数字逻辑与处理器基础

第二次处理器大作业

班级：无 01 班

学号：2020010641

姓名：程书鹏

日期：2022.6.19

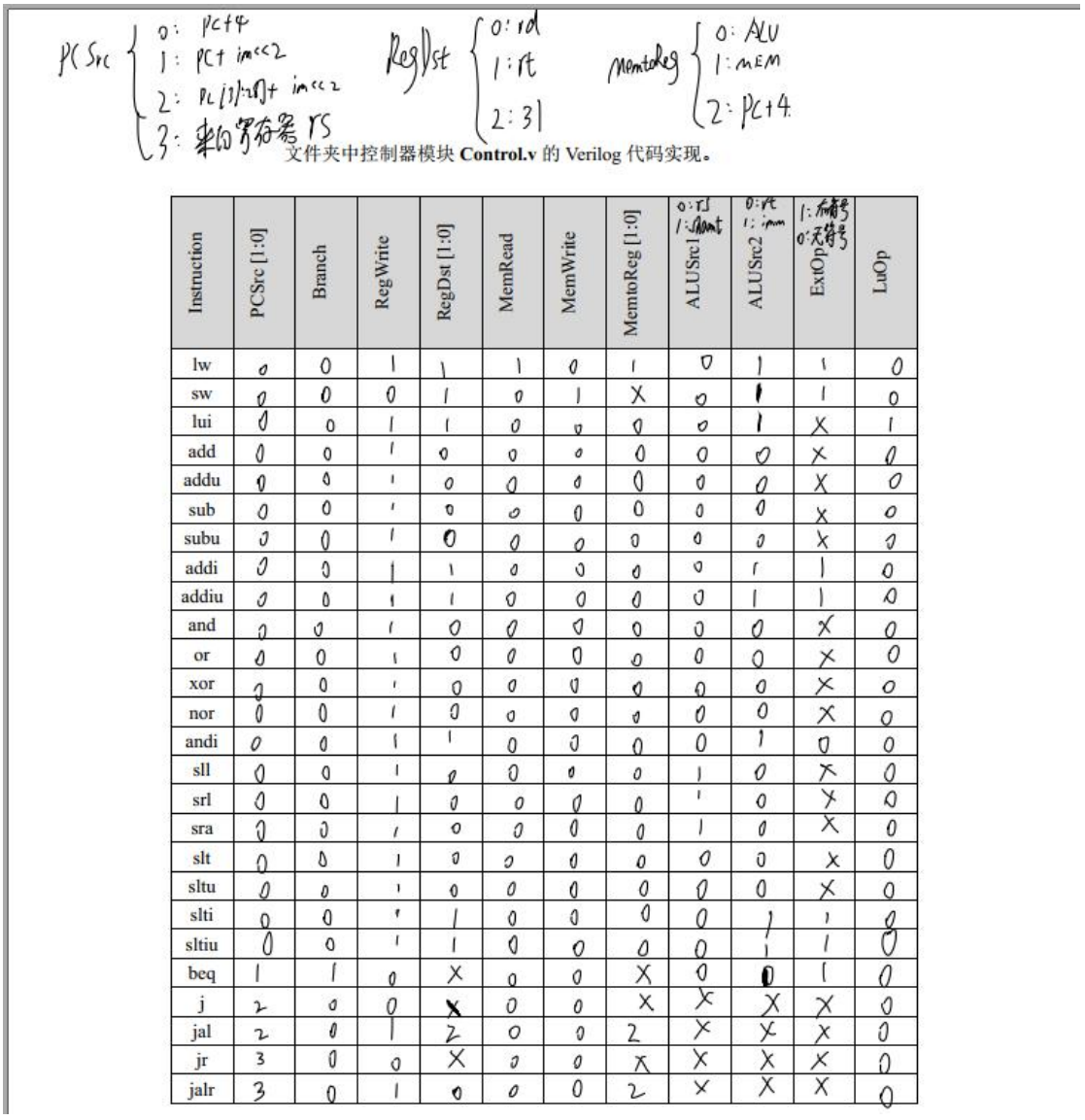
一． 实验目的

1. 掌握 MIPS 单周期处理器的控制通路和数据通路的设计原理和 RTL 实现方法。
2. 掌握 MIPS 多周期处理器的控制通路和数据通路的设计原理和 RTL 实现方法。
3. 深入理解 MIPS 单周期和多周期处理器在资源和性能上的设计折衷。

