**组成原理实验课程第六次实报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | 单周期CPU实现 | | | 班级 | 李涛 |
| 学生姓名 | 张丛 | 学号 | 2113662 | 指导老师 | 董前琨 |
| 实验地点 | A区308 | | 实验时间 | 2023.06.07 | |

1. **实验目的**

1. 理解 MIPS 指令结构，理解 MIPS 指令集中常用指令的功能和编码，学会对这些

指令进行归纳分类。

2. 了解熟悉 MIPS 体系的处理器结构，如延迟槽，哈佛结构的概念。

3. 熟悉并掌握单周期 CPU 的原理和设计。

4. 进一步加强运用 verilog 语言进行电路设计的能力。

5. 为后续设计多周期 cpu 的实验打下基础。

1. **实验内容说明**

1. 做好预习：

1） 熟知 MIPS 指令类型，深入理解常用指令的功能和编码；

2） 归纳常用的 MIPS 指令，确定自己准备实现的 MIPS 指令；

3） 对准备实现的指令进行分析，完成表 7.1 的填写；

4） 设计本次实验的方案，画出实验方案的设计框图，即补充完善图 7.1；

5） 如果对 FPGA 板了解的话，可确定设计中与 FPGA 板上交互的接口，画出

包含外围模块的整体设计框图，即补充完善图 7.2；

6） 依据自己设计中实现的指令，编写一段不少于 20 行的汇编程序，要求包

含所有实现的指令，完成表 7.2 的填写。

2. 实验实施：

1） 确认单周期 CPU 的设计框图的正确性；

2） 编写 verilog 代码，将表 7.2 中自己编写的汇编程序翻译为二进制，内

嵌到指令 ROM 中；

3） 对该模块进行仿真，得出正确的波形，截图作为实验报告结果一项的材

料；

4） 完成调用单周期 CPU 的外围模块的设计，并编写代码；

5） 对代码进行综合布局布线下载到实验箱里 FPGA 板上，进行上板验证。

3. 实验检查：

