电子科技大学计算机学院

标 准 实 验 报 告

（实验）课程名称 计算机组成与结构

电子科技大学教务处制表

题目名称：实验一 单周期处理器的设计与实现

学生姓名：郭亚龙　　学　　号：2019040401019

学生姓名：张杰瑞　　学　　号：2019040401006

课程教师：陈爱国　　实验时间：2020-2021 年度第 1 学期

任务及完成情况摘要(不超过 400 字)：

从总体架构到数据通路分析，再从各模块的设计实现，到 CPU 封装，我们成功完成了单周期处理器的设计和实现。单周期 CPU 各个必要的模块分别完成测评，功能正常；将各模块封装形成一个能够执行多条指令的完整单周期 CPU，也能正常运行。

该单周期 CPU 除了支持 add、sub、and、or、addi、andi、ori、lw、sw、beq、bne 和 j 等规定的基础 12 条指令外，我们还通过编写相关模块、更改拓展数据通路，使之额外实现了 CPU 对 xor、sll、srl、sra、jr、xori、lui、jal 等 8 条指令的支持。

我们设计了待运行的指令，并编写测试文件，总共 20 条指令全部完成测试，所有波形均分析完毕。相关指令运行结果与理论一致，实验成功。

指导老师评语：

功能实现及答辩（85 分）	课程设计报告（15 分）	总分

指导教师签字：_____

目录

实验一 单周期处理器的设计与实现

一、实验要求

（一）目的

该实验要求完成单周期处理器的设计和实现，要求至少支持 add、sub、and、or、addi、andi、ori、lw、sw、beq、bne 和 j 十二条指令。将 CPU 封装，并编写测试文件，解释波形，检验与理论波形是否一致。

（二）指令概述

下表为 MIPS 典型整数指令，其中未加粗部分为实验要求完成的基本指令，加粗部分为小组额外完成的指令。

表 1 MIPS 典型整数指令

R 型指令							
指令	[31:26]	[25:21]	[20:16]	[15:11]	[10:6]	[5:0]	功能
add	000000	rs	rt	rd			寄存器加
sub	000010	rs	rt	rd			寄存器减
and	000100	rs	rt	rd			寄存器与
or	000101	rs	rt	rd			寄存器或
xor	010000	rs	rt	rd			寄存器异或
sll	010100	000000	rt	rd	sa		左移
srl	010101	000000	rt	rd	sa		逻辑右移
sra	010110	000000	rt	rd	sa		算术右移
jr	001011	rs	000000	000000	000000	000000	寄存器跳转
I 型指令							
addi	000001	rs	rt	immediate			立即数加
andi	001110	rs	rt	immediate			立即数与
ori	000011	rs	rt	immediate			立即数或

xori	010001	rs	rt	immediate	立即数异或
lw	001000	rs	rt	offset	取数
sw	000111	rs	rt	offset	存数
beq	001001	rs	rt	offset	相等转移
bne	001111	rs	rt	offset	不等转移
lui	001111	00000	rt	immediate	设置高位
J 型指令					
j	001010	address			跳转
jal	001100	address			调用

（三）运行指令设计

下表为在该实验中我们设计的待运行指令，运行结果依赖指令存储器中指令。

表 2 指令集设计

地址	指令	含义
0	**add $s3,$s1,$s2**	$s3=$s1+$2
	32'b000000_00001_00010_00011_00000_100000;	
4	**sub $s3,$s1,$s2**	$s3=$s1-$s2
	32'b000000_00001_00010_00011_00000_100010;	
8	**and $s3,$s1,$s2**	$s3=$s1&$s2
	32'b000000_00001_00010_00011_00000_100100;	
12	**or $s3,$s1,$s2**	$s3=$s1|$s2
	32'b000000_00001_00010_00011_00000_100101;	
16	**xor $s3,$s1,$s2**	$s3=$s1⊕$s2
	32'b000000_00001_00010_00011_00000_100110;	
20	**sll $s3,$s2,3**	$s3=$s2<<3
	32'b000000_00000_00010_00011_00011_000000;	
24	**srl $s3,$s2,3**	$s3=$s2>>3
	32'b000000_00000_00010_00011_00011_000010;	
28	**sra $s3,$s2,3**	$s3=$s2>>>3
	32'b000000_00000_00010_00011_00011_000011;	
32	**jr $sa**	PC=$ra
	32'b000000_11111_00000_00000_00000_001000;	

36	**addi $s3,$s1,0x1234**	$s3=$s1+0x1234
	32'b001000_00001_00011_0001001000110100;	
40	**andi $s3,$s1,0x00ef**	$s3=$s1&0x00ef
	32'b001100_00001_00011_0000000011101111;	
44	**ori $s3,$s1,0x00ef**	$s3=$s1|0x00ef
	32'b001101_00001_00011_0000000011101111;	
48	**xori $s3,$s1,0x00ef**	$s3=$s1 ⊕ 0x00ef
	32'b001110_00001_00011_0000000011101111;	
52	**lw $s5,1($s3)**	$s5=memory[$s3+1]
	32'b100011_00011_00101_0000000000000001;	
56	**sw $s1,1($s3)**	memory[$s3+1]=$s1
	32'b101011_00011_00001_0000000000000001;	
60	**beq $s4,$s5,8**	if($s4==$s5) goto PC+4+32
	32'b000100_00100_00101_0000000000001000;	
64	**bne $s5,$s5,8**	if($s5!=$s5) goto PC+4+32
	32'b000101_00101_00101_0000000000001000;	
68	**lui $s6,0000000000001000**	$s6=0000000000001000<<16
	32'b001111_00000_00110_0000000000001000;	
72	**j 0x21913**	$PC=(PC+4)_{31..28}(0x21913)_{27..2}00$
	32'b000010_00000000100001100100010011;	
76	**jal 0x21907**	$ra=PC+4$ $PC=(PC+4)_{31..28}(0x21907)_{27..2}00$
	32'b000011_00000000100001100100000111;	