二． 实验内容一

(1) 控制器模块设计

MIPS 指令集子集与控制信号的真值表如下：



Instruction	PCSrc [1:0]	Branch	RegWrite	RegDst [1:0]	MemRead	MemWrite	MemtoReg [1:0]	ALUSrc1	ALUSrc2	ExtOp	LuOp
lw	0	0	1	1	1	0	1	0	1	1	0
sw	0	0	0	1	0	1	X	0	1	1	0
lui	0	0	1	1	0	0	0	0	1	X	1
add	0	0	1	0	0	0	0	0	0	X	0
addu	0	0	1	0	0	0	0	0	0	X	0
sub	0	0	1	0	0	0	0	0	0	X	0
subu	0	0	1	0	0	0	0	0	0	X	0
addi	0	0	1	1	0	0	0	0	1	1	0
addiu	0	0	1	1	0	0	0	0	1	1	0
and	0	0	1	0	0	0	0	0	0	X	0
or	0	0	1	0	0	0	0	0	0	X	0
xor	0	0	1	0	0	0	0	0	0	X	0
nor	0	0	1	0	0	0	0	0	0	X	0
andi	0	0	1	1	0	0	0	0	1	0	0
sll	0	0	1	0	0	0	0	1	0	X	0
srl	0	0	1	0	0	0	0	1	0	X	0
sra	0	0	1	0	0	0	0	1	0	X	0
slt	0	0	1	0	0	0	0	0	0	X	0
sltu	0	0	1	0	0	0	0	0	0	X	0
slti	0	0	1	1	0	0	0	0	1	1	0
sltiu	0	0	1	1	0	0	0	0	1	1	0
beq	1	1	0	X	0	0	X	0	0	1	0
j	2	0	0	X	0	0	X	X	X	X	0
jal	2	0	1	2	0	0	2	X	X	X	0
jr	3	0	0	X	0	0	X	X	X	X	0
jalr	3	0	1	0	0	0	2	X	X	X	0

(2) 数据通路设计

数据通路中包含的多路选择器如下：

1. 立即数扩展方式多路选择器
   功能是选择立即数扩展方式为无符号扩展还是有符号扩展
   ```
   assign
   Ext_out={ExtOp?{16{instruction[15]}}:16'h0000,instruction[15:0]};
   ```

2. lui 指令输出多路选择器
   功能是选择输出立即数扩展结果还是 lui 指令专门的结果
   ```
   assign Lu_out=LuOp?{instruction[15:0],16'h0000}:Ext_out;
   ```

3. 写入寄存器多路选择器
   功能是选择写入的寄存器是 rd，rt 还是 31 号寄存器
   ```
   assign
   Write_register=(RegDst==2'b00)?instruction[15:11]:(RegDst==2'b01)?
   instruction[20:16]:5'b11111;
   ```

4. ALU 输入 1 多路选择器
   功能是选择 ALU 的第一路输入为偏移量还是寄存器堆的输出 busA
   ```
   assign in1=(ALUSrc1)?instruction[10:6]:databusA;
   ```

5. ALU 输入 2 多路选择器
   功能是选择 ALU 的第二路输入为 Lu_out 还是寄存器堆的输出 busB
   ```
   assign in2=(ALUSrc2)?Lu_out:databusB;
   ```

6. 寄存器堆写入多路选择器
   功能是选择寄存器堆写入的内容是 ALU 的输出还是从内存读取的内容还是 PC+4
   ```
   assign
   databusW=(MemtoReg==2'b00)?ALUout:(MemtoReg==2'b01)?Read_data:PCj
   ia4;
   ```

7. 分支指令多路选择器
   功能是选择 branch 的目标地址是分支地址还是 PC+4
   ```
   assign
   branch_address=(Branch&&zero)?(PCjia4+{Lu_out[29:0],2'b00}):PCjia
   4;
   ```

8. PC 多路选择器
   功能是选择下一条指令是 PC+4 还是分支地址还是跳转地址还是寄存器中的地址
   ```
   assign
   PC_next=(PCSrc==2'b00)?PCjia4:(PCSrc==2'b01)?branch_address:(PCS
   rc==2'b10)?jump_address:databusA;
   ```

