# 单周期 MIPS 处理器实验报告

### BobAnkh

## 1、实验目的:

完成单周期 CPU,综合实现并通过后仿真。

## 2、实验内容说明:

本次实验在数逻理论课大作业的基础上进行修改,仿真过程中执行的是大作业第 3 节给出的汇编程序,同时\$a0 的值采用的是目前数逻大作业的 4,而非实验指导书中的 3。在数逻大作业中,主要修改了 Control.v 和 InstructionMemory.v 并通过了行为级仿真;而在本实验中,为了避免因缺少输出而工具将大量电路优化掉,所以修改了 RegisterFile.v 和 CPU.v 中的部分代码,利用 SW1 和 SW0 选择\$a0,\$v0,\$sp,\$ra 的低 8 位引出到 CPU 项层端口,并接在 LED7~LED0 上作为输出。同时为了较好地调试,将打平的优化关掉了,即将flatten\_hierarchy 属性置成了 none。

## 3、关键代码:

由于代码量较多,大作业主要完成的是 Control.v 和 InstructionMemory.v,所以就仅在附件中给出,这里仅给出核心的 CPU.v 和 RegisterFile.v,其余代码可以参见附件。

#### CPU.v:

```
module CPU(reset, clk, select, register low);
    input reset, clk;
   input [1:0] select;
   output [7:0] register_low;
   wire [7:0] a0, v0, sp, ra;
   assign register_low=(select==2'b00)?a0: (select == 2'b01)?v0: (sele
ct == 2'b10)?sp: (select == 2'b11)?ra:8'b0;
    reg [31:0] PC;
   wire [31:0] PC next;
   always @(posedge reset or posedge clk)
        if (reset)
            PC <= 32'h00000000;
        else
            PC <= PC next;
   wire [31:0] PC plus 4;
   assign PC_plus_4 = PC + 32'd4;
```

```
wire [31:0] Instruction;
    InstructionMemory instruction_memory1(.Address(PC), .Instruction(In
struction));
    wire [1:0] RegDst;
   wire [1:0] PCSrc;
   wire Branch;
   wire MemRead;
   wire [1:0] MemtoReg;
   wire [3:0] ALUOp;
   wire ExtOp;
   wire LuOp;
   wire MemWrite;
   wire ALUSrc1;
   wire ALUSrc2;
   wire RegWrite;
   Control control1(
        .OpCode(Instruction[31:26]), .Funct(Instruction[5:0]),
        .PCSrc(PCSrc), .Branch(Branch), .RegWrite(RegWrite), .RegDst(Re
gDst),
        .MemRead(MemRead), .MemWrite(MemWrite), .MemtoReg(MemtoReg),
        .ALUSrc1(ALUSrc1), .ALUSrc2(ALUSrc2), .ExtOp(ExtOp), .LuOp(LuOp
     .ALUOp(ALUOp));
    wire [31:0] Databus1, Databus2, Databus3;
   wire [4:0] Write_register;
    assign Write_register = (RegDst == 2'b00)? Instruction[20:16]: (Reg
Dst == 2'b01)? Instruction[15:11]: 5'b11111;
    RegisterFile register file1(.reset(reset), .clk(clk), .RegWrite(Reg
Write),
        .Read_register1(Instruction[25:21]), .Read_register2(Instruction[25:21])
n[20:16]), .Write_register(Write_register),
        .Write_data(Databus3), .Read_data1(Databus1), .Read_data2(Datab
us2), .a0(a0), .v0(v0), .sp(sp), .ra(ra));
   wire [31:0] Ext out;
    assign Ext_out = {ExtOp? {16{Instruction[15]}}: 16'h0000, Instructi
on[15:0]};
   wire [31:0] LU out;
    assign LU_out = LuOp? {Instruction[15:0], 16'h0000}: Ext_out;
   wire [4:0] ALUCtl;
```

```
wire Sign;
    ALUControl alu_control1(.ALUOp(ALUOp), .Funct(Instruction[5:0]), .A
LUCtl(ALUCtl), .Sign(Sign));
   wire [31:0] ALU in1;
   wire [31:0] ALU_in2;
   wire [31:0] ALU_out;
   wire Zero;
    assign ALU in1 = ALUSrc1? {17'h00000, Instruction[10:6]}: Databus1;
    assign ALU_in2 = ALUSrc2? LU_out: Databus2;
    ALU alu1(.in1(ALU_in1), .in2(ALU_in2), .ALUCt1(ALUCt1), .Sign(Sign)
, .out(ALU_out), .zero(Zero));
   wire [31:0] Read data;
    DataMemory data_memory1(.reset(reset), .clk(clk), .Address(ALU_out)
 .Write_data(Databus2), .Read_data(Read_data), .MemRead(MemRead), .Mem
Write(MemWrite));
    assign Databus3 = (MemtoReg == 2'b00)? ALU_out: (MemtoReg == 2'b01)
? Read_data: PC_plus_4;
    wire [31:0] Jump target;
    assign Jump_target = {PC_plus_4[31:28], Instruction[25:0], 2'b00};
   wire [31:0] Branch_target;
    assign Branch target = (Branch & Zero)? PC plus 4 + {LU out[29:0],
2'b00}: PC_plus_4;
    assign PC_next = (PCSrc == 2'b00)? Branch_target: (PCSrc == 2'b01)?
 Jump_target: Databus1;
endmodule
```

### RegisterFile.v:

```
module RegisterFile(reset, clk, RegWrite, Read_register1, Read_register
2, Write_register, Write_data, Read_data1, Read_data2, a0, v0, sp, ra);
   input reset, clk;
   input RegWrite;
   input [4:0] Read_register1, Read_register2, Write_register;
   input [31:0] Write_data;
   output [31:0] Read_data1, Read_data2;
   output [7:0] a0, v0, sp, ra;

reg [31:0] RF_data[31:1];
```