1） 完成上板验证后，让指导老师或助教进行检查，进行现场演示。先解读

表 7.2 中自己编写的汇编程序，然后采用手动输入时钟，每个周期查看 CPU 状

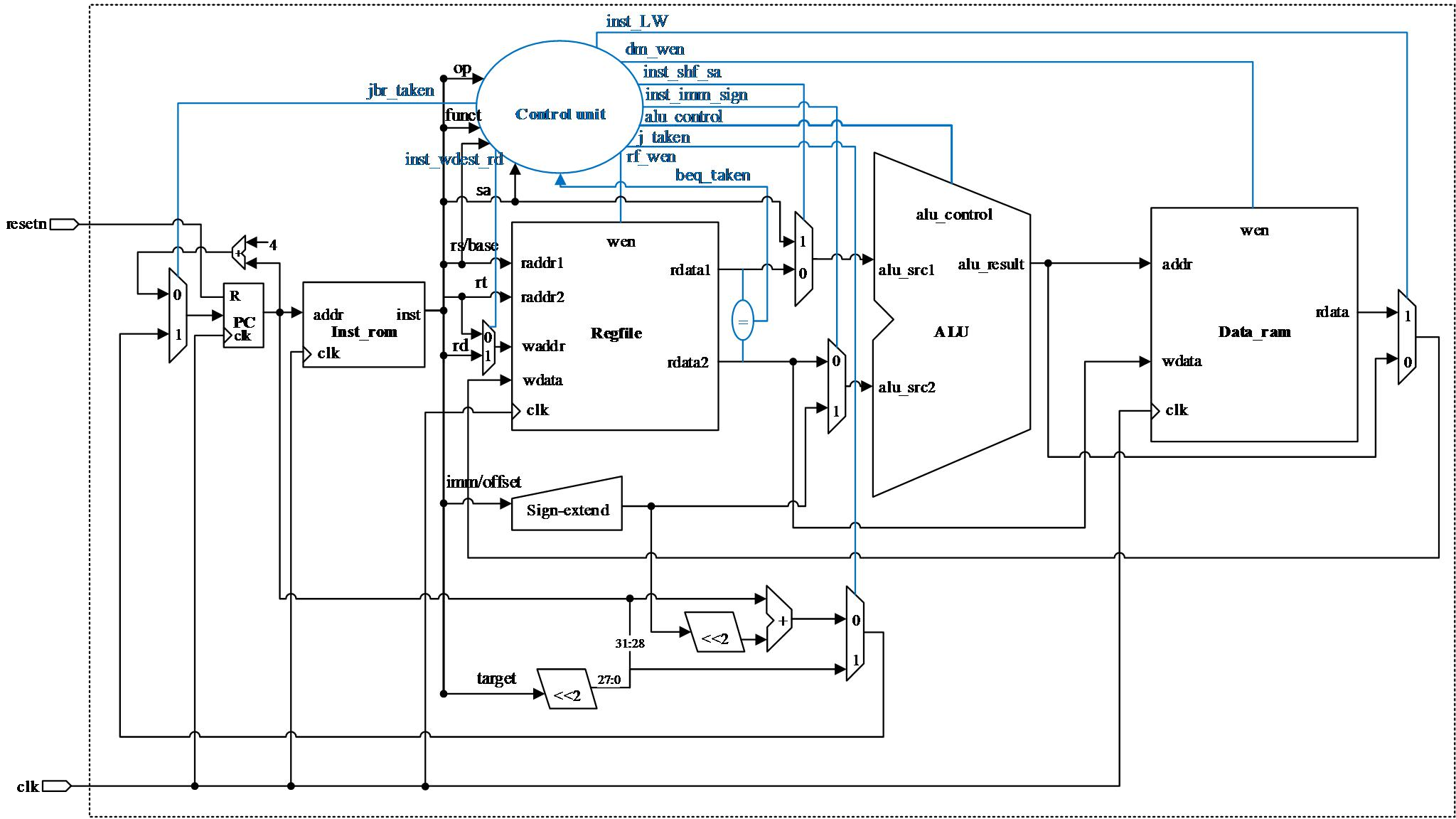
态，按照检查人员的要求进行演示，检查指令运行结果的正确性，可对演示结

果进行拍照作为实验报告结果一项的材料。

4. 实验报告的撰写：

1） 实验结束后，需按照规定的格式完成实验报告的撰写。

1. **实验原理图**



本实验的目的是实现指令的取指、译码、取数据、ALU运算、内存访存以及数据回写寄存器。通过将指令进行分类，并加入控制信号，可以正确执行每一个指令。本次实验中主要采用了R型和I型指令，使用不同的控制信号来区分这两种指令的执行过程。

1. **实验步骤**

1. 添加两个新指令：

- R型指令：SRT $10,$6,$7，op操作码为111011，指判断六号寄存器的值是否大于七号寄存器的值，若大于则置位为1；

- I型指令：LBI $13,#12，op操作码为111010，指将立即数12装载到13号寄存器。

2. 修改inst\_rom，使其存放22条指令。

- 将之前的j 00H指令放在最后；

- 将新加入的两条指令放在此条指令前面。

3. 修改控制信号与运算结果，定义并补充以下内容：

- srt\_result控制信号：用于判断操作数1是否大于操作数2；

- lbi\_result运算结果：将立即数12装载到13号寄存器中。

4. 针对两种新的指令，我们需要添加判断的控制信号以及两种运算结果。

5. 指令的rd和rt判断需要加上新补充指令的控制信号，如下所示：

- rt判断：inst\_ADDIU、inst\_LW、inst\_LUI、inst\_LBI、inst\_SRT。

- rd判断：inst\_ADDU、inst\_SUBU、inst\_SLT、inst\_AND、inst\_NOR、inst\_OR、inst\_XOR、inst\_SLL、inst\_SRL、inst\_SRT。