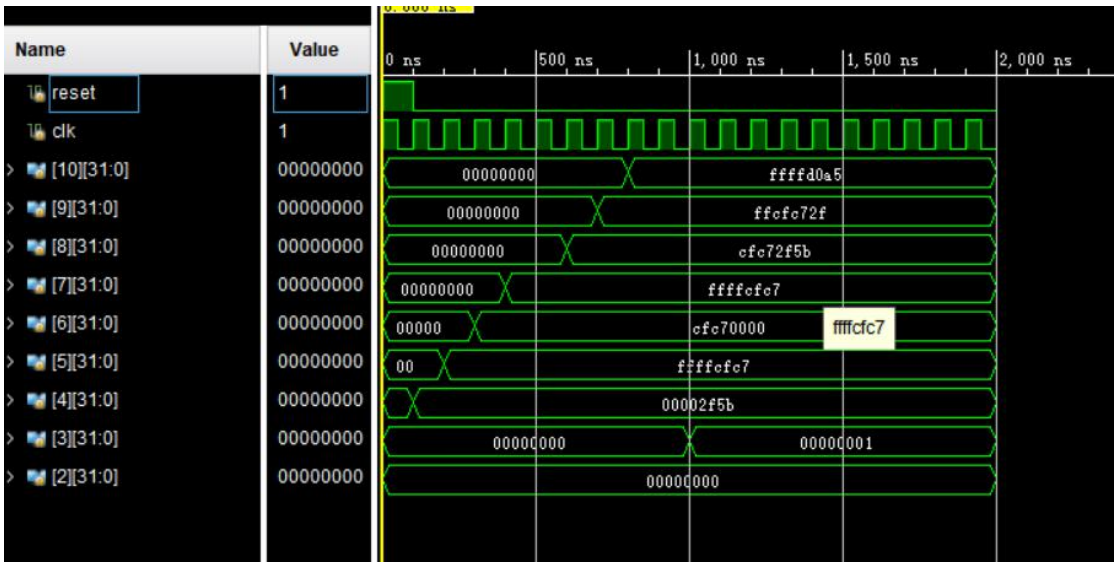
(3) 汇编程序分析 1

1. 分析计算

按指令顺序执行过程如下：
a0=0x00002f5b
a1=0xffffcfc7
a2=0xcfc70000
a3=0xffffcfc7
beq 指令发现 a3=a1，于是跳转到 L1，跳过 lui 指令，不改变 a0 的值
to=a2+a0=0xcfc72f5b
t1=0xffcfc72f
t2=0xffffd0a5
slt 指令发现按照有符号数比较 a0>t2，于是将 v0 置为 0
sltu 指令发现按照无符号数比较 a0<t2,于是将 v1 置为 1