二、主要实验过程

（一）总体架构

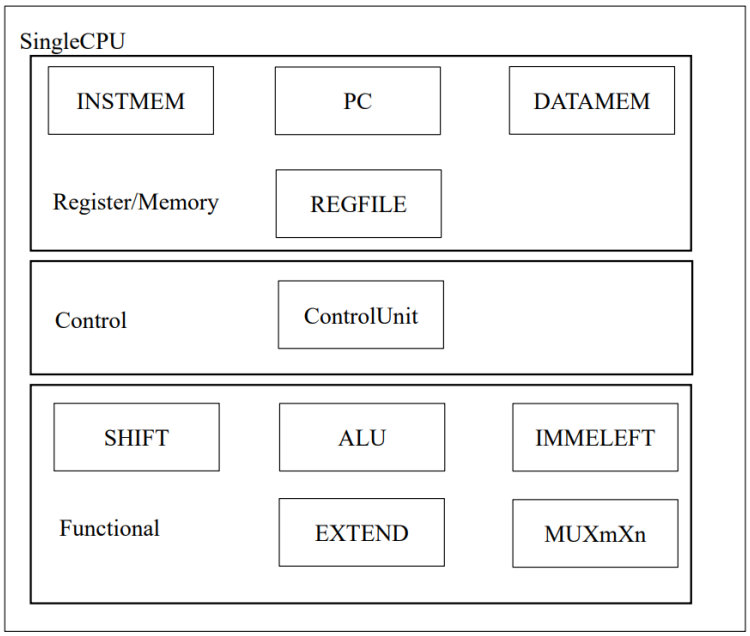

图 1 总体架构图

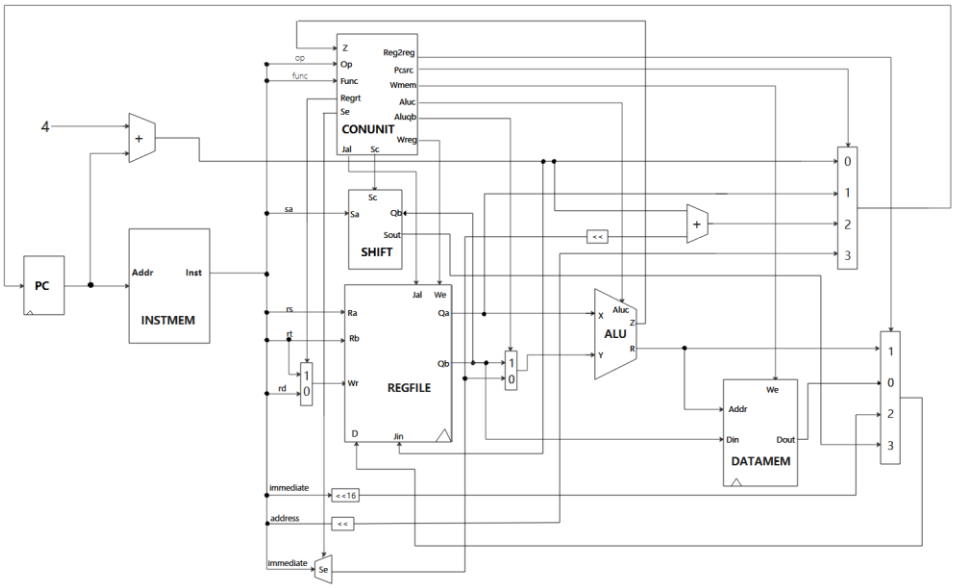

图 2 总体数据通路图

（二）数据通路分析

（1）R 型指令的数据通路

Step1:读取 PC，进入加法器进行 PC+4 操作，将结果作为下个周期的 PC 值

Step2:PC 输入地址到指令寄存器，指令寄存器根据地址[6:2]位输出指令

Step3:rs 输入到寄存器堆的 Ra，rt 输入到 Rb，rd 输入到 Wr

Step4:寄存器堆输出 Qa 到 ALU 的 X 口，Qb 到 ALU 的 Y 口

Step5:ALU 将结果经过选择器输出到寄存器堆的写入数据口，并根据 Wr 值存入对应寄存器

（2）R 型移位指令的数据通路

Step1:读取 PC，进入加法器进行 PC+4 操作，将结果作为下个周期的 PC 值

Step2:PC 输入地址到指令寄存器，指令寄存器根据地址[6:2]位输出指令

Step3:sa 输入 SHIFT 输入口，rt 输入到 Rb，rd 输入到 Wr

Step4:寄存器堆输出 Qb 到 SHIFT 的输入口

Step5:SHIFT 将结果经过选择器输出到寄存器堆的写入数据口，并根据 Wr 值存入对应寄存器

（3）I 型算术运算类指令的数据通路

Step1:读取 PC，进入加法器进行 PC+4 操作，将结果作为下个周期的 PC 值

Step2:PC 输入地址到指令寄存器，指令寄存器根据地址[6:2]位输出指令

Step3:rs 输入到寄存器堆的 Ra，rt 输入到 Wr，immediate 输入到扩展器

Step4:寄存器堆输出 Qa 到 ALU 的 X 口，扩展器结果输入到 ALU 的 Y 口

Step5:ALU 将结果经过选择器输出到寄存器堆的写入数据口，并根据 Wr 值存入对应寄存器

（4）I 型 lw 指令的数据通路

Step1:读取 PC，进入加法器进行 PC+4 操作，将结果作为下个周期的 PC 值

Step2:PC 输入地址到指令寄存器，指令寄存器根据地址[6:2]位输出指令

Step3:rs 输入到寄存器堆的 Ra，rt 输入到 Wr，immediate 输入到扩展器

Step4:寄存器堆输出 Qa 到 ALU 的 X 口，扩展器结果输入到 ALU 的 Y 口

Step5:ALU 将结果输入到数据寄存器的 Addr 端口

Step6:数据存储器根据地址[6:2]位访问并获取数据，Dout 端口将取得的数据输入到寄存器堆的写入数据口，根据 Wr 值存入对应寄存器

（5）I 型 sw 指令的数据通路

Step1:读取 PC，进入加法器进行 PC+4 操作，将结果作为下个周期的 PC 值

Step2:PC 输入地址到指令寄存器，指令寄存器根据地址[6:2]位输出指令

Step3:rs 输入到寄存器堆的 Ra，rt 输入到 Rb，immediate 输入到扩展器

Step4:寄存器堆输出 Qa 到 ALU 的 X 口，扩展器结果输入到 ALU 的 Y 口

Step5:ALU 将结果输入到数据寄存器的 Addr 端口

Step6:Qb 输入到数据寄存器的 Din 端口，数据存储器将寄存器 rt 的值写入该地址的存储单元

（6）I 型条件分支类指令的数据通路

Step1:读取 PC，进入加法器进行 PC+4 操作，将结果作为下个周期的备选 PC 值

Step2:PC 输入地址到指令寄存器，指令寄存器根据地址[6:2]位输出指令

Step3:rs 输入到寄存器堆的 Ra，rt 输入到 Rb，immediate 输入到扩展器

Step4:寄存器堆输出 Qa 到 ALU 的 X 接口，Qb 输入 ALU 的 Y 接口

Step5:扩展器结果左移 2 位输入到与 PC+4 的值相加的加法器中得到跳转地址

Step6:ALU 将输出 Z 传入控制单元进行判断

Step7:控制单元输出选择信号到 PC 选择器中

Step8:PC 选择器输出选择结果作为下一个 PC 值

（7）J 型指令的数据通路

Step1:读取 PC

Step2:PC 输入地址到指令寄存器，指令寄存器根据地址[6:2]位输出指令

Step3:PC+4 的最高 4 位与 address<<2 的值进行拼接得到跳转地址，并将结果输入到 PC 选择器中

Step4:选择器选择跳转地址作为下一个 PC 值

（8）jal 指令的数据通路

Step1:读取 PC，进入加法器进行 PC+4 操作

Step2:加法器将结果输入到寄存器堆的的 Jin 写入端口，准备写入$ra 寄存器

Step3:PC 输入地址到指令寄存器，指令寄存器根据地址[6:2]位输出指令

Step4:PC+4 的最高四位与 address<<2 的值进行拼接得到跳转地址，并将结果输入到 PC 选择器中

Step5:选择器选择跳转地址作为下一个 PC 值

（9）jr 指令的数据通路

Step1:读取 PC

Step2:PC 输入地址到指令寄存器，指令寄存器根据地址[6:2]位输出指令

Step3:rs 输入到寄存器堆的 Ra 口

Step4:寄存器堆将 Qa 输出到 PC 选择器中作为跳转地址

Step5:PC 选择器选择跳转地址作为下一个 PC 值

（10）lui 指令的数据通路

Step1:读取 PC，进入加法器进行 PC+4 操作，将结果作为下个周期的 PC 值
Step2:PC 输入地址到指令寄存器，指令寄存器根据地址[6:2]位输出指令
Step3:rt 输入到寄存器堆的 Wr 接口
Step3:将 immediate 左移 16 位，其结果输入到 D 的多路选择器中
Step4:选择器将结果输出到寄存器堆的写入数据口，并存入 rt 寄存器中

（三）各模块设计

（1）PC 寄存器

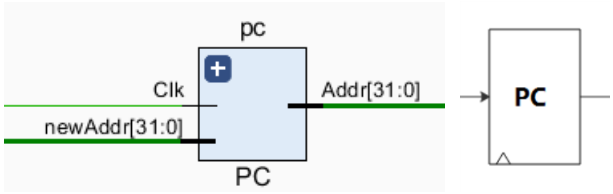

图 3 PC 寄存器

功能：根据时钟信号将输入的下一个 PC 值输出

（2）指令存储器

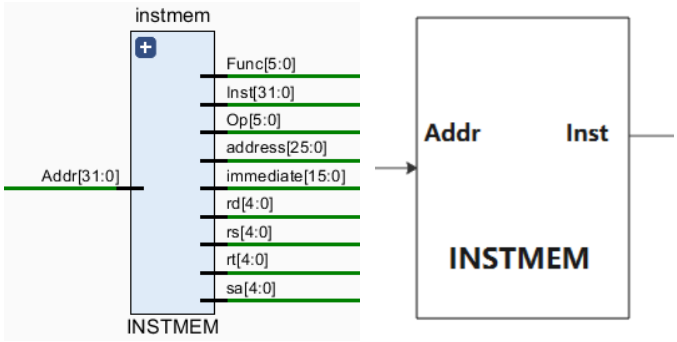

图 4 指令存储器

功能：存放实验所需使用的指令，根据 PC 输入的地址寻找对应指令并输出。预设指令的地址及作用详见一、实验要求

（三）运行指令设计。

（3）寄存器堆

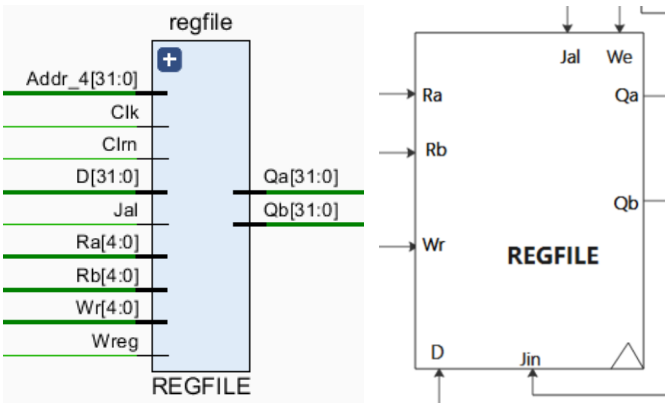

图 5 寄存器堆

功能：存放计算时需要使用的数据；根据 Ra，Rb 端口的输入来输出 Qa，Qb，根据 Wr 信号将 D 写入对应寄存器。初始化时我们将第 i 号寄存器的值设置为 i。