6. 具体实现：

- 大于则置位：将SRT指令中的操作数1和操作数2进行比较，若操作数1>操作数2，则将srt\_result的最低位设为1，否则设为0；

- 装载立即数到低位：将立即数12加载到13号寄存器的低16位（alu\_src2[15:0]）。

1. 具体virelog语言：

assign inst\_rom[ 0] = 32'h24010001; // 00H: addiu $1 ,$0,#1 | $1 = 0000\_0001H

assign inst\_rom[ 1] = 32'h00011100; // 04H: sll $2 ,$1,#4 | $2 = 0000\_0010H

assign inst\_rom[ 2] = 32'h00411821; // 08H: addu $3 ,$2,$1 | $3 = 0000\_0011H

assign inst\_rom[ 3] = 32'h00022082; // 0CH: srl $4 ,$2,#2 | $4 = 0000\_0004H

assign inst\_rom[ 4] = 32'h00642823; // 10H: subu $5 ,$3,$4 | $5 = 0000\_000DH

assign inst\_rom[ 5] = 32'hAC250013; // 14H: sw $5 ,#19($1) | Mem[0000\_0014H] = 0000\_000DH

assign inst\_rom[ 6] = 32'h00A23027; // 18H: nor $6 ,$5,$2 | $6 = FFFF\_FFE2H

assign inst\_rom[ 7] = 32'h00C33825; // 1CH: or $7 ,$6,$3 | $7 = FFFF\_FFF3H

assign inst\_rom[ 8] = 32'h00E64026; // 20H: xor $8 ,$7,$6 | $8 = 0000\_0011H

assign inst\_rom[ 9] = 32'hAC08001C; // 24H: sw $8 ,#28($0) | Mem[0000\_001CH] = 0000\_0011H

assign inst\_rom[10] = 32'h00C7482A; // 28H: slt $9 ,$6,$7 | $9 = 0000\_0001H

assign inst\_rom[11] = 32'h11210002; // 2CH: beq $9 ,$1,#2 | 跳转到指令34H

assign inst\_rom[12] = 32'h24010004; // 30H: addiu $1 ,$0,#4 | 不执行

assign inst\_rom[13] = 32'h8C2A0013; // 34H: lw $10,#19($1) | $10 = 0000\_000DH

assign inst\_rom[14] = 32'h15450003; // 38H: bne $10,$5,#3 | 不跳转

assign inst\_rom[15] = 32'h00415824; // 3CH: and $11,$2,$1 | $11 = 0000\_0000H

assign inst\_rom[16] = 32'hAC0B001C; // 40H: sw $11,#28($0) | Men[0000\_001CH] = 0000\_0000H

assign inst\_rom[17] = 32'hAC040010; // 44H: sw $4 ,#16($0) | Mem[0000\_0010H] = 0000\_0004H

assign inst\_rom[18] = 32'h3C0C000C; // 48H: lui $12,#12 | [R12] = 000C\_0000H

assign inst\_rom[19] = 32'hECC75000; // 50H 大于则置位

assign inst\_rom[20] = 32'hE80D000C; // 54H 装载立即数到低位

assign inst\_rom[21] = 32'h08000000; // 4CH: j 00H | 跳转指令00H

大于则置位具体实现：

assign srt\_result[31:1] = 31'd0;

assign srt\_result[0] = (adder\_result==32'b0)?1'd0:(~alu\_src1[31] & alu\_src2[31]) | (~(alu\_src1[31]^alu\_src2[31]) & ~adder\_result[31]);

装载立即数到低位：

assign lbi\_result = {16'd0,alu\_src2[15:0]};