2. 仿真验证计算及单周期处理器的功能正确性

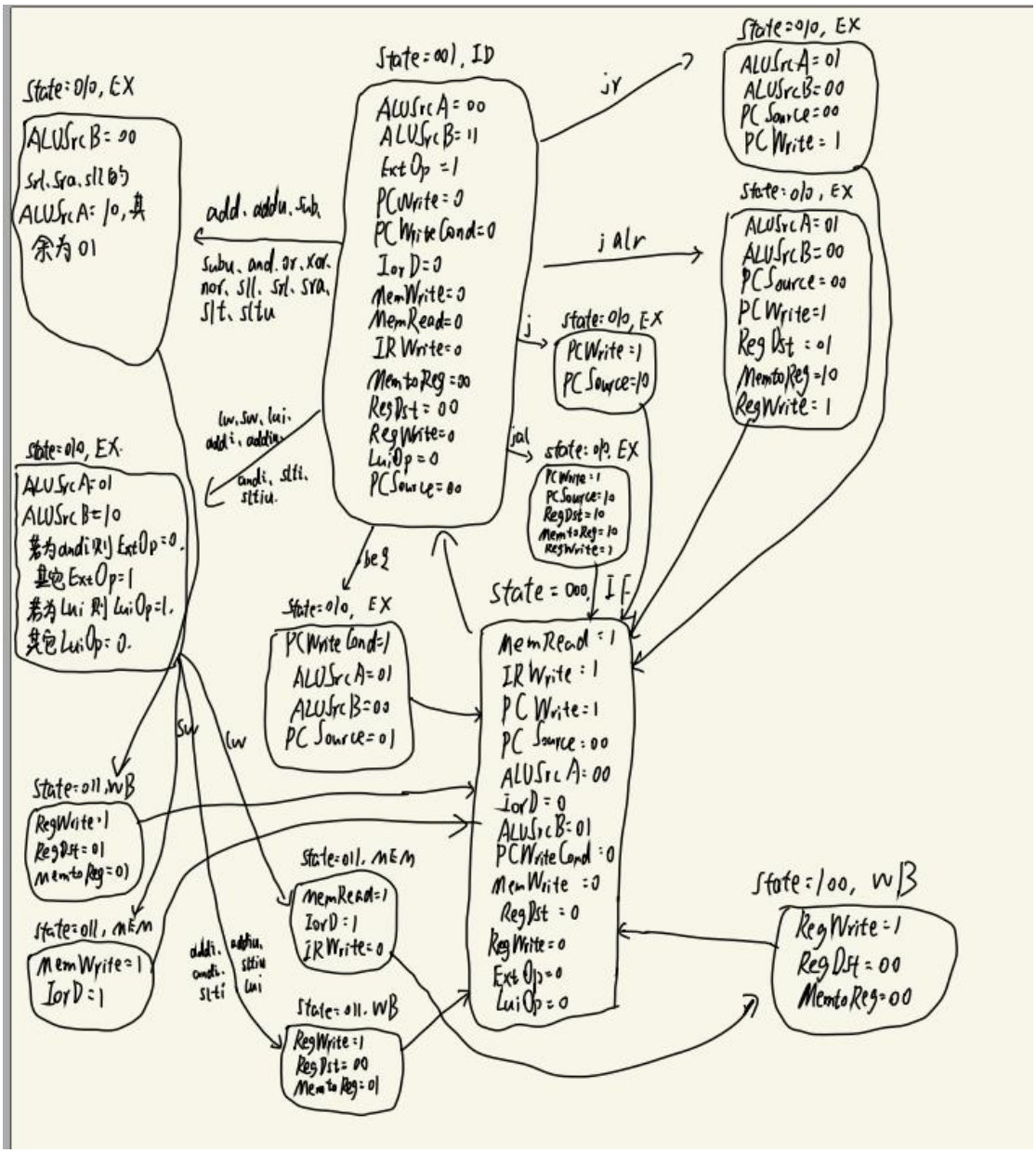
仿真结果如下图所示：



根据上图，仿真结果与分析计算结果相同，验证了计算结果和设计的单周期处理器的功能正确性。

三． 实验内容二

(1) 状态转移图



(2) 设计思路与代码逻辑

根据有限状态机思想，在助教原有的代码基础上在每个状态的控制信号中直接加入 ALUOp，不用专门写一个 ALUControl 模块生成 ALU 的控制信号。ALU 在每个控制信号的控制下执行的操作与单周期没有差别，多周期与单周期的不同在于单周期 ALU 控制信号的生成全部由指令决定，而多周期 ALU 控制信号不仅与指令有关，还与每条指令所处的状态有关。具体的代码请见压缩包内的源代码。

(3) 数据通路

数据通路中包含的寄存器和多路选择器如下：

1. 存储器输出数据的寄存器 MDR，功能为暂时存储存储器输出的数据，并在下一个上升沿到来时将存储的数据输出给后续电路。代码如下：

```
module RegTemp(reset, clk, Data_i, Data_o);
```

```
        //Input Clock Signals
        input reset;
        input clk;
        //Input Data
        input [31:0] Data_i;
        //Output Data
        output reg [31:0] Data_o;

        always@(posedge reset or posedge clk) begin
            if (reset) begin
                Data_o <= 32'h00000000;
            end else begin
                Data_o <= Data_i;
            end
        end
endmodule
RegTemp MDR(reset,clk,Mem_data0,Mem_data_tmp);
```

2. 寄存器堆输出通道 A 寄存器，功能为暂时存储寄存器堆输出通道 A 的数据并在下一个上升沿到来的时候将其输出给后续电路。代码如下：

```
    module RegTemp(reset, clk, Data_i, Data_o);
        //Input Clock Signals
        input reset;
        input clk;
        //Input Data
        input [31:0] Data_i;
        //Output Data
        output reg [31:0] Data_o;

        always@(posedge reset or posedge clk) begin
            if (reset) begin
                Data_o <= 32'h00000000;
            end else begin
                Data_o <= Data_i;
            end
        end
endmodule
RegTemp A_register(reset,clk,Read_data10,Read_data10_tmp);
```

3. 寄存器堆输出通道 B 寄存器，功能为暂时存储寄存器堆输出通道 B 的数据并在下一个上升沿到来的时候将其输出给后续电路。代码如下：

```
    module RegTemp(reset, clk, Data_i, Data_o);
        //Input Clock Signals
        input reset;
```

```
        input clk;
        //Input Data
        input [31:0] Data_i;
        //Output Data
        output reg [31:0] Data_o;

        always@(posedge reset or posedge clk) begin
            if (reset) begin
                Data_o <= 32'h00000000;
            end else begin
                Data_o <= Data_i;
            end
        end
endmodule
RegTemp B_register(reset,clk,Read_data20,Read_data20_tmp);
```

4. ALU 输出寄存器，功能为暂时 ALU 输出的数据并在下一个上升沿到来的时候将其输出给后续电路。代码如下：

```
    module RegTemp(reset, clk, Data_i, Data_o);
        //Input Clock Signals
        input reset;
        input clk;
        //Input Data
        input [31:0] Data_i;
        //Output Data
        output reg [31:0] Data_o;

        always@(posedge reset or posedge clk) begin
            if (reset) begin
                Data_o <= 32'h00000000;
            end else begin
                Data_o <= Data_i;
            end
        end
endmodule
RegTemp ALUout_register(reset,clk,Result0,Result0_tmp);
```