（4）算术逻辑单元

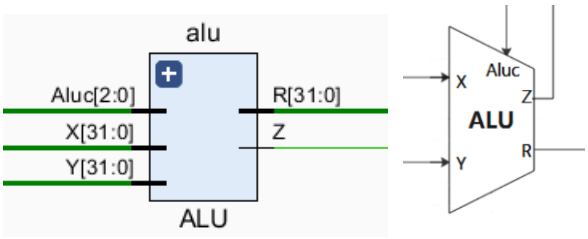

图 6 算术逻辑单元

功能：根据 Aluc 信号选择操作，对输入的 X 和 Y 进行加法、减法、按位与、按位或、按位异或的其中一种操作，并输出运算结果 R 和零标志信号 Z。

（5）数据存储器

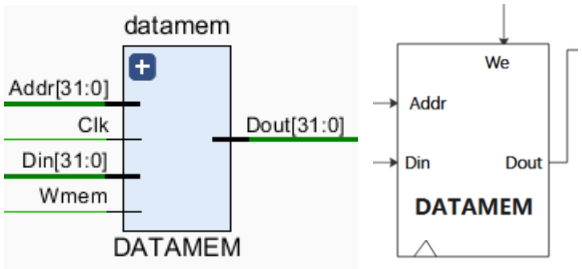

图 7 数据存储器

功能：存储数据，初始化时我们将第 i 个存储单元的值设置位为 i^2。根据 Addr 输入读出对应存储单元的数据，并通过 Dout 输出；或是根据 Addr 输入找到对应

存储单元，并将 Din 的输入值存储到对应存储单元。

（6）控制单元

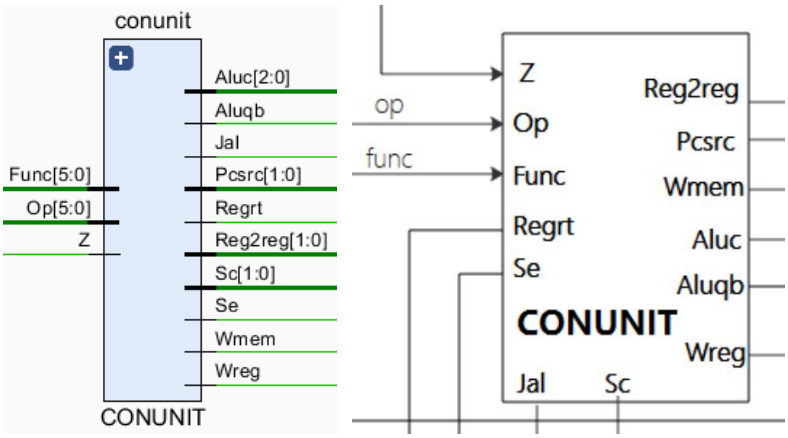

图 8 控制单元

功能：根据含指令 op、func 字段、零标志信号 Z 的输入信号来产生相应的控制信号，对整个 CPU 进行调控，使不同数据通路连通，得到正确的运行结果。

其具体输出信号与输入信号的关系如下表：

表 3 控制部件的输入输出信号真值表

输入端口				输出端口									
Op [5:0]	Func [5:0]	备注	Z	Regrt	Se	Wreg	Aluqb	Aluc [2:0]	Wmem	Pcsrc [1:0]	Reg2reg [1:0]	jal	Sc [1:0]
000000	100000	add	X	0	0	1	1	000	0	00	01	0	00
000000	100010	sub	X	0	0	1	1	001	0	00	01	0	00
000000	100100	and	X	0	0	1	1	010	0	00	01	0	00
000000	100101	or	X	0	0	1	1	011	0	00	01	0	00
001000	—	addi	X	1	1	1	0	000	0	00	01	0	00
001100	—	andi	X	1	0	1	0	010	0	00	01	0	00
001101	—	ori	X	1	0	1	0	011	0	00	01	0	00
100011	—	lw	X	1	1	1	0	000	0	00	00	0	00
101011	—	sw	X	1	1	0	0	000	1	00	01	0	00
000100	—	beq	0	1	1	0	1	001	0	00	01	0	00
000100	—	beq	1	1	1	0	1	001	0	10	01	0	00
000000	100110	xor	X	0	0	1	1	100	0	00	01	0	00
001110	—	xori	X	1	0	1	0	100	0	00	01	0	00
000000	000000	sll	X	0	0	1	1	000	0	00	11	0	00
000000	000010	srl	X	0	0	1	1	000	0	00	11	0	01
000000	000011	sra	X	0	0	1	1	000	0	00	11	0	10
000101	—	bne	0	1	1	0	1	001	0	10	01	0	00
000101	—	bne	1	1	1	0	1	001	0	00	01	0	00
001111	—	lui	X	1	1	1	0	000	0	00	10	0	00

000010	—	j	X	1	0	0	1	000	0	11	01	0	00
000011	—	jal	X	1	0	0	1	000	0	11	01	1	00
000000	001000	jr	X	0	0	1	1	000	0	01	01	0	00