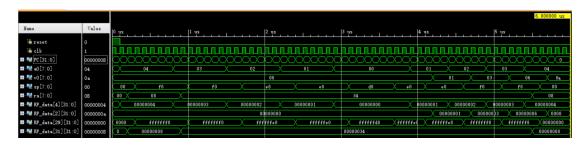
## 4、文件清单:

附件中,设计代码文件夹内有 ALU.v,ALUControl.v,Control.v,CPU.v,DataMemory.v,InstructionMemory.v,RegisterFile.v。testbench 和 xdc 约束文件的文件夹内有 x\_cpu.xdc 以及test\_cpu.v

# 5、仿真结果及分析:

### 前仿-行为仿真:

整体(从 0ns 运行到 6000ns):



上图中,复位信号 reset 和时钟信号 clk 往下,分别是 PC,\$a0 低八位,\$v0 低八位,\$sp 低八位,\$ra 低八位,\$a0,\$v0,\$sp,\$ra(因为前仿可以直接看寄存器堆中的对应寄存器,所以前仿直接把对应寄存器的波形展示出来)

# 局部放大(每张图分别为 1000ns 的运行时间):

Name	Value										1,000.000 ns
1 reset	0	0 ns	100 ns	200 ns	300 ns 4	00 ns 50	0 ns	600 ns	700 ns	800 ns	900 ns
Va clk  ■ PC[31:0]  ■ 0[7:0]	1 00000010 03	00000000	00000004	00000000	00000010	00000014	00000018	0000001c	0000002c	00000030	00000000
₩ w0[7:0] ₩ sp[7:0]	00 £0		00			0	0	f8			×
■ ¶ra[7:0] ■ ¶ RF_data[4][31:0]	34 00000003	00000000				00000004	08			00	34 000003
RF_data[2][31:0]			00000000			00000		fffffff8			
<b>Ⅲ-⋘</b> RF_data[31] [31:0]	00000034	00000	000				80000000				00000034
Name	Value										1, 999.480 ns
la reset	0	1,000 ns	1, 100 ns	1,200 ns	1,300 ns	1,400 ns 1,	500 ns	1,600 ns	1,700 ns	1,800 ns	1,900 ns
Walk □ W PC[31:0] □ Walo[7:0]	0 00000018 02	00000010	00000014	00000018	0000001c	00000020	00000030	0000000c	00000010	00000014	00000018
u w v0[7:0] u w v0[7:0] u w sp[7:0]	00 e8			03	f0	0	0		(	e8	
II-∰ ra[7:0] II-₩ RF_data[4][31:0]	34 00000002			00000003		3.	4		00000002		
⊒-₩ RF_data[2][31:0] ⊒-₩ RF_data[29][31:0]	00000000 ffffffe8				fffffff0	00000				ffffffe8	
<b>∐ ™</b> RF_data[31][31:0]	00000034					00000	1034				
Name	Value										2,999.680 ns
Wareset	0	2,000 ns	2, 100 ns	2,200 ns	2,300 ns	2,400 ns	2,500 ns	2,600 ns	2,700 ns	2,800 ns	2,900 ns
Walk ■ Marian PC[31:0] ■ Marian Republication   Marian Republicatio	00000030	00000010	0000002c	00000030	00000000	00000010	00000014	00000018	0000001c	00000020	00000030
a v0[7:0] a v0[7:0] a v sp[7:0]	00 00 e0		02	8		01	01		40		00
I-	34 00000000	0000		· ·		34	00000001		Ì		00000000
RF_data[2][31:0]	00000000 ffffffe0		ffff	fe8		00000			ffffffe0		
<b>Ⅲ-⋘</b> RF_data[31][31:0]	00000034					00000	034				
											3, 998. 620 ns
Nome la reset	Value 0	3,000 ns	3, 100 ns	3, 200 ns	3,300 ns	3,400 ns 3	,500 ns	3,600 ns	3,700 ns	3,800 ns	3,900 ns
% clk ■	0	00000000	00000010	00000014	00000018						
⊒-¶ a0[7:0] ⊒-¶ v0[7:0]	01					0000001c	00000020	00000024	00000028	00000034	00000038
	00					00	00000020	00000024	00000028		00000038
II	00 e0 34	e0	X		d	00 0 3 3	0	00000024	00000028	00000034 e0	01
3 -	00 e0 34 00000001 00000000		X		d	00 0 3 00000000	0	00000024	) 00000028	•0	
2 - M sp[7:0] 2 - M ra[7:0] 3 - M RF_data[4][31:0] 2 - M RF_data[2][31:0] 3 - M RF_data[29][31:0]	00 e0 34 00000001	e0	X		ferr	00 0 3 00000000	0 4	00000024	00000028		01
2 - M sp[7:0] 2 - M ra[7:0] 3 - M RF_data[4][31:0] 2 - M RF_data[2][31:0] 3 - M RF_data[29][31:0]	00 e0 34 00000001 00000000 ffffffe0		X		d feet	00 0 3 3 00000000 00000	0 4	00000024	00000028	•0	01
## sp[7:0] ## sp[7:0] ## sp_(7:0] ## sp_data[4][31:0] ## sp_data[2][31:0] ## sp_data[2][31:0] ## sp_data[2][31:0] ## sp_data[31][31:0]	00 e0 34 00000001 00000000 ffffffe0		4, 100 ns	4, 200 ns		00 0 15 3 000000000 00000 4f48 00000	0 4	00000024	00000028	e0	01
## sp[7:0] ## rs [7:0] ## rs [4:14] ## RF_data[4] [31:0] ## RF_data[2] [31:0] ## RF_data[29] [31:0] ## RF_data[31] [31:0]	00 e0 34 00000001 00000000 ffffffe0 00000034	fffffe0				00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 1000 1034	4, 800 ns	X	#0 ###################################	01 00000001 4,999 060 ns
### sp[7:0] ### sp[7:0] ### ### ### ### #### ################	00 e0 34 00000001 00000000 ffffffe0 00000034	ffffff+0	00000040	00000044	4,300 ns	00 0 15 3 000000000 00000 4f48 00000	0 4 0000		4,700 ns	e0	01 00000001 4,999.060 ns
### sp[7:0] ### sp[7:0] ### sp[7:0] ### RP_data[3][31:0] ### RP_data[2][31:0] ### RP_data[2][31:0] ### RP_data[3][51:0] ### RP_data[3][51:0] ### RP_data[3][51:0] #### sp[7:0] ### sp[7:0] ### sp[7:0] ### sp[7:0]	00 e0 34 00000001 000000000 ffffffe0 00000034 Value 0 0 00000038 03 03 60 34	fffffe0	00000040	00000044	4,300 ns	00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 1000 1034	4, 600 ns.	4,700 ns	# 00 e0 ffffffe0 ffffffe0 ffffffe0 ffffffe0 ffffffe0 ffffffe0 ffffffe0 ffffffe0 fffffffe0 fffffffe0 ffffffff	4,999.000 ns 4,900 ns 00000038
## sp[7:0] ## sp[7:0] ## sp[7:0] ## sp[7:0] ## sp[4sta[4][31:0] ## sp[4sta[2][31:0] ## sp[4sta[2][31:0] ## sp[4sta[3][31:0] ## sp[3sta[3][31:0] ## sp[3sta[3][31:0] ## sp[7:0]	00 e0 34 00000001 00000000 ffffffe0 00000034 Value 0 0 0 00000038 03 f0 34 00000003 00000003 00000003	######################################	00000040	00000044	4, 300 ns 00000034	00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 1000 1034	4, 600 ns	4,700 ns,	# 60 e0	01 00000001 4,999 060 ns
### sp[7:0] ### sp[7:0] ### ### ### #### ####################	