5. 存储器输入数据多路选择器，其功能为选择存储器输入的地址来自于 PC（IorD=0）还是 ALU 结果寄存器中的数（IorD=1）代码如下：

```
assign Address0=(IorD0)?Result0_tmp:PC_o0;
```

6. 寄存器堆写入寄存器多路选择器，其功能为控制寄存器堆中即将被写入的寄存器编号为 rt（RegDst=00）还是 rd（RegDst=01）还是 31（RegDst=10）。代码如下：

```
assign
Write_register0=(RegDst0==2'b00)?rt0:(RegDst0==2'b01)?rd0:5'd
31;
```

7. 寄存器堆写入数据多路选择器，其功能为控制寄存器堆写入数据的来源是 MDR（MemtoReg=00）还是 ALU 结果寄存器中的值（MemtoReg=01）还是 PC+4（MemtoReg=10）。代码如下：

```
assign
Write_data1=(MemtoReg0==2'b00)?Mem_data_tmp:(MemtoReg0==2'b01)
?Result0_tmp:PC_o0;
```

8. ALU 输入通道 1 多路选择器，其功能为选择 ALU 第一路输入为 PC（ALUSrcA=00）还是寄存器堆输出通道 1 寄存器的值（ALUSrcA=01）还是来自指令中的 shamt 部分（ALUSrcA=10）。代码如下：

```
assign
in10=(ALUSrcA0==2'b00)?PC_o0:(ALUSrcA0==2'b01)?Read_data10_tm
p:Shamt0;
```

9. ALU 输入通道 2 多路选择器，其功能为选择 ALU 第二路输入为寄存器堆输出通道 2 寄存器的值（ALUSrcB=00）还是 4（ALUSrcB=01）还是立即数扩展（ALUSrcB=10）还是立即数左移两位（ALUSrcB=11）。代码如下：

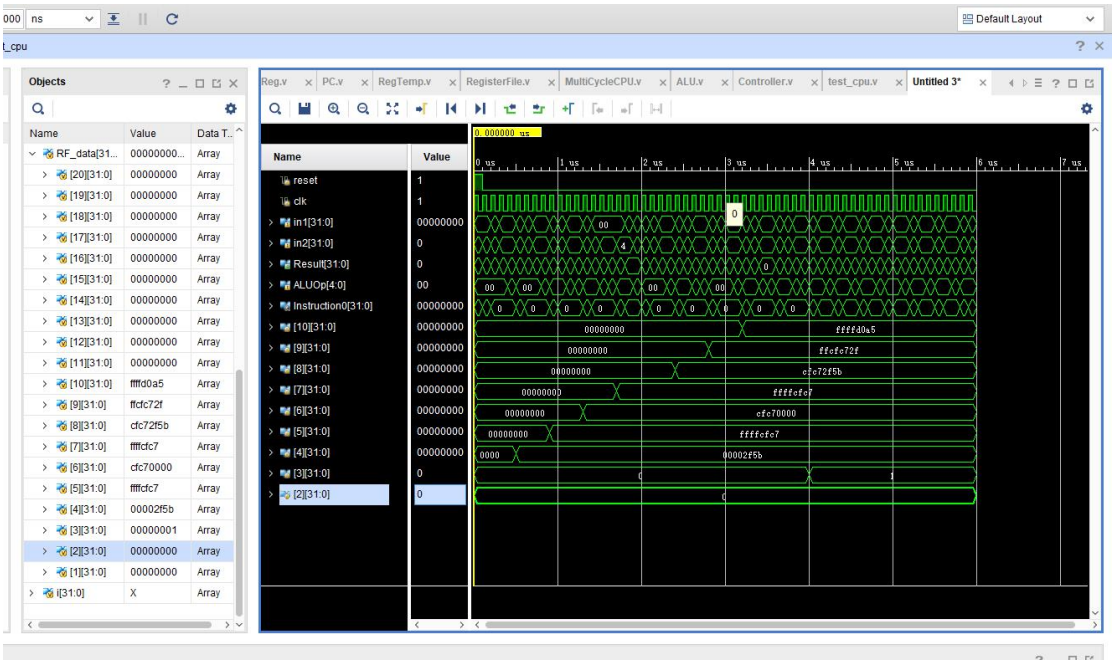
```
assign
in20=(ALUSrcB0==2'b00)?Read_data20_tmp:(ALUSrcB0==2'b01)?32'd
4:(ALUSrcB0==2'b10)?ImmExtOut0:ImmExtShift0;
```