写到rt还是rd的修改：

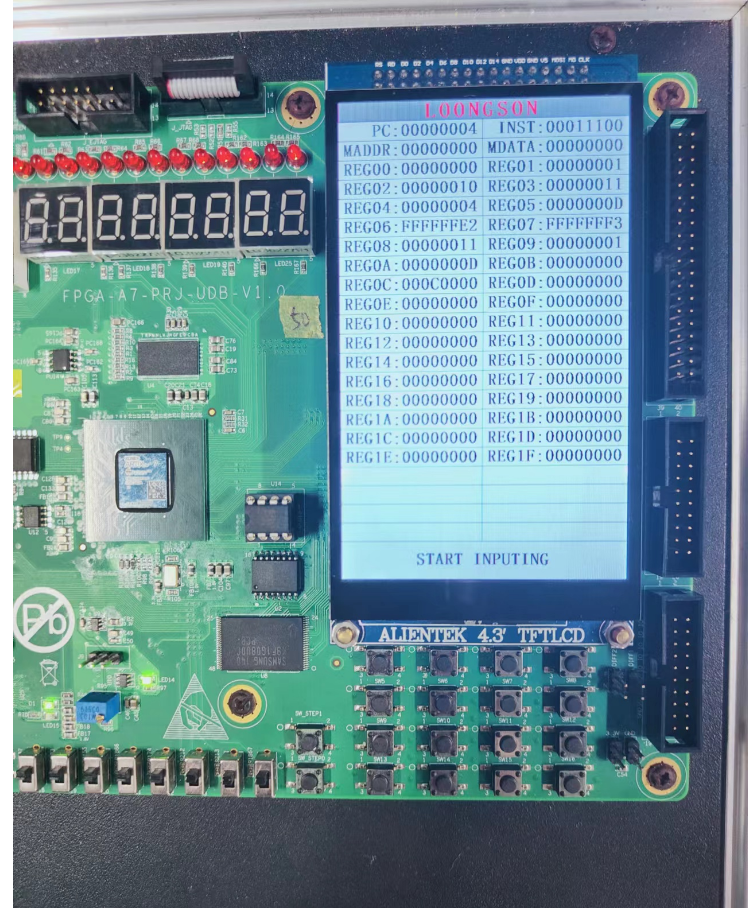
assign inst\_wdest\_rt = inst\_ADDIU | inst\_LW | inst\_LUI|inst\_LBI;

assign inst\_wdest\_rd = inst\_ADDU | inst\_SUBU | inst\_SLT | inst\_AND | inst\_NOR

| inst\_OR | inst\_XOR | inst\_SLL | inst\_SRL|inst\_SRT;