（7）扩展器

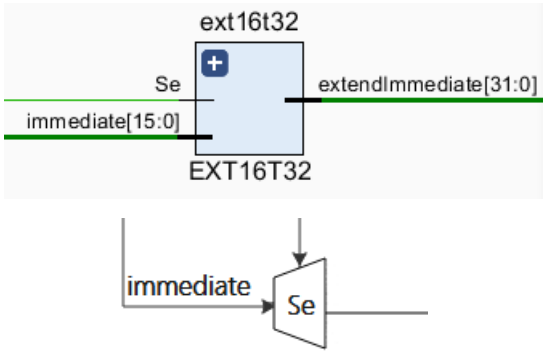

图 9 扩展器

功能：将 16 位立即数扩展成 32 位，并根据输入信号区分是零扩展还是符号扩展。

（8）多路选择器

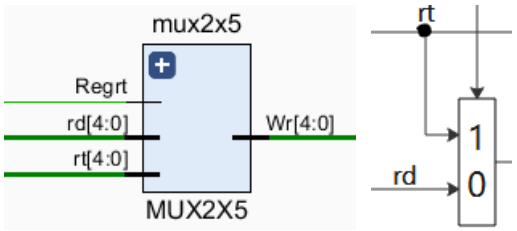

图 10 五位 2 选 1 多路选择器

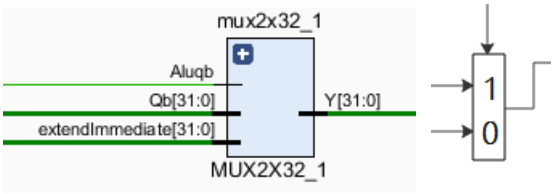

图 11 三十二位 2 选 1 多路选择器

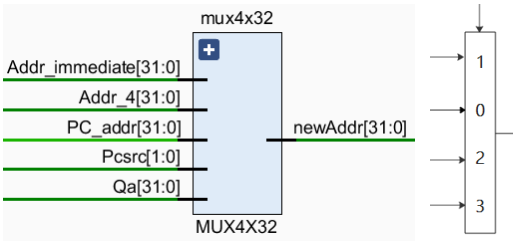

图 12 三十二位 4 选 1 多路选择器

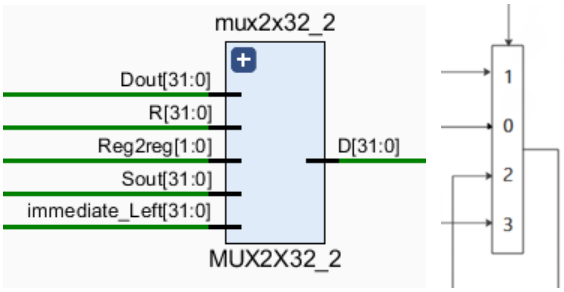

图 13 三十二位 4 选 1 多路选择器

功能：根据输入的控制信号判断从多路输入中选择哪个信号作为输出信号。

（9）移位器

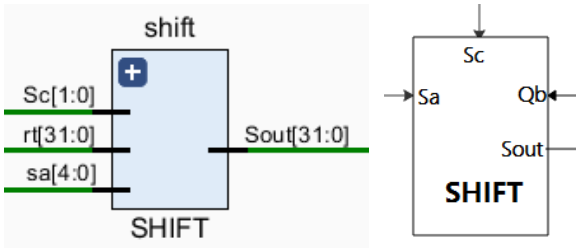

图 14 移位器

功能：根据输入的控制信号 Sc 判断移位方式，根据 sa 判断移位位数，并将移位结果通过 Sout 输出。

（10）支持 lui 的功能部件

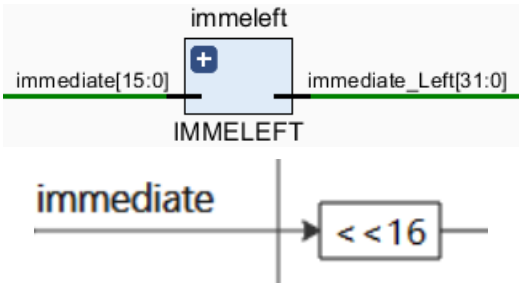

图 15 支持 lui 的功能部件

功能：为支持 lui 设置高位操作，将输入的立即数左移 16 位，并将结果输出。

（四）封装

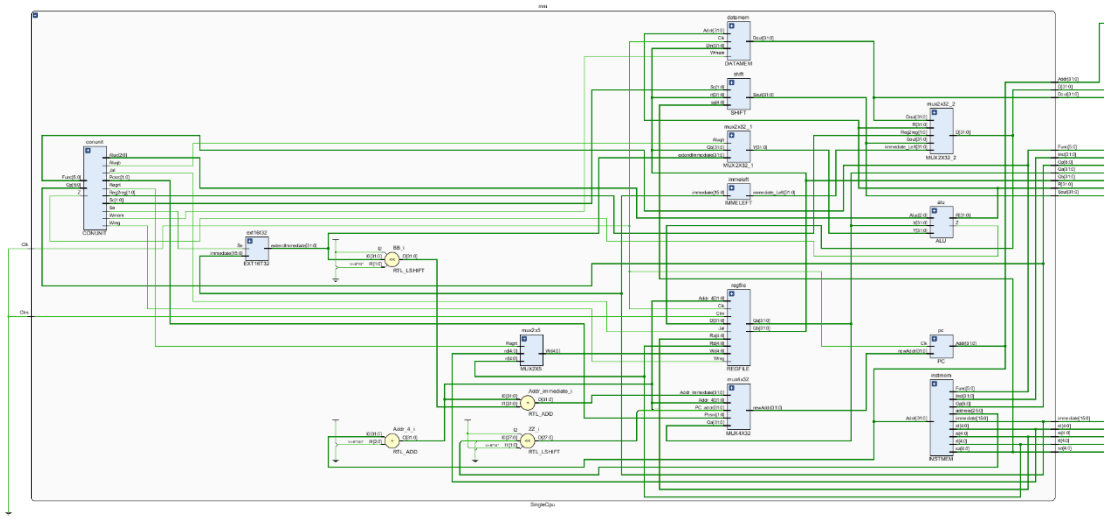

图 16 封装完成的单周期 CPU（全览）

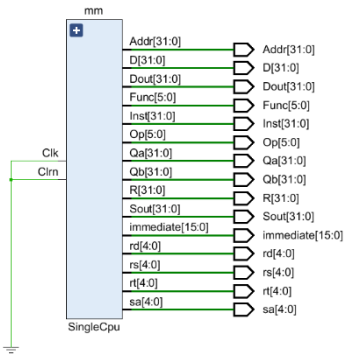

图 17 封装完成的单周期 CPU

功能：将各模块连接起来，形成一个完整的单周期 CPU。预留输入输出接口，便于输入测试信号进行测试，以及方便输出波形。

三、主要实现代码

（一）主要功能代码

（1）PC 寄存器

`timescale 1ns/1ps

```
module PC(Clk,newAddr,Addr);
input Clk;                          //输入时钟信号
input [31:0] newAddr;               //输入新地址
output reg [31:0] Addr;             //当前输出地址
initial    begin
      Addr<=0;
end
always@(posedge Clk)                //在时钟上升沿写入新指令
begin
      Addr <= newAddr;
end
endmodule
```

（2）指令存储器

```
`timescale 1ns/1ps
module INSTMEM(Addr,Op,rs,rt,rd,sa,immediate,Func,address,Inst);
input [31:0] Addr;
output [5:0] Op;
output [5:0] Func;
output [4:0] rs;
output [4:0] rt;
output [4:0] rd;
output [4:0] sa;
output [15:0] immediate;
output [25:0] address;
output [31:0] Inst;                 //以上为输出指令的各字段
wire [31:0] ram [0:19];             //定义 20 个 32 位寄存器存储指令
assign ram[5'h00] = 32'b000000_00001_00010_00011_00000_100000; //add
assign ram[5'h01] = 32'b000000_00001_00010_00011_00000_100010; //sub
assign ram[5'h02] = 32'b000000_00001_00010_00011_00000_100100; //and
assign ram[5'h03] = 32'b000000_00001_00010_00011_00000_100101; //or
assign ram[5'h04] = 32'b000000_00001_00010_00011_00000_100110; //xor
assign ram[5'h05] = 32'b000000_00000_00010_00011_00011_000000; //sll
assign ram[5'h06] = 32'b000000_00000_00010_00011_00011_000010; //srl
assign ram[5'h07] = 32'b000000_00000_00010_00011_00011_000011; //sra
assign ram[5'h08] = 32'b000000_11111_00000_00000_00000_001000; //jr $ra
assign ram[5'h09] = 32'b001000_00001_00011_0001001000110100;   //addi
assign ram[5'h0a] = 32'b001100_00001_00011_0000000011101111;   //andi
assign ram[5'h0b] = 32'b001101_00001_00011_0000000011101111;   //ori
assign ram[5'h0c] = 32'b001110_00001_00011_0000000011101111;   //xori
assign ram[5'h0d] = 32'b100011_00011_00101_0000000000000001;   //lw
assign ram[5'h0e] = 32'b101011_00011_00001_0000000000000001;   //sw
assign ram[5'h0f] = 32'b000100_00100_00101_0000000000001000;   //beq
```

```
assign ram[5'h10] = 32'b000101_00101_00101_0000000000001000;    //bne
assign ram[5'h11] = 32'b001111_00000_00110_0000000000001000;    //lui
assign ram[5'h12] = 32'b000010_00000000100001100100010011;       //j
assign ram[5'h13] = 32'b000011_00000001000011001000_00111;      //jal
assign Inst = ram[Addr[6:2]];
assign Op = ram[Addr[6:2]][31:26];
assign rs = ram[Addr[6:2]][25:21];
assign rt = ram[Addr[6:2]][20:16];
assign rd = ram[Addr[6:2]][15:11];
assign sa = ram[Addr[6:2]][10:6];
assign immediate = ram[Addr[6:2]][15:0];
assign Func = ram[Addr[6:2]][5:0];
assign address = ram[Addr[6:2]][25:0];
endmodule
```

（3）寄存器堆

```
`timescale 1ns/1ps
module REGFILE(Wreg,Ra,Rb,Wr,Clk,D,Qa,Qb,Addr_4,Jal,Clrn);
input Clk,Wreg,Jal,Clrn; //输入时钟信号、写入信号、jal 指令写入信号、清零信号
input [4:0] Ra;
input [4:0] Rb;
input [4:0] Wr;
input [31:0] D;
input [31:0] Addr_4;         //输入 PC+4  的值
output [31:0] Qa,Qb;
reg [31:0] register[0:31];
integer i;
initial begin         //令前 31 个寄存器存储：第 i 个寄存器=i。第 32 个存储值为 36
   for(i = 0; i < 32; i = i + 1)
      begin
         if(i!=31) begin
              register[i] <= i;
              end
         else begin
              register[i] <= 36;          //目的是使 jr 指令跳转到其下一条 addi 指令
              end
      end
end
always@(posedge Clk)            //jal 指令在时钟上升沿将当前 PC+4 写入$ra
   begin
      if(Jal)
         register[31]=Addr_4;
   end
```

```
assign Qa=register[Ra];            //输出 Qa,Qb
assign Qb=register[Rb];
always@(posedge Clk)
   begin
    if(Clrn)
        begin
              if (Wreg&&Wr != 0) begin
                  register[Wr]=D;       //写入目的寄存器
              end
          end
      else    begin
            for(i = 0; i < 32; i = i + 1)
              begin
              register[i] <= 0;          //Clrn=0 时实现将寄存器堆清零
              end
      end
   end
endmodule
```