10. PC 更新来源多路选择器，其功能为选择 PC 更新的来源为 ALU 计算结果（PCSource=00）还是 ALU 输出寄存器中的值（PCSource=01）还是立即数后 26 位左移两位并与 PC 高四位拼接（PCSource=10）。代码如下：

```
assign
PC_i0=(PCSource0==2'b10)?{PC_o0[31:28],rs0,rt0,rd0,Shamt0,Fun
ct0,2'b00}:(PCSource0==2'b01)?Result0_tmp:Result0;
```

(4) 功能验证

仿真波形如下：



根据上图，仿真结果与之前分析的结果相同，验证了多周期处理器的功能正确性。

四． 实验内容三

(a)

1.是求出 n（n-1）的值保存在$v0 中。Loop 段的作用是在程序最后执行完之后不断循环此语句。Sum 段作用是参数入栈并且判断是否继续循环。L1 段作用是将$v0 与$a0 相加并将$a0 减 1，同时参数出栈。代码添加注释如下：

```
addi $a0,$zero,5    //$a0=5,$a0 初始化
xor $v0,$zero,$zero    //$v0=0，$v0 初始化
jal sum    //跳转到 sum 段，同时将下一条指令即 Loop 语句的地址存入$ra
Loop:
beq $zero,$zero,Loop    //beq 指令分支恒成立，不断循环本指令
sum:
addi $sp,$sp,-8    //移动栈指针，准备入栈
sw $ra,4($sp)    //将当前$ra 的值入栈
sw $a0,0($sp)    //将当前$a0 的值入栈
slti $t0,$a0,1    //如果$a0<1,则将$t0 置为 1
beq $t0,$zero,L1    //如果$t0=0，则跳到 L1，即$a0<1 时才执行后面两句
addi $sp,$sp,8    //移动栈指针
jr $ra    //跳转到当前$ra 里存的地址
L1:
add $v0,$a0,$v0    //将$v0 与$a0 相加的值存入$v0
addi $a0,$a0,-1    //$a0--
jal sum      //跳到 sum 段，同时将下一条指令地址存入$ra
lw $a0,0($sp) //从栈里取值存入$a0
lw $ra,4($sp)    //从栈里取值存入$ra
```

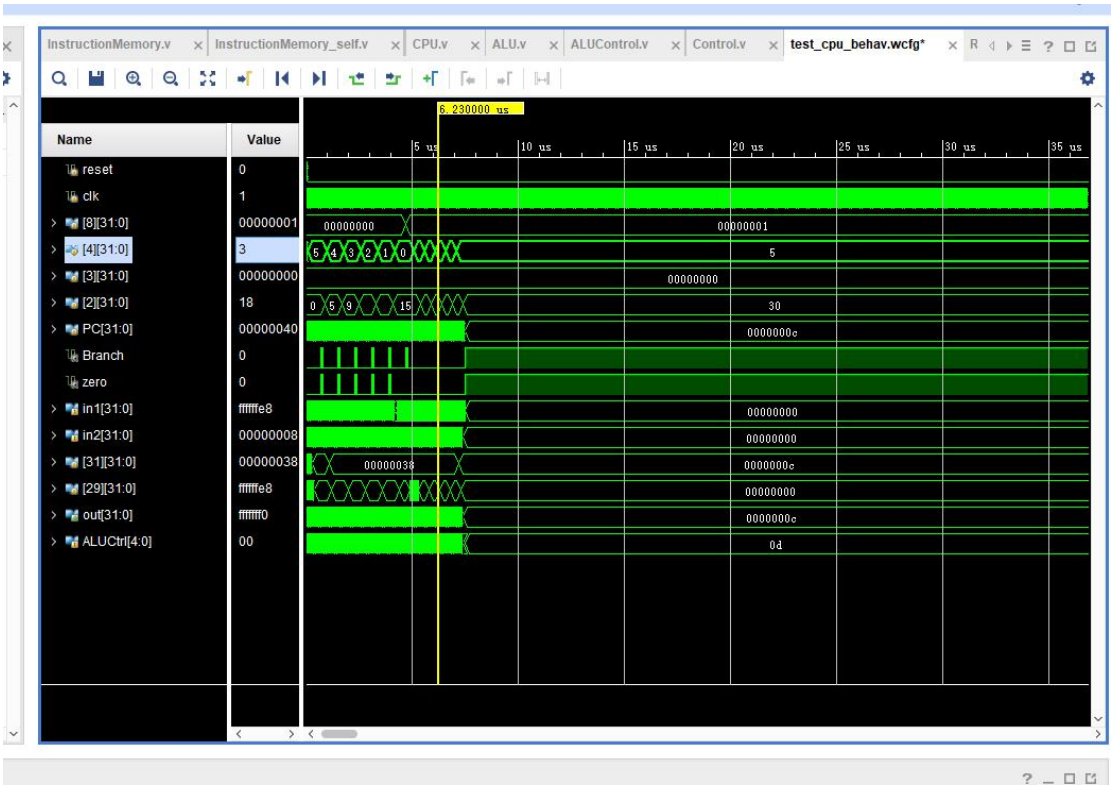
```
addi $sp,$sp,8  //移动栈指针
add $v0,$a0,$v0  //将$v0 与$a0 相加的值存入$v0
jr $ra //跳转到$ra 中存的地址
```