00 e0 34 00000001 00000000 ffffffe0 00000034 Value 0 0 00000038 03 03 60 34 00000003	######################################	00000040	00000044	4,300 ns 00000034	00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	4, 600 ns.	4,700 ns,	# 00 fffffe0	4,999.000 ns 4,900 ns 00000038
### sp[7:0] ### sp[7:0] ### ### ### ### #### ################	00 e0 34 00000001 000000034 Value 0 00000038 03 03 14 00000003 15ffffff 0 0000003 15fffff 0 0000003 15ffff 0 0000003 15ffff 0 0000003 15fff 0 0000003 15ff 0 00000003 15ff 0 000000003 15ff 0 00000000000 15ff 0 00000000000 15ff 0 0000000000	######################################	00000040	00000044	4, 300 ns 00000034	00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4	4, 600 ns.	4,700 ns,	# 60 e0	4,999.000 ns 4,900 ns 00000038
## sp[7:0] ## sp[7:0] ## RF_data[4] [31:0] ## RF_data[2] [31:0] ## RF_data[29] [31:0] ## RF_data[29] [31:0] ## RF_data[31] [31:0] ## sp[7:0]	00 e0 34 00000001 000000034 Value 0 00000038 03 03 14 00000003 15ffffff 0 0000003 15fffff 0 0000003 15ffff 0 0000003 15ffff 0 0000003 15fff 0 0000003 15ff 0 00000003 15ff 0 000000003 15ff 0 00000000000 15ff 0 00000000000 15ff 0 0000000000	### 000 00 000 000 000 000 000 000 000	00000040	00000044	4,300 ns 00000034 #8	00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 4 100000 100000 100000 100000 100000 100000 100000 100000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 100000 10000 10000 10000 1000000	4, 600 ns 00000040 02 00000002	4,700 as 00000044	e0  fffffe0  4,800 ns  00000034  03  f0  00000003	4, 999 060 ns 4, 999 060 ns 00000038
### sp[7:0] ### sp[7:0] ### RF_data[4] [31:0] ### RF_data[2] [31:0] ### RF_data[2] [31:0] ### RF_data[3] [31:0] ### RF_data[3] [31:0] #### RF_data[3] [31:0] #### RF_data[3] [31:0] #### RF_data[3] [31:0] #### sp[7:0] ### sp[7:0] #### sp[7:0] #### sp[7:0] #### sp[7:0] #### sp[7:0] ### sp[7:0] #### sp[7:0] #### sp[7:0] #### sp[7:0] #### sp[7:0] ### sp[7:0] ### sp[7:0] ### sp[7:0] #### sp[7:0] #### sp[7:0] #### sp[7:0] ###	00 e0 34 00000001 Ffffffe 0 00000034 Value	######################################	00000040 01 0000000	00000044	4,300 ns 00000034 e8 ffffffe8	00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 4 1000 1000 1000 1000 1000 1000 100	4,600 ns 00000040 02 00000002	6,700 ns f	### ### ##############################	01  00000001  4,999.060 ns  4,990.050 ns  00000003
Mar sp[7:0]	00 e0 e	### ##################################	00000040 01 01 0000000	0000044	4,300 ns 00000034 #8	00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 4 100000 100000 100000 100000 100000 100000 100000 100000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 10000 100000 10000 10000 10000 1000000	4, 600 ns 00000040 02 00000002	4,700 as 00000044	#4,800 ns   00000034   03   f0   00000003   6fffff   000000003   0   0   0   0   0   0   0	4, 999 060 ns 4, 999 060 ns 00000038
### sp[7:0] ### sp[7:0] ### RF_deta[4] [31:0] ### RF_deta[2] [31:0] ### RF_deta[2] [31:0] ### RF_deta[2] [31:0] ### RF_deta[2] [31:0] ### RF_deta[3] [31:0] ### reset ### off:0] ### sp[7:0] ### RF_deta[4] [31:0] ### RF_deta[2] [31:0] ### RF_deta[3] [31:0]	00 e0 34 0000001 000000034 Value D 00000038 03 60 00000034 Value D 0 00000034 Value D 0 00000035 03 03 60 0000003 0000003 00000003 00000003 000000	### ##################################	00000040 01 000000000000000000000000000	00000044 00000044 00000044	4,300 ns 00000034 e8 ffffffe8	00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 4 1000 1000 1000 1000 1000 1000 100	4,600 ns 00000040 02 00000002	4,700 ms	### ### ##############################	01  00000001  4,999.060 ns  4,990.050 ns  00000003
## sp[7:0] ## sp[7:0] ## rep[7:0] ## RP_data[4] [31:0] ## RP_data[2] [31:0] ## RP_data[2] [31:0] ## RP_data[3] [31:0] ## RP_data[3] [31:0] ## rep[31:0]	Value    Value   D	######################################	00000040 01 00000000 000000000000000000	00000044 101 15, 200 ns, 00000044	4,300 ns 00000034 e8 ffffffe8	00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 4 1000 1000 1000 1000 1000 1000 100	4,600 ns 00000040 02 00000002	6,700 ns f	#4,800 ns	01  00000001  4,999.060 ns  4,990.050 ns  00000003
## sp[7:0] ## sp[7:0] ## sp[7:0] ## sp[7:0] ## sp[2] data[2] [31:0] ## sp_data[2] [31:0] ## sp_data[3] [31:0] ## sp_data[3] [31:0] ## sp[7:0]	00 e0 e	######################################	00000040 01 00000000 000000000000000000	00000044 101 15, 200 ns, 00000044	4,300 ns 00000034 e8 ffffffe8	00 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 4 4 1000 1000 1000 1000 1000 1000 100	4,600 ns 00000040 02 00000002	4,700 ms,	#4,800 ns   00000034   03   f0   00000003   6fffff   000000003   0   0   0   0   0   0   0	01  00000001  4,999.060 ns  4,990.050 ns  00000003