（4）算术逻辑单元

```
`timescale 1ns/1ps
module ALU(X,Y,Aluc,R,Z);
input [31:0] X,Y;
input [2:0] Aluc;
output reg [31:0] R;
output reg Z;
always@(*)
   begin
    case(Aluc)
       3'b000:R=X+Y;          //加
       3'b001:R=X-Y;          //减
       3'b010:R=X&Y;          //与
       3'b011:R=X|Y;          //或
       3'b100:R=(X&(~Y))|((~X)&Y);   //异或
       default:R=0;
    endcase
if(R)
       Z=0;
    else
       Z=1;
end
endmodule
```

（5）数据存储器

```
`timescale 1ns/1ps
module DATAMEM(Wmem,Addr,Din,Dout,Clk);
input [31:0] Addr,Din;
input Clk,Wmem;
output [31:0] Dout;
reg [31:0] Ram[0:31];            //定义 32 个 32 位寄存器
assign Dout=Ram[Addr[6:2]];
always@(posedge Clk)         //时钟上升沿写入
   begin
      if(Wmem)
          Ram[Addr[6:2]]<=Din;
end
integer i;
initial begin
for(i=0;i<32;i=i+1)
      Ram[i]=i*i;               //初始时令每个寄存器的值为其编号的平方
end
endmodule
```

（6）控制单元

```
`timescale 1ns / 1ps
module
CONUNIT(Op,Func,Z,Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal);
input [5:0] Op,Func;
input Z;
output reg Regrt,Se,Wreg,Aluqb,Wmem,Jal;       //Jal 为 jal 指令的写入信号
output reg [1:0] Pcsrc,Sc,Reg2reg;                  //Sc 为 sll、sra、srl 指令的选择信号
output reg [2:0] Aluc;
initial
   begin
{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b0000000000_00_00_
0;
end
always@(Op or Z or Func)     //按照控制信号输出表规定每条指令的控制信号
   begin
      case(Op)
         6'b000000:
           begin
             case(Func)
6'b100000:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b0011_000
```

```
_000_01_00_0;//add
6'b100010:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b0011_001
_000_01_00_0;//sub
6'b100100:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b0011_010
_000_01_00_0;//and
6'b100101:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b0011_011
_000_01_00_0;//or
6'b100110:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b0011_100
_000_01_00_0;//xor
6'b000000:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b0011_000
_000_11_00_0;//sll
6'b000010:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b0011_000
_000_11_01_0;//srl
6'b000011:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b0011_000
_000_11_10_0;//sra
6'b001000:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b0011_000
_001_01_00_0;//jr
            endcase
          end
6'b001000:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1110_000
_000_01_00_0;//addi
6'b001100:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1010_010
_000_01_00_0;//andi
6'b001101:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1010_011
_000_01_00_0;//ori
6'b001110:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1010_100
_000_01_00_0;//xori
6'b100011:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1110_000
_000_00_00_0;//lw
6'b101011:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1100_000
_100_01_00_0;//sw
6'b001111:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1110_000
_000_10_00_0;//lui
        6'b000100:
          begin
    case(Z)
1'b0:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1101_001_000_
01_00_0;//beq
1'b1:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1101_001_010_
01_00_0;//beq
    endcase
          end
        6'b000101:
          begin
```

```
      case(Z)
1'b0:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1101_001_010_
01_00_0;//bne
1'b1:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1101_001_000_
01_00_0;//bne
      endcase
           end
6'b000010:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1001_000
_011_01_00_0;//j
6'b000011:{Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal}=15'b1001_000
_011_01_00_1;//jal
       endcase
   end
endmodule
```

（7）扩展器

```
`timescale 1ns / 1ps
module EXT16T32(Se,immediate,extendImmediate);
input Se;
input [15:0] immediate;
output [31:0] extendImmediate;     //输出拓展后的 32 位立即数
wire [31:0] E0,E1;
wire [15:0] e={16{immediate[15]}};
parameter z=16'b0;
assign E0={z,immediate};       //零拓展
assign E1={e,immediate};       //符号拓展
function [31:0]select;
input [31:0] E0,E1;
input Se;
case(Se)
1'b0:select=E0;
1'b1:select=E1;
endcase
endfunction
assign extendImmediate=select(E0,E1,Se);
endmodule
```

（8）多路选择器

（ALU 的 Y 输入端口前 32 位 2 选 1 选择器）

```
`timescale 1ns/1ps
module MUX2X32_1(Qb,extendImmediate,Aluqb,Y);
```

```
input [31:0] extendImmediate,Qb;          //输入拓展完的立即数、$rt 寄存器的值
output [31:0] Y;                          //输出 Y
input Aluqb;
function [31:0]select;
input [31:0] extendImmediate,Qb;
input [1:0] Aluqb;
case(Aluqb)
1'b0:select=extendImmediate;
1'b1:select=Qb;
endcase
endfunction
assign Y=select(extendImmediate,Qb,Aluqb);
endmodule
```

（REGFILE 的 Wr 输入端口前 5 位 2 选 1 选择器）

```
`timescale 1ns/1ps
module MUX2X5(rt,rd,Regrt,Wr);
input [4:0] rt,rd;
output [4:0] Wr;
input Regrt;
function [4:0]select;
input [4:0] rd,rt;
input Regrt;
case(Regrt)
1'b0:select=rd;
1'b1:select=rt;
endcase
endfunction
assign Wr=select(rd,rt,Regrt);
endmodule
```

（输出 D 的 32 位 4 选 1 选择器）

```
`timescale 1ns/1ps
module MUX2X32_2(R,Dout,Sout,immediate_Left,Reg2reg,D);
input [31:0] Dout,R,Sout,immediate_Left;
output [31:0] D;
input [1:0] Reg2reg;
function [31:0]select;
input [31:0] R,Dout,Sout,immediate_Left;
input [1:0] Reg2reg;
case(Reg2reg)
2'b01:select=R;                   //输出 R
2'b00:select=Dout;                //输出 DATAMEM 的 Dout
2'b10:select=immediate_Left; //输出立即数左移 16 位后的 32 位数  即 lui 指令输出
```

```
2'b11:select=Sout;                //输出移位后的 Sout  即 sll、sra、srl 指令输出
endcase
endfunction
assign D=select(R,Dout,Sout,immediate_Left,Reg2reg);
endmodule
```

（输出新地址的 32 位 4 选 1 选择器）

```
`timescale 1ns/1ps
module MUX4X32(Addr_4,Addr_immediate,PC_addr,Pcsrc,newAddr,Qa);
input [31:0] Addr_4,Addr_immediate,PC_addr,Qa;
input [1:0] Pcsrc;
output [31:0] newAddr;
function [31:0]select;
input [31:0] Addr_4,Addr_immediate,PC_addr,Qa;
input [1:0] Pcsrc;
case(Pcsrc)
2'b00:select=Addr_4;              //输出 PC+4
2'b01:select=Qa;                  //输出$rs 的值  即 jr 指令跳转地址
2'b10:select=Addr_immediate;   //输出 offset<<2+4+PC  即 beq、bne 指令跳转地址
2'b11:select=PC_addr;            //输出 PC+4 的最高四位与 address<<2 的拼接  j、jal
endcase
endfunction
assign newAddr=select(Addr_4,Addr_immediate,PC_addr,Qa,Pcsrc);
endmodule
```