2.翻译成的机器码如下：

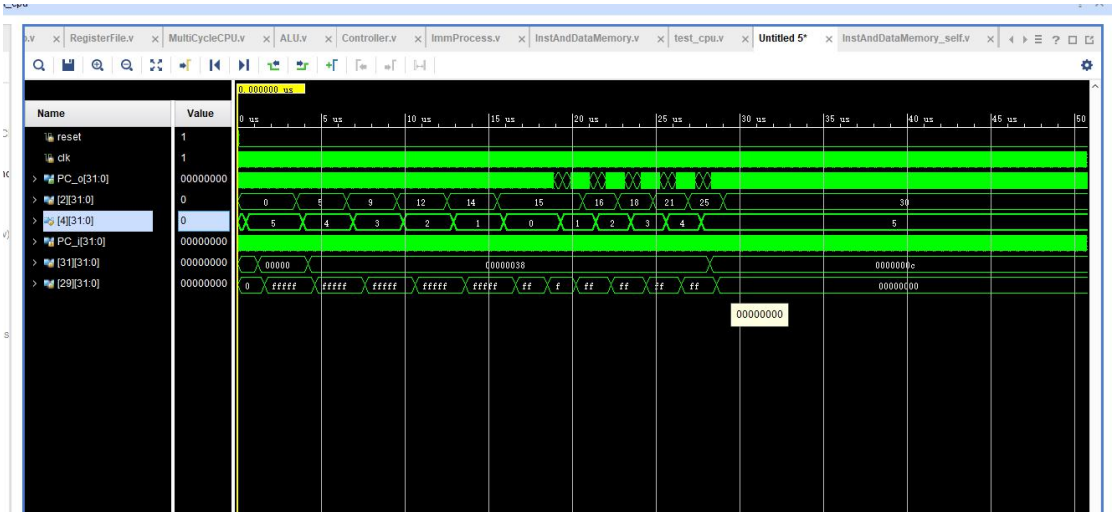
0x20040005
0x00001026
0x0c000004
0x1000ffff
0x23bdfff8
0xafbf0004
0xafa40000
0x28880001
0x11000002
0x23bd0008
0x03e00008
0x00821020
0x2084ffff
0x0c000004
0x8fa40000
0x8fbf0004
0x23bd0008
0x00821020
0x03e00008

新保存的指令存储器模块名为 InstructionMemory_self.v（单周期）和 InstAndDataMemory_self(多周期)。

3.足够长时间后$a0=5,$v0=30,符合理论分析。单周期仿真波形如下：



多周期仿真波形如下：



（b）资源与性能对比

单周期硬件资源开销如下：

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Bonded IOB (210)	BUFGCTRL (32)
CPU	3475	8442	1024	512	34	2
ALU0 (ALU)	309	54	0	0	0	0
ALU_control (ALUCont...	158	4	0	0	0	0
datamemory (DataMe...	2248	8192	1024	512	0	0
register_file (RegisterF...	760	160	0	0	0	0

静态时序分析报告如下：

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	90.166 ns	Worst Hold Slack (WHS):	0.291 ns	Worst Pulse Width Slack (WPWS):	4.500 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:	16736	Total Number of Endpoints:	16736	Total Number of Endpoints:	8385

All user specified timing constraints are met.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	90.166	17	18	112	PC_reg[8]/C	PC_reg[28]/D	9.683	4.209	5.474	100.0	CLK	CLK	
Path 2	90.280	16	17	112	PC_reg[8]/C	PC_reg[24]/D	9.569	4.095	5.474	100.0	CLK	CLK	
Path 3	90.297	18	19	112	PC_reg[8]/C	PC_reg[31]/D	9.552	4.244	5.308	100.0	CLK	CLK	
Path 4	90.340	18	19	112	PC_reg[8]/C	PC_reg[30]/D	9.509	4.341	5.168	100.0	CLK	CLK	
Path 5	90.394	15	16	112	PC_reg[8]/C	PC_reg[20]/D	9.455	3.981	5.474	100.0	CLK	CLK	
Path 6	90.411	17	18	112	PC_reg[8]/C	PC_reg[27]/D	9.438	4.130	5.308	100.0	CLK	CLK	
Path 7	90.454	17	18	112	PC_reg[8]/C	PC_reg[26]/D	9.395	4.227	5.168	100.0	CLK	CLK	
Path 8	90.465	18	19	112	PC_reg[8]/C	PC_reg[29]/D	9.384	4.217	5.167	100.0	CLK	CLK	
Path 9	90.508	14	15	112	PC_reg[8]/C	PC_reg[16]/D	9.341	3.867	5.474	100.0	CLK	CLK	
Path 10	90.525	16	17	112	PC_reg[8]/C	PC_reg[23]/D	9.324	4.016	5.308	100.0	CLK	CLK	