从上面的仿真结果中,可以观察到 PC 和\$a0,\$v0,\$sp,\$ra 的变化(变化与汇编程序的对应关系的解释见后仿时序仿真部分):

PC:

此后一直保持 0x00000008

\$a0:

0x00000000->0x00000004->0x00000003->0x000000002->0x00000001->0x000000000->0x00000000 1->0x00000002->0x00000003->0x000000004

此后一直保持 0x00000004

\$v0:

0x00000000->0x00000001->0x000000003->0x000000006->0x00000000a

此后一直保持 0x0000000a

\$sp:

\$ra:

0x00000000->0x00000008->0x00000034->0x00000008

足够长时间后返回值\$v0=0x0000000a,即返回 10,是正确的程序执行结果,所有寄存器的变化与理论情况一致,与预期程序功能一致,可以验证处理器正确性。

### 后仿-时序仿真:

整体 (从 Ons 运行到 6000ns):

																6.	000000 us
Name	Value	0 us			1 us			2 us		3 us		4 us			5 us		
₩ reset	0											+					
№ clk	1	hhnn	ותתתת	П	וחחו		nnn				mm	חחו	ппп	пппп	hnn	ппп	nnn.
■ N PC[31:0]	00000008		XXXX	$\infty$	$\infty$	0000	$\infty$	0000	000000	$\infty$	XXXXXX	$\infty$	$\infty$	XXXXX	$\infty$	$\infty$	$\infty$
■ ¶ \RF_data_reg[4]_0 [31:8]	000000								000	000							
<b>□</b> -♥ a0[7:0]	04	00	04	$\supset$	03	X	02	$\longrightarrow$	01		00	01	$\longrightarrow$	02	03	$=$ $\times$	04
□ ■ \RF_data_reg[2]_0 [31:8]	000000								000	000							
□-¶ v0[7:0]	0a							00					01	X	03	06	( Oa
□ ■ \RF_data_reg[29]_0 [31:8]	000000	000000	X						ff	ffff							(000000
<b>□ - 1</b> sp[7:0]	00	00	f8		$\supset$	f0	X	e8	X e0	$\square$	d8 X	e) X	e8	/ f0	$\longrightarrow$	f8	X 00
RF_data_reg[31]0 [31:8]	000000								000	000							
□ • ra[7:0]	08	00	08		<b>K</b>					34							08
D	000000	00	,		X	f0		e8	X e0	X 000	d8 X	e) X	e8	\f0	X	f8	

上图中,复位信号 reset 和时钟信号 clk 往下,分别是 PC,\$a0 高 24 位,\$a0 低八位,\$v0 高 24 位, \$v0 低八位, \$sp 高 24 位, \$sp 低八位, \$ra 高 24 位, \$ra 低八位

局部放大 (每张图分别为 1000ns 的运行时间):