（9）移位器

```
`timescale 1ns/1ps
module SHIFT(rt,sa,Sc,Sout);       //实现 sll、sra、srl 指令的移位功能
input [31:0] rt;
input [4:0] sa;
input [1:0] Sc;
output [31:0] Sout;
wire [31:0] a,c;
reg [31:0] b;
always @(*) begin
      case(sa)
            5'b00000:b<=rt;
            5'b00001:b<={{rt[31]},rt[31:1]};
            5'b00010:b<={{2{rt[31]}},rt[31:2]};
            5'b00011:b<={{3{rt[31]}},rt[31:3]};
            5'b00100:b<={{4{rt[31]}},rt[31:4]};
            5'b00101:b<={{5{rt[31]}},rt[31:5]};
            5'b00110:b<={{6{rt[31]}},rt[31:6]};
```

```
            5'b00111:b<={{7{rt[31]}},rt[31:7]};
            5'b01000:b<={{8{rt[31]}},rt[31:8]};
            5'b01001:b<={{9{rt[31]}},rt[31:9]};
            5'b01010:b<={{10{rt[31]}},rt[31:10]};
            5'b01011:b<={{11{rt[31]}},rt[31:11]};
            5'b01100:b<={{12{rt[31]}},rt[31:12]};
            5'b01101:b<={{13{rt[31]}},rt[31:13]};
            5'b01110:b<={{14{rt[31]}},rt[31:14]};
            5'b01111:b<={{15{rt[31]}},rt[31:15]};
            5'b10000:b<={{16{rt[31]}},rt[31:16]};
            5'b10001:b<={{17{rt[31]}},rt[31:17]};
            5'b10010:b<={{18{rt[31]}},rt[31:18]};
            5'b10011:b<={{19{rt[31]}},rt[31:19]};
            5'b10100:b<={{20{rt[31]}},rt[31:20]};
            5'b10101:b<={{21{rt[31]}},rt[31:21]};
            5'b10110:b<={{22{rt[31]}},rt[31:22]};
            5'b10111:b<={{23{rt[31]}},rt[31:23]};
            5'b11000:b<={{24{rt[31]}},rt[31:24]};
            5'b11001:b<={{25{rt[31]}},rt[31:25]};
            5'b11010:b<={{26{rt[31]}},rt[31:26]};
            5'b11011:b<={{27{rt[31]}},rt[31:27]};
            5'b11100:b<={{28{rt[31]}},rt[31:28]};
            5'b11101:b<={{29{rt[31]}},rt[31:29]};
            5'b11110:b<={{30{rt[31]}},rt[31:30]};
            5'b11111:b<={{31{rt[31]}},rt[31]};
            default:b<=rt;
        endcase
end
assign a=rt>>sa;
assign c=rt<<sa;
function [31:0]select;
input [31:0] a,b,c;
input [1:0] Sc;
case(Sc)
        2'b00:select=c;             //shift left    左移
        2'b01:select=a;             //Logical shift right    逻辑右移
        2'b10:select=b;           //Count shift right    算数右移
endcase
endfunction
assign Sout=select(a,b,c,Sc);
endmodule
```

（10）支持 lui 的功能部件

```
`timescale 1ns/1ps
module IMMELEFT(immediate,immediate_Left);     //实现 lui 指令的设置高位功能
input [15:0] immediate;
output [31:0] immediate_Left;
assign immediate_Left=immediate<<16;
endmodule
```

（二）封装代码

```
`timescale 1ns/1ps
module
SingleCpu(Clk,Clrn,Op,rs,rt,rd,sa,immediate,Inst,Qa,Qb,D,Dout,Sout,Func,Addr,R);
input Clk;
input Clrn;
output [5:0] Op;
output [4:0] rs;
output [4:0] rt;
output [4:0] rd;
output [4:0] sa;
output [15:0] immediate;
output [31:0] Inst;
output [31:0] Qa;
output [31:0] Qb;
output [31:0] D;
output [31:0] Dout;
output [31:0] Sout;
output [5:0] Func;
output [31:0] Addr;
output [31:0] R;
wire [2:0] Aluc;
wire [1:0] Sc;
wire [31:0] Y,newAddr;
wire [31:0] Addr_4, extendImmediate, Addr_immediate,PC_addr,immediate_Left;
wire [4:0] Wr;
wire [25:0] address;
wire [27:0] ZZ;
wire [31:0] BB;
wire [3:0] AA;
wire [1:0] Pcsrc;
wire [1:0] Reg2reg;
```

```
wire Z,Regrt,Se,Wreg,Aluqb,Wmem,Jal;
PC pc(Clk,newAddr,Addr);
INSTMEM instmem(Addr,Op,rs,rt,rd,sa,immediate,Func,address,Inst);
REGFILE regfile(Wreg,rs,rt,Wr,Clk,D,Qa,Qb,Addr_4,Jal,Clrn);
ALU alu(Qa,Y,Aluc,R,Z);
DATAMEM datamem(Wmem,R,Qb,Dout,Clk);
CONUNIT
conunit(Op,Func,Z,Regrt,Se,Wreg,Aluqb,Aluc,Wmem,Pcsrc,Reg2reg,Sc,Jal);
EXT16T32 ext16t32(Se,immediate,extendImmediate);
MUX2X32_1 mux2x32_1(Qb,extendImmediate,Aluqb,Y);
MUX2X32_2 mux2x32_2(R,Dout,Sout,immediate_Left,Reg2reg,D);
MUX2X5 mux2x5(rt,rd,Regrt,Wr);
SHIFT shift(Qb,sa,Sc,Sout);
IMMELEFT immeleft(immediate,immediate_Left);
assign Addr_4 = Addr + 4;                    //实现 PC+4  功能
assign BB=extendImmediate<<2;
assign Addr_immediate = Addr_4+BB;       //实现 offset<<2+4+PC  功能
assign ZZ=address<<2;
assign AA=Addr_4[31:28];
assign PC_addr={AA,ZZ};       //实现 PC+4 的最高四位与 address<<2 的拼接功能
MUX4X32 mux4x32(
.Addr_4(Addr_4),
.Addr_immediate(Addr_immediate),
.PC_addr(PC_addr),
.Pcsrc(Pcsrc),
.newAddr(newAddr),
.Qa(Qa)
);
endmodule
```

（三）仿真代码

```
`timescale 1ns/1ps
module
SingleCpu_tb(Op,rs,rt,rd,sa,immediate,Inst,Qa,Qb,D,Addr,Func,Dout,Sout,R);
reg Clk,Clrn;
output [5:0] Op;
output [4:0] rs;
output [4:0] rt;
output [4:0] rd;
output [4:0] sa;
output [15:0] immediate;
output [31:0] Inst;
```

```
output [31:0] Qa;
output [31:0] Qb;
output [31:0] D;
output [31:0] Addr;
output [5:0] Func;
output [31:0] Dout;
output [31:0] Sout;
output [31:0] R;
wire [5:0] Op;
wire [4:0] rs;
wire [4:0] rt;
wire [4:0] rd;
wire [4:0] sa;
wire [15:0] immediate;
wire [31:0] Inst;
wire [31:0] Qa;
wire [31:0] Qb;
wire [31:0] D;
wire [31:0] Dout;
wire [31:0] Sout;
wire [5:0] Func;
wire [31:0] Addr;
wire [31:0] R;
SingleCpu mm(
        .Clk(Clk),
       .Op(Op),
       .rs(rs),
       .rt(rt),
       .rd(rd),
       .sa(sa),
       .immediate(immediate),
        .Inst(Inst),
       .Qa(Qa),
       .Qb(Qb),
       .D(D),
       .Dout(Dout),
       .Sout(Sout),
       .Func(Func),
       .Addr(Addr),
       .R(R)
);
initial begin
Clk=0;
Clrn=1;
```

```
#50;
     Clk=!Clk;
     Clrn=0;
forever #50 begin
   Clk=!Clk;
   Clrn=0;
end              //前 50ns 是第一条指令，之后每 100ns 是一条指令、一个时钟周期
end
endmodule
```

四、仿真结果

（一）预期仿真结果

表 4 预期仿真结果（从上至下依次出现）

地址	指令	结果
0	**add $s3,$s1,$s2**	$s3=$s1+$2=1+2=0x00000003
	32'b000000_00001_00010_00011_00000_100000;	
4	**sub $s3,$s1,$s2**	$s3=$s1-$s2=1-2=0xffffffff
	32'b000000_00001_00010_00011_00000_100010;	
8	**and $s3,$s1,$s2**	$s3=$s1&$s2=1&2=0x00000000
	32'b000000_00001_00010_00011_00000_100100;	
12	**or $s3,$s1,$s2**	$s3=$s1|$s2=1|2=0x00000003
	32'b000000_00001_00010_00011_00000_100101;	
16	**xor $s3,$s1,$s2**	$s3=$s1⊕$s2=1⊕2=0x00000003
	32'b000000_00001_00010_00011_00000_100110;	
20	**sll $s3,$s2,3**	$s3=$s2<<3=2<<3=0x00000010
	32'b000000_00000_00010_00011_00011_000000;	
24	**srl $s3,$s2,3**	$s3=$s2>>3=2>>3=0x00000000
	32'b000000_00000_00010_00011_00011_000010;	
28	**sra $s3,$s2,3**	$s3=$s2>>>32>>>3=0x00000000
	32'b000000_00000_00010_00011_00011_000011;	
32	**jr $sa**	PC=$ra=36
	32'b000000_11111_00000_00000_00000_001000;	
36	**addi $s3,$s1,0x1234**	$s3=$s1+0x1234=1+0x1234=0x1235

