داریم. شرط حلقه این است که s0، مساوی با t3 شود یا به عبارتی s0، مساوی با 10 شود.

اجرای شرط حلقه 3 سیکل به طول می‌انجامد که شامل مراحل زیر می‌شود:

1. Fetch
2. Decode
3. BEQ

در بدنه حلقه با استفاده از دو دستور I-TYPE ALU INSTRUCTION، هر بار s1 با s0 جمع می‌شود و s0 یک واحد بیشتر می‌شود.هر کدام از این دستور ها 4 سیکل به طول می‌انجامند. همچنین یک دستور J imm هم داریم که معادل jal x0, imm می‌باشد که در 4 سیکل اجرا می‌شود که عبارت است از:

1. Fetch
2. Decode
3. Jal
4. ALUWB

با توجه به موارد گفته شده یعنی، اجرای بدنه حلقه، 12 سیکل طول می‌کشد.

در نهایت با توجه به شرط حلقه، بدنه حلقه 10 بار اجرا می‌شود. شرط حلقه هم 11 بار اجرا می‌شود.

4 + 4 + 4 + 11*(3) + 10*(12) = 165

165 سیکل طول می‌کشد تا این برنامه اجرا شود.

با توجه به توضیحات داده شده، قبل از حلقه 3 دستور I-TYPE ALU INSTRUCTION و در حلقه 2 دستور I-TYPE ALU INSTRUCTION داریم. پس با اجرای 10 بار حلقه، 23 دستور I-TYPE ALU INSTRUCTION داریم. همچنین شرط حلقه 11 بار اجرا می‌شود پس 11 دستور BEQ TYPE داریم. با 10 بار اجرای حلقه 10 بار دستور j LOOOP اجرا می‌شود.

پس تعداد کل دستور ها عبارت است از:

Number of instructions = 23 + 11 + 10 = 44

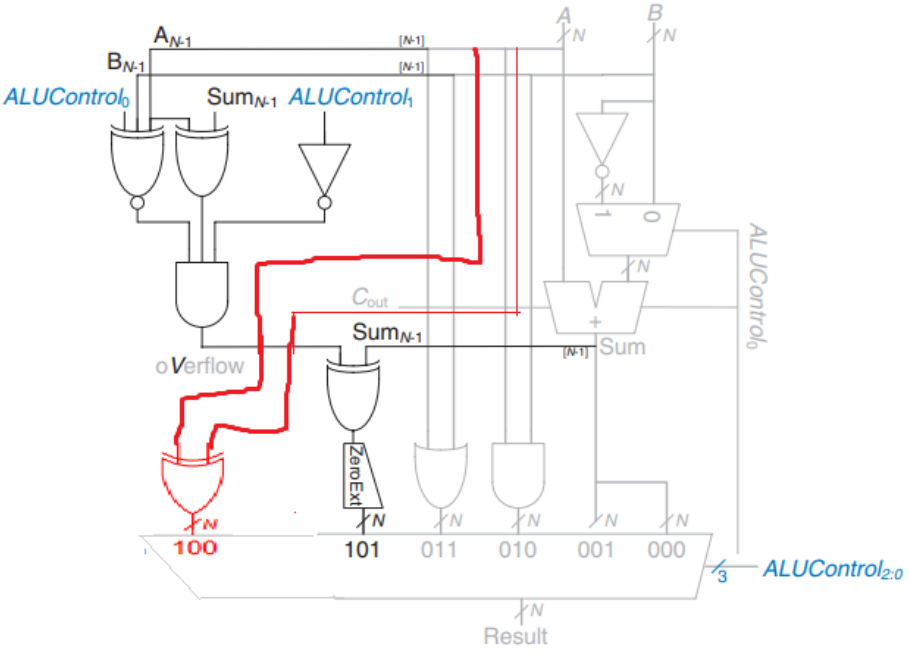
میانگین CPI مجموع وزنی CPI هر دستور، ضرب در کسری از زمان که آن دستور مصرف می‌کند، است.

نوع دستور	تعداد سیکل مورد نیاز برای هربار اجرا	تعداد	درصد نسبت به کل
I-TYPE ALU INSTRUCTION	4	23	0.5227
BEQ	3	11	0.2500
JAL	4	10	0.2273

average CPI = (0.5227*4) + (0.2500 * 3) + (0.2273 * 4) = 3.75

سوال3)

الف) می‌دانیم که دستور xor از نوع R-TYPE است. datapath وMain FSM نیازی به اعمال تغییری ندارند. فقط ALU باید قادر باشد تا XOR را محاسبه کند. نیازمندیم که به مالتی پلکسر ALUControl حالت XOR را نیز اضافه کنیم. تغییراتی که باید در شکل 5.18 که نشان دهنده یک ALU است، اعمال کنیم در شکل زیر نشان داده شده است.