											4,999.380 ns
Name	Value	4,000 ns	4, 100 ns	4, 200 ns,	4,300 ns	4,400 ns	4,500 ns	4,600 ns	4,700 ns	4,800 ns,	4,900 ns
W reset	0										
₩ clk	0										
■ - ₩ PC [31:0]	00000034	00000038	0000003e	00000040	00000044	00000034	00000038	0000003e	00000040	00000044	00000034
I	000000					0000	00				
I-₩ a0[7:0]	02	X		01					02		
I 🤻 \RF_data_reg[2]0 [31:	000000					0000	00				
I-₩ v0[7:0]	03		00		K		01				3
I ■ RF_data_reg[29]_0 [31	ffffff					ffff	ff				
	£0	e0		K		e8			K	f0	
I 🤻 \RF_data_reg[31]_0 [31	000000					0000	00				
I-Mra[7:0]	34					34					

											6,000.000 ns
Name	Value	5,000 ns	5, 100 ns	5, 200 ns	5, 300 ns	5,400 ns	5,500 ns	5,600 ns	5,700 ns	5, 800 ns	5,900 ns
1 reset	0										
la clk	1										
■ M PC[31:0]	00000008	00000038	0000003e	00000040	00000044	00000034	00000038	0000003c	00000040	00000044	00000008
■ ¶ \RF_data_reg[4]_0 [31:	000000					00000	0				
■ • a0[7:0]	04	X		03			<b></b>		04		
RF_data_reg[2]_0 [31:	000000					00000	0				
■ • v0 [7:0]	0a		03				06			X	a .
🖬 📲 \RF_data_reg[29]0 [31	000000				ffffff				K	000000	
■ 🦏 sp[7:0]	00	f0				f8			K	00	
■ 🤻 \RF_data_reg[31]0 [31	000000					00000	0				
■ ¶ ra[7:0]	08			34				<b>(</b>	0	8	

从上面的仿真结果中,可以观察到 PC 和\$a0,\$v0,\$sp,\$ra 的变化,同时对于其变化对应 汇编程序对应指令进行解释,从而说明 CPU 功能正确性:

#### PC:

此后一直保持 0x00000008

### 解释:

Ox00000000->0x00000004 这是执行前两条指令,而第二条指令是 jal sum,所以接下来跳转到 sum 模块第一条指令的位置,也就是 Ox0000000c,然后从这里开始,循环四次:

0x0000000c->0x00000010->0x000000014->0x000000018->0x00000001c->0x00000002c->0x
00000030 (->0x0000000c)

对应的是执行指令: addi \$sp, \$sp, -8 -> sw \$ra, 4(\$sp) -> sw \$a0, 0(\$sp) ->slti \$t0, \$a0, 1-> beg \$t0, \$zero, L1 -> addi \$a0, \$a0, -1 -> jal sum

就是分别对应执行 sum(4), sum(3), sum(2), sum(1)

然后, 完整执行 sum(0), 因\$a0=0 而不再跳转到 L1:

0x0000000c->0x00000010->0x00000014->0x00000018->->0x0000001c->0x00000020-> 0x00000024->0x00000028

对应指令为:

addi \$sp, \$sp, -8 -> sw \$ra, 4(\$sp) -> sw \$a0, 0(\$sp) -> slti \$t0, \$a0, 1-> beq \$t0, \$zero, L1 ->xor \$v0, \$zero, \$zero, addi \$sp, \$sp, 8, jr \$ra

然后循环四次出栈和累加返回值的过程:

0x00000034->0x00000038->0x0000003c->0x00000040->0x00000044

对应指令为:

lw \$a0, 0(\$sp) -> lw \$ra, 4(\$sp) -> addi \$sp, \$sp, 8 -> add \$v0, \$a0, \$v0 -> jr \$ra 最后跳转回 Loop 后,保持 0x00000008 不变,对应着指令: beg \$zero, \$zero, Loop

\$a0:

0x00000000->0x00000004->0x000000003->0x000000002->0x00000001->0x00000000->0x00000000 1->0x00000002->0x00000003->0x000000004

此后一直保持 0x00000004

解释:

一开始 reset 成 0x00000000, 然后第一条指令为\$a0 赋值为 0x00000004, 然后每次 sum 递归调用的时候,都会因指令 addi \$a0,\$a0,-1 而减 1,由此从 0x00000004->0x00000003->0x00000002->0x00000001->0x00000000, 到 0 之后就开 始将之前保存的\$a0 出栈,于是逐次恢复也就是逐次加1, (0x00000000->)0x00000001->0x00000002->0x00000003->0x00000004,此后不再改变, 一直保持 0x00000004

\$v0:

0x00000000->0x00000001->0x00000003->0x00000006->0x00000000a

此后一直保持 0x0000000a

解释:

在一开始的过程中一直保持 0x00000000, 直到开始出栈后开始累加保存在栈中的值,该值就对应着 sum(n)递归调用的返回值,因为 n=4,所以一开始初始化成 0,然后开始累加存在栈中的值,第一次加 1,变成 0x00000001,第二次加 2,变成 0x00000003,第三次加 3,变成 0x00000006,第四次加 4,变成 0x0000000a,此后该值不再改变,一直保持。

\$sp:

解释:

一开始 reset 成 0x00000000,然后一共压栈 5 次,对应 sum(4),sum(3) ,sum(2) ,sum(1),sum(0),每次减 8,就有了:

0x0000000->0xfffffff8->0xfffffff0->0xffffffe8->0xffffffd8

然后对应 5 次出栈,每次加 8,就有了:

(0xffffffd8->)0xffffffe0->0xffffffe8->0xfffffff0->0xfffffff8->0x00000000

\$ra:

0x00000000->0x00000008->0x00000034->0x00000008

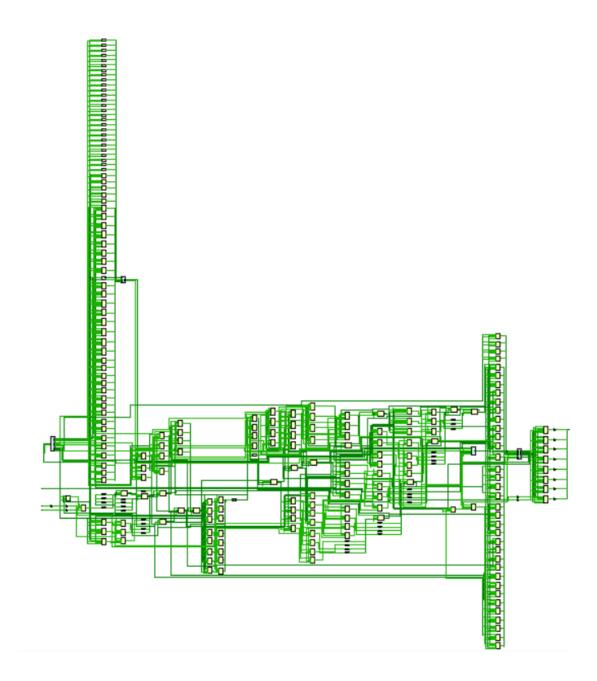
解释:

一开始 reset 成 0x00000000,然后第一次执行第 1 条指令 jal sum 时,保存 Loop 的位置,将\$ra 变为 0x00000008,之后递归调用中反复执行第 12 条指令 jal sum 时,保存的\$ra 的值均为 0x00000034,所以该值保持了很长时间,最后一次出栈时,将\$ra 恢复到 0x00000008,然后一直保持,不再改变。

足够长时间后返回值\$v0=0x0000000a,即返回 10,是正确的程序执行结果,从上述的分析可以看到,所有寄存器的变化和 PC 的变化与理论情况一致,与预期程序功能一致,可以验证处理器正确性。

## 6、综合情况:

在进行综合时选择将 flatten hierarchy 属性置成了 none,综合后对应的电路原理图为:



综合后 FPGA 逻辑资源情况:

↑ 1 Name	Slice LUTs* (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Bonded IOB (210)	BUFGCTRL (32)
⊟-∭ CPU	3750	8704	1322	512	12	1
alu1 (ALU)	423	0	10	0	0	0
alu_control1 (ALUCont	12	0	0	0	0	0
control1 (Control)	12	0	0	0	0	0
··· data_memory1 (DataMem	2510	8192	1056	512	0	0
instruction_memory1 (	21	0	0	0	0	0
register_file1 (Regis	591	480	256	0	0	0

可以看到,整个 CPU 使用了 3750 个 LUT 和 8704 个 Register,1322 个 F7-MUX,512 个 F8-MUX,12 个 BondedOIOB 和 1 个 BUFGCTRL。

可以从具体各个模块的占比中看到, LUT 主要是被数据存储器、寄存器堆和 ALU 这三个模块所使用,这三个模块都有较多的逻辑运算部分。寄存器主要是被数据存储器和寄存器堆这两个模块所使用,这是合理的,不过指令指令存储器并没有使用到寄存器,原因应该是本实验中的指令存储器是通过条件判断以组合逻辑来实现的,没有实现真正的存取指令,所以

这个结果也是合理的。MUX 主要被数据存储器和寄存器堆这两个模块所使用,推测应该是这两个模块的寻址功能需用使用 MUX 来实现。CPU 模块独有 12 个 Bonded-IOB 是对应这个其 4 位输入和 8 位输出共计 12 位,其还独有 1 个 BUFGCTRL,对应的是其使用到的时钟。

# 7、 时序性能

经过一定的尝试,在综合的时候,使用的时钟频率的约束为 10MHz。

### 整体的时序性能:

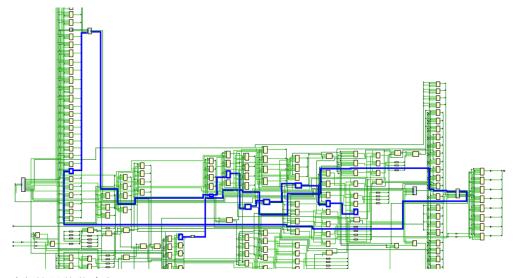
Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	85.850 ns	Worst Hold Slack (WHS):	0.262 ns	Worst Pulse Width Slack (WPWS):	49.460 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000  ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	17376	Total Number of Endpoints:	17376	Total Number of Endpoints:	8705

All user specified timing constraints are met.

可以看到,建立时间约束最坏的 slack 为 85.850ns,保持时间约束最坏的 slack 为 0.262ns 因为我采用的是 10MHz 的时钟频率,那么有建立时间约束最坏的 slack,也就是关键路径建立时间的 slack 为 85.850ns,那么可以知道,理论上最短的时钟周期为 100-85.850=14.15ns,由此可以求得最高工作频率约为 70.67MHz

从 10 条 setup 的 slack 最短的路径中,在原理图上找出其中时间最短的一条路径(关键路径):

Name	Slack	Levels	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Source Clock	Destination Clock	Exception
Path 1	85.850	22	291	PC_reg[2]/C	PC_reg[28]/D	14.137	5.137	9.000	CLK	CLK	
Path 2	85.964	21	291	PC_reg[2]/C	PC_reg[24]/D	14.023	5.023	9.000	CLK	CLK	
Path 3	85.981	23	291	PC_reg[2]/C	PC_reg[31]/D	14.006	5.172	8.834	CLK	CLK	
Path 4	86.024	23	291	PC_reg[2]/C	PC_reg[30]/D	13.963	5. 269	8.694	CLK	CLK	
Path 5	86.095	22	291	PC_reg[2]/C	PC_reg[27]/D	13.892	5.058	8.834	CLK	CLK	
Path 6	86.138	22	291	PC_reg[2]/C	PC_reg[26]/D	13.849	5. 155	8.694	CLK	CLK	
Path 7	86.149	23	291	PC_reg[2]/C	PC_reg[29]/D	13.838	5.145	8.693	CLK	CLK	
Path 8	86.188	23	291	PC_reg[2]/C	PC_reg[20]/D	13. 799	5. 241	8. 558	CLK	CLK	
Path 9	86. 209	21	291	PC_reg[2]/C	PC_reg[23]/D	13.778	4.944	8.834	CLK	CLK	
Path 10	86. 252	21	291	PC_reg[2]/C	PC_reg[22]/D	13. 735	5.041	8.694	CLK	CLK	