	32'b001000_00001_00011_0001001000110100;	
40	**andi $s3,$s1,0x00ef**	$s3=$s1&0x00ef=1&0x00ef=0x00000001
	32'b001100_00001_00011_0000000011101111;	
44	**ori $s3,$s1,0x00ef**	$s3=$s1|0x00ef1|0x00ef=0x000000ef
	32'b001101_00001_00011_0000000011101111;	
48	**xori $s3,$s1,0x00ef**	$s3=$s1⊕0x00ef=1⊕0x00ef=0x000000ee
	32'b001110_00001_00011_0000000011101111;	
52	**lw $s5,1($s3)**	$s5=memory[$s3+1]=memory[0x000000ef] =ram[27]=729=0x000002d9
	32'b100011_00011_00101_0000000000000001;	
56	**sw $s1,1($s3)**	memory[$s3+1]=$s1=0x00000001
	32'b101011_00011_00001_0000000000000001;	
60	**beq $s4,$s5,8**	($s4=4)!=($s5=0x2d9) 故不跳转,PC=PC+4
	32'b000100_00100_00101_0000000000001000;	
64	**bne $s5,$s5,8**	($s5=0x2d9)==($s5=0x2d9) 故不跳 转,PC=PC+4
	32'b000101_00101_00101_0000000000001000;	
68	**lui $s6,0000000000001000**	$s6=0000000000001000<<16=0x00080000
	32'b001111_00000_00110_0000000000001000;	
72	**j 0x21913**	PC=(PC+4)$_{31..28}$(0x21913)$_{27..2}$00=0x000864c
	32'b000010_00000000100001100100010011;	
76	**jal 0x21907**	$ra=PC+4=0x0000004c PC=(PC+4)$_{31..28}$(0x21907)$_{27..2}$00 =0x0008641c
	32'b000011_000000001000 0110 0100 000111;	