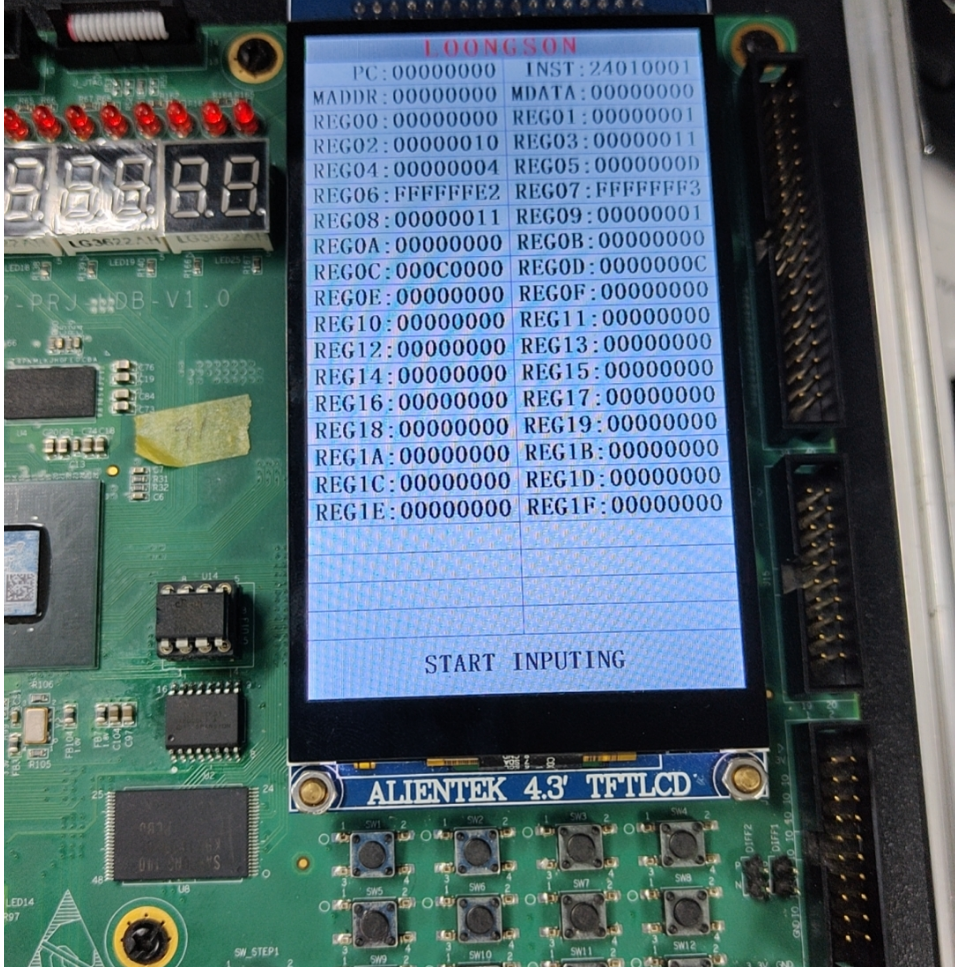
1. **实验结果分析**

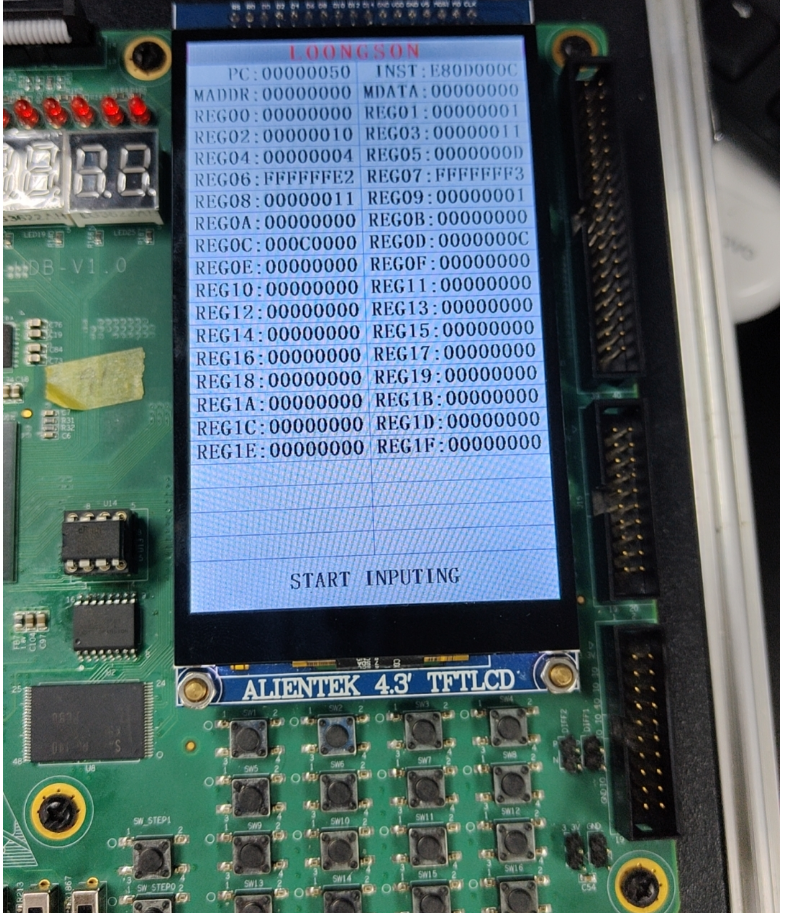
没有添加指令的实验箱执行完后的截图：



显然对比指令，我们可以看到每个寄存器上显示的数值正确。

添加指令后的实验箱拍照：





在执行完我们添加的所有指令后，我们观察到验证成功。因为6号寄存器小于7号寄存器的值，所以最后确实10号寄存器置位为0。并且13号寄存器装载了12到低位。

1. **总结感想**

通过这个实验，学习了CPU的基本结构和工作原理，以及如何使用Verilog语言实现CPU:

1. 熟悉了CPU的基本结构和工作原理.通过这个实验，深入了解了CPU的基本结构和工作原理。了解了CPU的指令集、控制器、运算器、存储器等组成部分。

2. 掌握了vVerilog语言的使用：在这个实验中，使用Verilog语言实现了CPU。这让我更加熟悉了Verilog语言的使用，包括如何编写模块、如何处理信号、如何调试等。

1. 学习了仿真和验证技术的应用：在这个实验中，使用仿真和验证技术来验证我的CPU实现的正确性。

4. 认识了CPU性能的局限与实际应用：在这个实验中，实现了单周期CPU，但是这种CPU在实际应用中性能较低，不能满足现代计算机应用的要求。认识到CPU设计需要考虑实际应用，不能只停留在理论层面。