همانطور که گفته شد مسیر داده دچار تغییری نمی شود فقط Alu قابلیت جدید پیدا می‌کند:

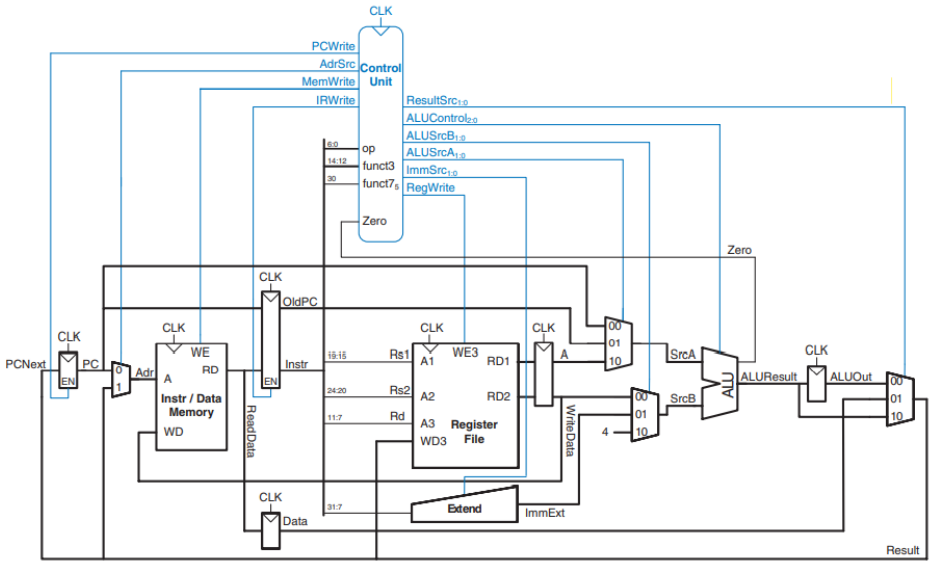
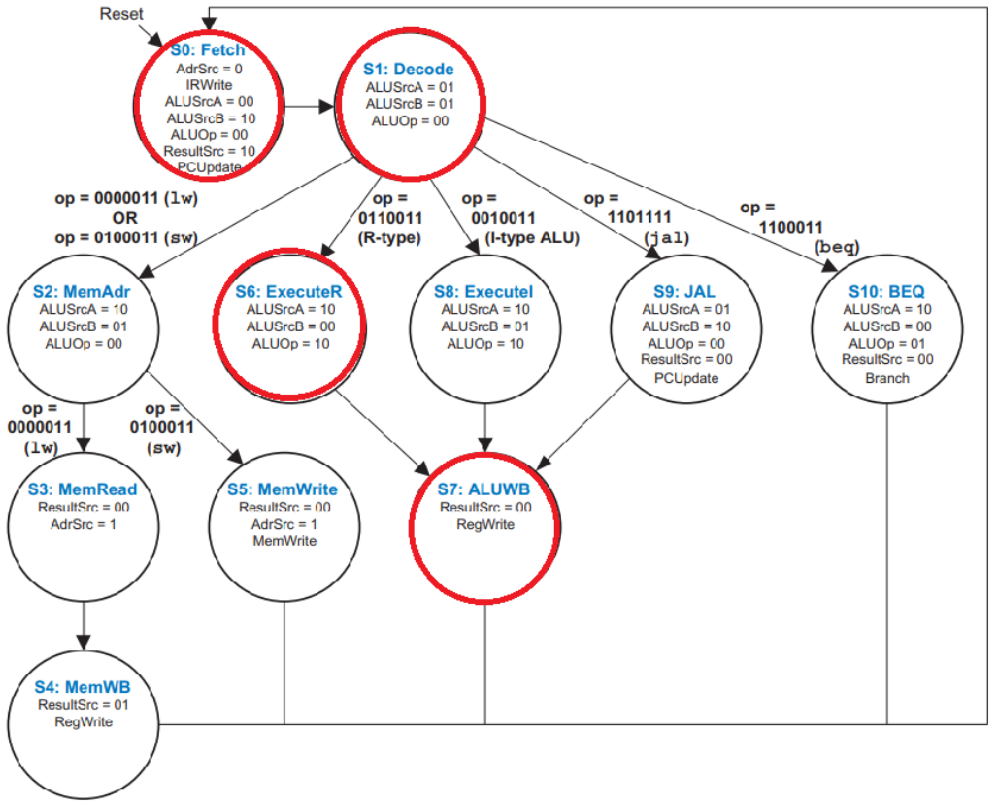


Figure 7.27 Complete multicycle processor

ب) به سیگنال کنترلی جدیدی نیاز نداریم. همان سایکل R-TYPE برای xor هم اجرا می‌شود.(بخش مشخص شده با نوارهای قرمز)



ج) به سیگنال کنترلی جدیدی نیاز نداریم فقط نیاز است که جداول ALU Decoder وALU Operations را آپدیت کنیم.

Table 7.3 ALU Decoder truth table

ALUOp	funct3	$\{op_5, funct7_5\}$	ALUControl	Instruction
00	x	x	000 (add)	`lw, sw`
01	x	x	001 (subtract)	`beq`
10	000	00, 01, 10	000 (add)	`add`
	000	11	001 (subtract)	`sub`
	010	x	101 (set less than)	`slt`
	110	x	011 (or)	`or`
	111	x	010 (and)	`and`
	100	x	100(xor)	xor/xori

Table 5.1 ALU operations

ALUControl 2:0	Function
0 00	Add
001	Subtract
0 10	AND
0 11	OR
1 0 0	XOR