（二）实际仿真结果

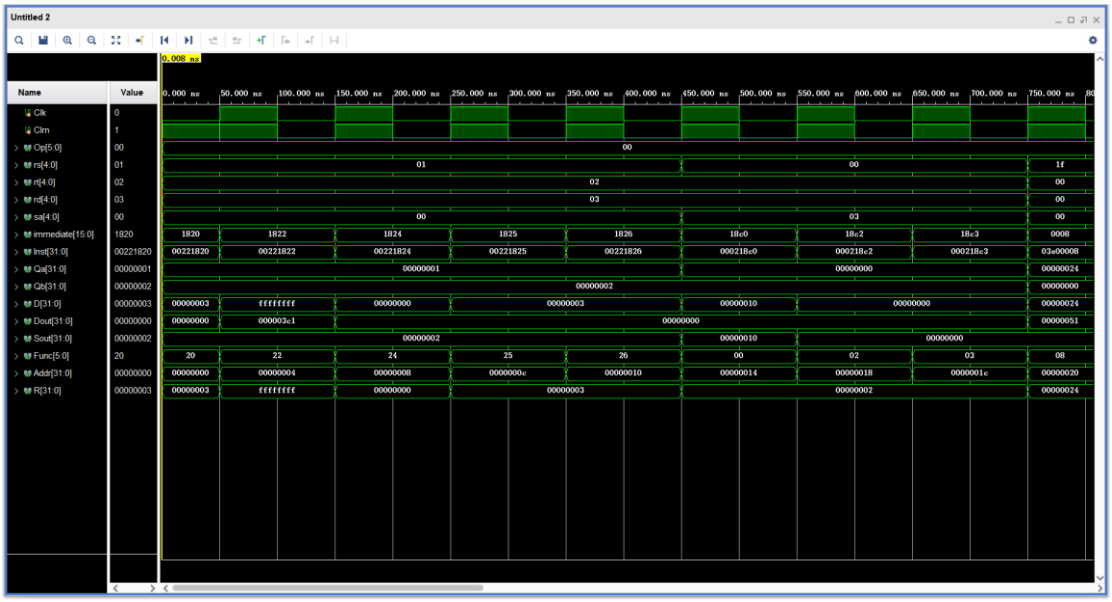

图 18 实际仿真波形（第一段）

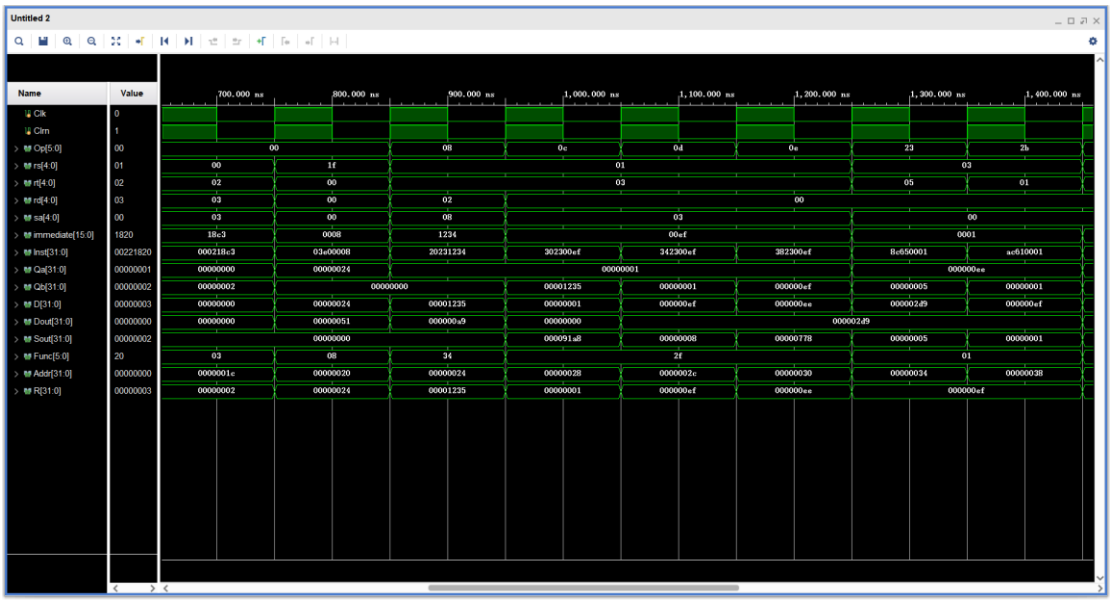

图 19 实际仿真波形（第二段）

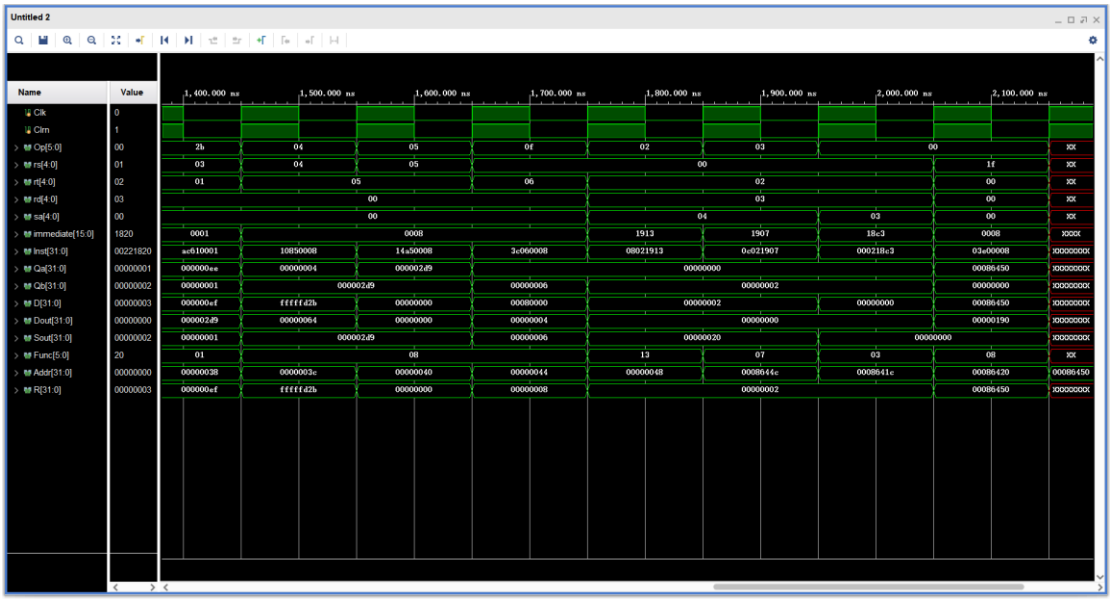

图 20 实际仿真波形（第三段）

图中 R 表示每条指令在 ALU 中计算的结果，D 表示每个时钟周期结束时写回寄存器堆的值。

（三）分析

以其中两条指令为例，展示 CPU 运算情况：

1. 运行刚开始时 PC 等于 0x00000000，在其 6:2 位为 0b00000，也即 0，故取出第 0 条指令 32'b000000_00001_00010_00011_00000_100000，用汇编语言表示即 add $s3,$s1,$s2，也即取出寄存器 s1、s2 的值$s1、$s2 做加法运算，并在时钟周期结束时将运算结果写入寄存器 s3 中。根据我们对寄存器的初始化情况，$s1=1,$s2=2，故将 3 写入寄存器 s3。

对照**实际仿真波形（第一段）**或**图 21 检验 1** 可以验证，Qa 输出了寄存器 s1 的值 1，Qb 输出了寄存器 s2 的值 2，在第一个时钟周期中 ALU 运算结果 R=0x00000003，写回 Regfile 的值 D=0x00000003，符合预期。

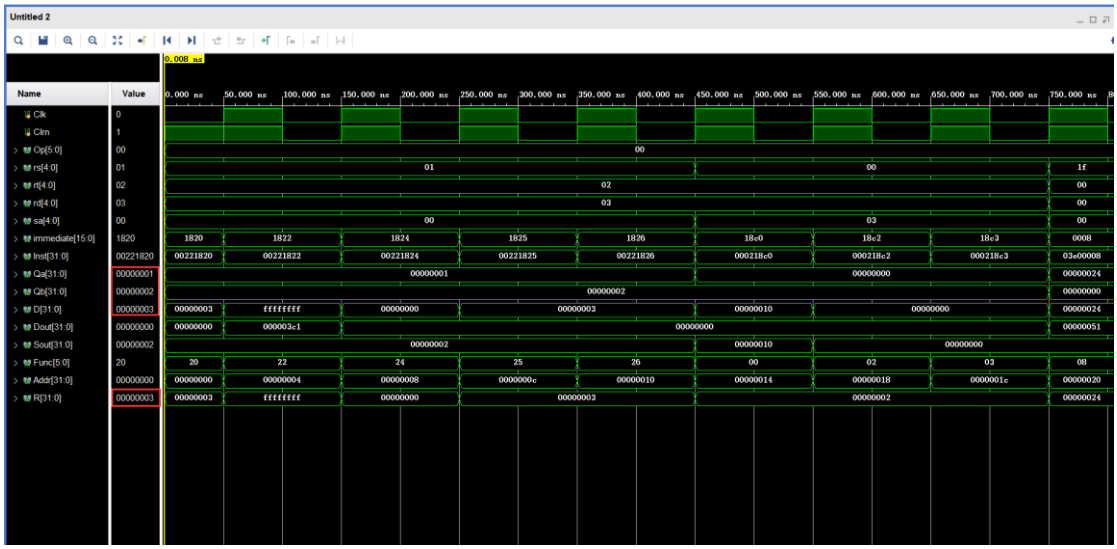

图 21 检验 1

2. 在运行到 PC 等于 0x00000036(也即 54 或 0b00…00110110 时)，在其 6:2 位为 0b01101，也即 13，故应该取出 Regfile 中第 13 条指令，根据初始化情况，也即 32'b100011_00011_00101_0000000000000001，用汇编语言表示即 lw $s5,1($s3)，其作用是将寄存器 s3 的值与立即数相加，得到的值作为地址在内存中寻找对应数据，并在该时钟周期结束时将数据存到 s5 中。经过上一条指令，$s3=$s1⊕0x00ef=1⊕0x00ef=0x000000ee ,s3 寄存器的值变成了 0x000000ee ，加上立即数 1 得到 0x000000ef ，其 6:2 位为 0b11011，也即 27，根据数据存储器的初始化情况，ram[27]的值为 27^2=729，也即 0x000002d9 ,故数据 0x000002d9 将在时钟周期结束时将运算结果写入寄存器 s5。

对照**实际仿真波形（第二段）**或**图 22 检验 2** 可以验证，在该指令的时钟周期中数据存储器的输出 Dout 为 0x000002d9，写回 Regfile 的值 D=0x000002d9，符合预期。

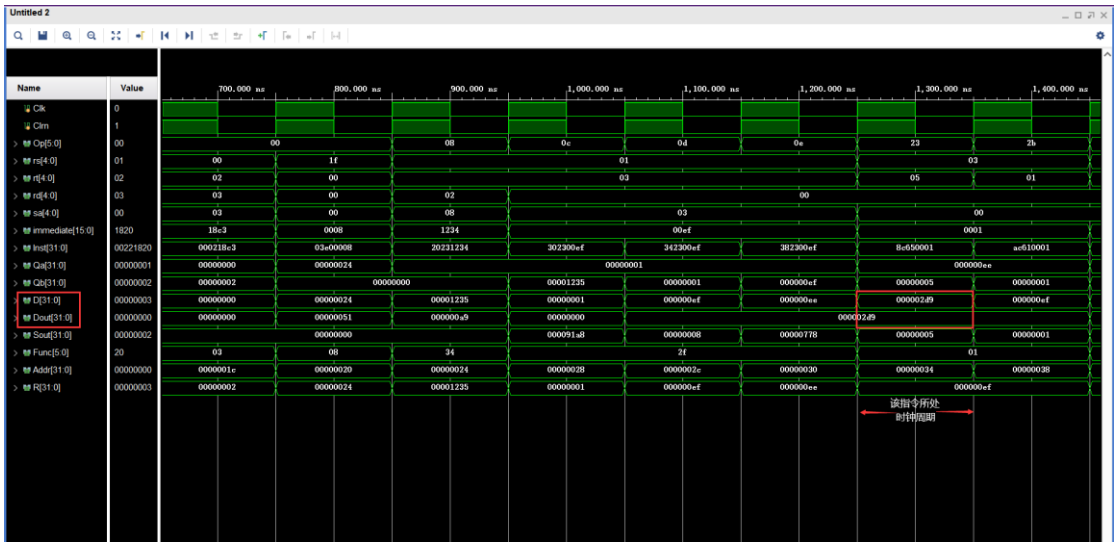

图 22 检验 2

（四）结论

按照如上检测方法，将测试结果对比预测结果，可以验证所有测试结果均符合预测，波形输出正常，单周期 CPU 设计成功。

五、实验结论与体会

（一）实验结论

成功完成单周期处理器的设计与实现，单周期 CPU 支持 20 条指令，所有波形输出正常，与预期结果一致，所以指令均完成测试。

（二）体会

在完成实验的过程中，我们对单周期 CPU 的运行原理和计算机相关知识更加清楚了。因为需要做实验，我们在看书时更加认真，对每一段文字都认真阅读，仔细推敲。我们认真研究每一条指令的结构和功能，理解每一个模块的设计原理，了解每个接口的作用，正是因为对书的认真阅读、理解，我们才可能成功完成 lui 等附加模块的设计。

在实验过程中也遇到很多问题，诸如 regfile、datamem 的初值如何设置，输出信号为 ZZZZZZZZ、XXXXXXXX 的区别，在小组讨论以及请教老师、助教后问题能够得到解决，这也使我们学到更多知识，了解了 CPU 设计中存在的很多需要注意的细节。

实验课与理论课不同，它更考验我们的动手能力、将理论知识运用实际的能力；它也训练了我们工程方面的意识：如何解决一个大的问题？如何为要完成的任务做出规划？如何达到有效的合作？如何解释理论和实际的差距？等等。在这门课中收获到的绝对不止知识，还有许多必要的能力。

最终成功搭出单周期 CPU 并运行得到正确的结果时是开心的，它不仅是对我们努力的肯定，也为我们完成下一个实验带来了很大信心。相信通过这门课能

学习到与计算机相关的很多有用的知识。