由报告可知单次计算所需要的最低延时为 9.324ns，T>100ns-90.166ns=9.834ns，因此最高时钟频率为 101.688MHz。

多周期硬件资源开销如下：

Name	Slice LUTs (20800)	Slice Registers (41600)	F7 Muxes (16300)	F8 Muxes (8150)	Bonded IOB (210)	BUFGCTRL (32)
MultiCycleCPU	4052	9534	1313	528	2	2
A_register (RegTemp_...	4	32	0	0	0	0
ALU0 (ALU)	0	32	0	0	0	0
ALUout_register (RegT...	307	32	0	0	0	0
B_register (RegTemp_...	0	130	0	0	0	0
Controller0 (Controller)	826	26	33	16	0	0
InstAndDataMemory0 (...	2208	8192	1024	512	0	0
InstReg0 (InstReg)	31	34	0	0	0	0
MDR (RegTemp_2)	0	32	0	0	0	0
PC0 (PC)	2	32	0	0	0	0
RegisterFile0 (Registe...	674	992	256	0	0	0

静态时序分析报告如下：

Setup

Worst Negative Slack (WNS):	94.325 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	18732

Hold

Worst Hold Slack (WHS):	0.134 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	18732

Pulse Width

Worst Pulse Width Slack (WPWS):	4.500 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	9503

All user specified timing constraints are met.

Name	Slack	Levels	Routes	High Fanout	From	To	Total Delay	Logic Delay	Net Delay	Requirement	Source Clock	Destination Clock	Exception
Path 1	94.325	6	7	32	InstReg0/Funct_reg[1]/C	PC0/PC_o_reg[0]/CE	5.293	2.099	3.194	100.0	CLK	CLK	
Path 2	94.325	6	7	32	InstReg0/Funct_reg[1]/C	PC0/PC_o_reg[10]/CE	5.293	2.099	3.194	100.0	CLK	CLK	
Path 3	94.325	6	7	32	InstReg0/Funct_reg[1]/C	PC0/PC_o_reg[11]/CE	5.293	2.099	3.194	100.0	CLK	CLK	
Path 4	94.325	6	7	32	InstReg0/Funct_reg[1]/C	PC0/PC_o_reg[12]/CE	5.293	2.099	3.194	100.0	CLK	CLK	
Path 5	94.325	6	7	32	InstReg0/Funct_reg[1]/C	PC0/PC_o_reg[13]/CE	5.293	2.099	3.194	100.0	CLK	CLK	
Path 6	94.325	6	7	32	InstReg0/Funct_reg[1]/C	PC0/PC_o_reg[14]/CE	5.293	2.099	3.194	100.0	CLK	CLK	
Path 7	94.325	6	7	32	InstReg0/Funct_reg[1]/C	PC0/PC_o_reg[15]/CE	5.293	2.099	3.194	100.0	CLK	CLK	
Path 8	94.325	6	7	32	InstReg0/Funct_reg[1]/C	PC0/PC_o_reg[16]/CE	5.293	2.099	3.194	100.0	CLK	CLK	
Path 9	94.325	6	7	32	InstReg0/Funct_reg[1]/C	PC0/PC_o_reg[17]/CE	5.293	2.099	3.194	100.0	CLK	CLK	
Path 10	94.325	6	7	32	InstReg0/Funct_reg[1]/C	PC0/PC_o_reg[18]/CE	5.293	2.099	3.194	100.0	CLK	CLK	

由报告可知单次计算所需要的最低延时为 5.293ns，T>100ns-94.325ns=5.675ns，最高时钟频率为 176.211MHz。

比较单周期和多周期处理器可知，周期处理器在硬件资源开销上略微大于单周期处理器，但是多周期处理器的最高时钟频率可比单周期处理器提高 70%。

五． 实验总结

次大作业中，我掌握了单周期和多周期处理器的控制信号、数据通路设计的原理和 RTL 级的实现方法，极大地加深了对 MIPS 处理器原理的理解和 verilog 的使用。同时，根据硬件资源占用情况和时序分析报告，深入理解了单周期与多周期在资源和性能上的设计折中。