查看它的具体信息如下:

Summary	
Name	₹ Path 1
Slack	85. 850ns
Source	▶ PC_reg[2]/C (rising edge-triggered cell FDCE clocked by CLK {rise@0.000ns fall@50.000ns period=100.000ns})
Destination	PC_reg[28]/D (rising edge-triggered cell FDCE clocked by CLK {rise@0.000ns fall@50.000ns period=100.000ns})
Path Group	CLK
Path Type	Setup (Max at Slow Process Corner)
Requirement	100.000ns (CLK rise@100.000ns - CLK rise@0.000ns)
Data Path Delay	14.137ns (logic 5.137ns (36.337%) route 9.000ns (63.663%))
Logic Levels	22 (CARRY4=7 LUT2=1 LUT4=4 LUT5=6 LUT6=3 MUXF7=1)
Clock Path Skew	<u>-0.040ns</u>
Clock Uncertainty	0.035ns

### Source Clock Path

Source Clock Lath				
Delay Type	Incr (ns)	Path (ns) Location	on Netlist Resource(s)	
(clock CLK rise edge)	(r) 0.000	0.000		
	(r) 0.000	0.000 Site: P	17 D clk	
net (fo=0)	0.000	0.000	∠ clk	
		Site: P	17 D clk_IBUF_inst/I	
IBUF (Prop ibuf I 0)	(r) 1.478	1.478 Site: P	17	
net (fo=1, unplaced)	0.800	2.278	∠ clk_IBUF	
			clk_IBUF_BUFG_inst/I	
BUFG (Prop bufg I 0)	(r) 0.096	2.374	<pre>clk_IBUF_BUFG_inst/0</pre>	
net (fo=8704, unplaced)	0.800	3.174	∠ clk_IBUF_BUFG	
			▶ PC reg[2]/C	

### Data Path

Data Path				
Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
FDCE (Prop fdce C Q)	(r) 0.496	3.670		<pre>PC_reg[2]/Q</pre>
net (fo=21, unplaced)	0.372	4.042		/ instruction_memory1/Address[2]
				instruction_memory1/g0_b16/I0
LUT5 (Prop lut5 IO 0)	(f) 0.295	4. 337		<pre>instruction_memory1/g0_b16/0</pre>
net (fo=1, unplaced)	1.111	5.448		/ instruction_memory1/n_0_g0_b16
				instruction_memory1/Instruction[16]_INST_0/I0
LUT4 (Prop lut4 IO 0)	(f) 0.118	5.566		<pre>instruction_memory1/Instruction[16]_INST_0/0</pre>
net (fo=291, unplaced)	0.693	6. 259		<pre>/ register_file1/Read_register2[4]</pre>
				register_file1/Read_data2[6]_INST_0_i_12/I2
LUT4 (Prop lut4 I2 0)	(r) 0.328	6.587		<pre>gister_file1/Read_data2[6]_INST_0_i_12/0</pre>
net (fo=1, unplaced)	0.000	6.587		<pre>/ register_file1/n_0_Read_data2[6]_INST_0_i_12</pre>
				register_file1/Read_data2[6]_INST_0_i_4/I1
MUXF7 (Prop muxf7 I1 0)	(r) 0.245	6.832		<pre>gister_file1/Read_data2[6]_INST_0_i_4/0</pre>
net (fo=1, unplaced)	0.452	7.284		<pre>/ register_file1/n_0_Read_data2[6]_INST_0_i_4</pre>
				register_file1/Read_data2[6]_INST_0/I5
LUT6 (Prop lut6 I5 0)	(r) 0.298	7.582		<pre>gister_file1/Read_data2[6]_INST_0/0</pre>
net (fo=257, unplaced)	1.025	8.607		→ Databus2[6]
				■ alu1_i_58/I2
LUT4 (Prop lut4 I2 0)	(r) 0.118	8. 725		<pre>alu1_i_58/0</pre>
net (fo=13, unplaced)	1.033	9. 758		/ alu1/in2[6]
				alu1/out[0]_INST_0_i_76/I0
LUT4 (Prop lut4 IO 0)	(r) 0.328	10.086		<pre>alu1/out[0]_INST_0_i_76/0</pre>
net (fo=1, unplaced)	0.000	10.086		/ alu1/n_0_out[0]_INST_0_i_76
				<pre>alu1/out[0]_INST_0_i_55/S[3]</pre>
CARRY4 (Prop carry4 S[3] CO[3])	(r) 0.401	10.487		<pre>alu1/out[0]_INST_0_i_55/C0[3]</pre>
net (fo=1, unplaced)	0.009	10.496		/ alu1/n_0_out[0]_INST_0_i_55
				alu1/out[0]_INST_0_i_37/CI
CARRY4 (Prop carry4 CI CO[3])	(r) 0.114	10.610		<pre>alu1/out[0]_INST_0_i_37/C0[3]</pre>
net (fo=1, unplaced)	0.000	10.610		/ alu1/n_0_out[0]_INST_0_i_37
				alu1/out[0]_INST_0_i_19/CI
CARRY4 (Prop carry4 CI CO[3])	(r) 0.114	10.724		<pre>alu1/out[0]_INST_0_i_19/C0[3]</pre>
net (fo=1, unplaced)	0.000	10.724		/ alu1/n_0_out[0]_INST_0_i_19
				alu1/out[0]_INST_0_i_9/CI

CARRY4 (Prop carry4 CI CO[3])	(f) 0.114	10.838	<pre>alu1/out[0] INST_0 i_9/C0[3]</pre>
net (fo=1, unplaced)	0.918	11. 756	alul/n 0 out[0] INST 0 i 9
			D alu1/out[0] INST 0 i 6/I4
LUT5 (Prop lut5 I4 0)	(f) 0.124	11.880	<pre>alu1/out[0] INST 0 i 6/0</pre>
net (fo=1, unplaced)	0.449	12.329	/ alu1/n 0 out[0] INST 0 i 6
			■ alu1/out[0] INST 0 i 2/I0
LUT5 (Prop lut5 IO 0)	(f) 0.124	12.453	alu1/out[0] INST 0 i 2/0
net (fo=1, unplaced)	0.449	12.902	/ alu1/n 0 out[0] INST 0 i 2
			alu1/out[0] INST 0/I2
LUT5 (Prop lut5 I2 0)	(f) 0.124	13.026	alu1/out[0] INST 0/0
net (fo=2, unplaced)	0.460	13.486	/ alu1/out[0]
			alu1/zero INST 0 i 3/I2
LUT5 (Prop lut5 I2 0)	(r) 0.124	13.610	alu1/zero INST_0 i 3/0
net (fo=1, unplaced)	0.449	14.059	/ alu1/n_0_zero_INST_0 i_3
			alu1/zero_INST_0/I2
LUT6 (Prop lut6 I2 0)	(r) 0.124	14. 183	alu1/zero_INST_0/0
net (fo=25, unplaced)	0.484	14.667	∕ Zero
			PC[31]_i_6/I0
LUT2 (Prop lut2 IO 0)	(r) 0.124	14. 791	PC[31]_i_6/0
net (fo=6, unplaced)	0.478	15. 269	/n_0_PC[31]_i_6
			▶ PC[20]_i_5/I5
LUT6 (Prop lut6 I5 0)	(r) 0.124	15.393	<pre>PC[20]_i_5/0</pre>
net (fo=1, unplaced)	0.000	15.393	/n_0_PC[20]_i_5
			▶ PC_reg[20]_i_2/S[1]
CARRY4 (Prop carry4 S[1] CO[3])	(r) 0.550	15.943	<pre>PC_reg[20]_i_2/C0[3]</pre>
net (fo=1, unplaced)	0.000	15.943	/n_0_PC_reg[20]_i_2
			▶ PC_reg[24]_i_2/CI
CARRY4 (Prop carry4 CI CO[3])	(r) 0.114	16.057	<pre>PC_reg[24]_i_2/C0[3]</pre>
net (fo=1, unplaced)	0.000	16.057	/n_0_PC_reg[24]_i_2
Tarried Control of the Control of th			▶ PC_reg[28]_i_2/CI
CARRY4 (Prop carry4 CI 0[3])	(r) 0.329	16.386	<pre>PC_reg[28]_i_2/0[3]</pre>
net (fo=1, unplaced)	0.618	17.004	/ Branch_target[28]
71111177111771177			▶ PC[28]_i_1/I3
LUT5 (Prop lut5 I3 0)	(r) 0.307	17.311	<pre>PC[28]_i_1/0</pre>
net (fo=1, unplaced)	0.000	17.311	/n_0_PC[28]_i_1
FDCE			▶ PC_reg[28]/D
Arrival Time		17.311	

#### Destination Clock Path

ZUUULUUUU UZUUL TUUL				
Delay Type	Incr (ns)	Path (ns)	Location	Netlist Resource(s)
(clock CLK rise edge)	(r) 100.000	100.000		
	(r) 0.000	100.000	Site: P17	D clk
net (fo=0)	0.000	100.000		∕ clk
			Site: P17	clk_IBUF_inst/I
IBUF (Prop ibuf I 0)	(r) 1.408	101.408	Site: P17	<pre>clk_IBUF_inst/0</pre>
net (fo=1, unplaced)	0.760	102.168		/ clk_IBUF
				clk_IBUF_BUFG_inst/I
BUFG (Prop bufg I 0)	(r) 0.091	102.259		<pre>clk_IBUF_BUFG_inst/0</pre>
net (fo=8704, unplaced)	0.760	103.019		/ clk_IBUF_BUFG
				▶ PC_reg[28]/C
clock pessimism	0.116	103.134		
clock uncertainty	<b>−</b> 0.035	103.099		
FDCE (Setup fdce C D)	0.062	103.161		<pre>PC_reg[28]</pre>
Required Time		103.161		

分析上述信息可以知道,这条关键路径经过了: PC->指令存储器(instruction memory)->寄存器堆(register file)->ALU->PC 更新电路。同时由路径中出现的 zero 信号和 Branch\_target 信号推断这条关键路径对应的是 beq 指令,这出乎我的预料,关键路径对应的指令不是 lw,但是结合之前的综合资源使用情况,我猜测可能是由于 vivado 综合采用的资源不同,使得读取存储时间不再那么久。

根据上述信息,结合这一条关键路径的电路图,把各单元之间的连线的延时去除,各单元的耗时如下:

单元	耗时/ns	
指令存储器	1.524	
寄存器堆	1.323	

ALU	5.576
PC 更新电路	2.644

通过上表可以看到,ALU 单元是耗时最长的单元。