# 第3章 单周期CPU系统设计实验

## 3.1 实验五 5条指令的单周期CPU系统设计实验

### 1 实验目的

1. **了解龙芯LoongArch指令系统架构，掌握常用指令的功能和编码。**
2. **熟悉并掌握5条指令的单周期CPU的原理和设计。**
3. **加强运用Verilog语言进行电路设计的能力。**

### 2 实验设备

主机：Lenovo小新Pro 13ITL 2021,Vivado 2019.2

龙芯CPU设计与体系结构教学实验系统：

* 基于FPGA芯片的嵌入式系统开发板：FPGA,Artix-7,xC7A200T-FBG676
* 开发板配套的电源适配器1个
* JTAG下载线

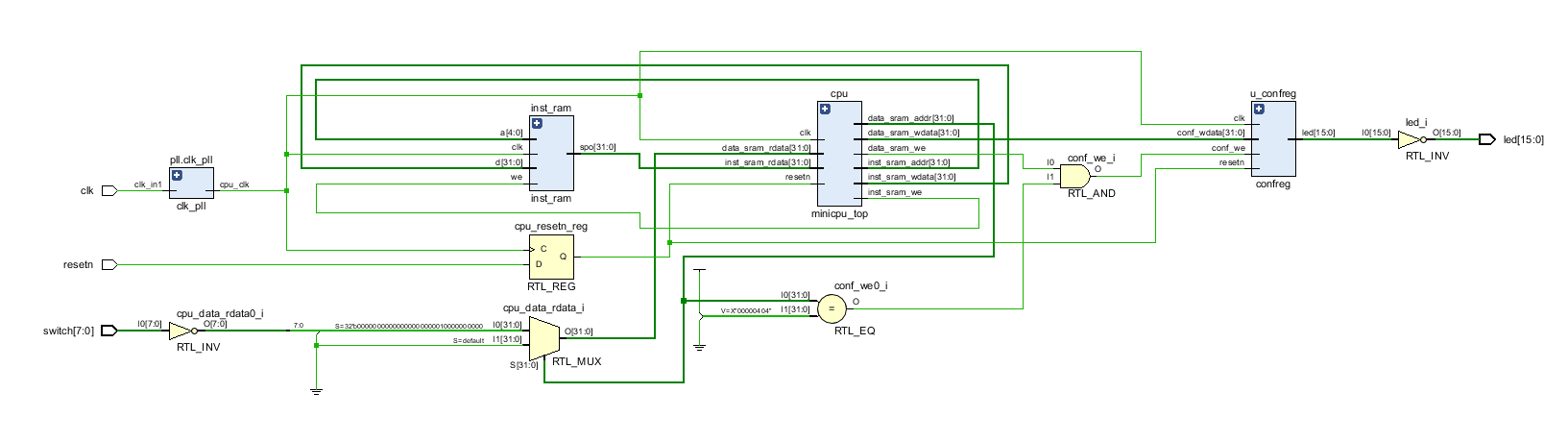
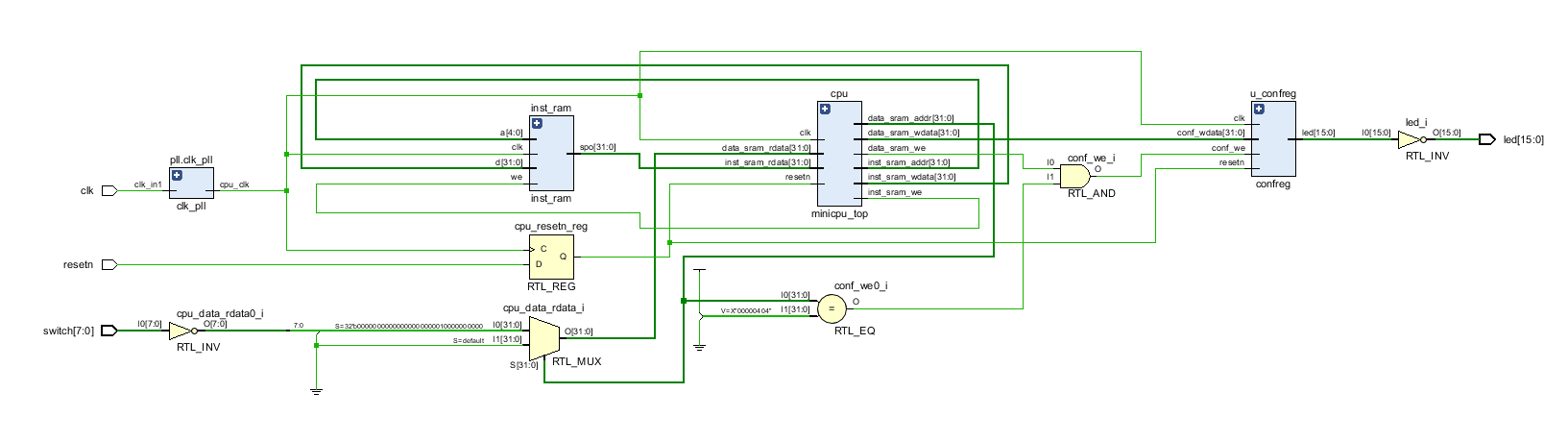
### 3 单周期CPU实现的5条LoongArch指令

表3.1 五条基础指令特性归纳表

|  |  |  |
| --- | --- | --- |
| 汇编指令 | 指令码 | 功能描述 |
| add.w rd, rj, rk | 00000000000100000|rk|rj|rd | tmp = GR[rj]+GR[rk]  GR[rd]= tmp[31:0]  将通用寄存器rj中的数据加上通用寄存器rk中的数据，将结果的[31:0]位写入通用寄存器rd中 |
| addi.w rd,rj,si12 | 0000001010|si12|rj|rd | tmp = GR[rj] + SignExtend(si12, 32)  GR[rd] = tmp[31:0]  将通用寄存器rj中的数据加上12比特立即数si12符号扩展后的32位数据，将结果写入通用寄存器rd中 |
| ld.w rd,rj,si12 | 0010100010|si12|rj|rd | vaddr = GR[rj] + SignExtend(si12, 32) AddressComplianceCheck(vaddr)  paddr = AddressTranslation(vaddr)  word = MemoryLoad(paddr, WORD)  GR[rd] = word  从内存取回一个字的数据写入通用寄存器rd |
| st.w rd,rj,si12 | 0010100110|si12|rj|rd | vaddr = GR[rj] + SignExtend(si12, 32) AddressComplianceCheck(vaddr)  paddr = AddressTranslation(vaddr) MemoryStore(GR[rd][31:0], paddr, WORD)  将通用寄存器rd的[31:0]位数写入到内存中 |
| bne rj,rd,offs | 010111|offs[15:0]|rj|rd | if GR[rj]!=GR[rd] :  PC = PC + SignExtend({offs16, 2'b0}, 32) 将通用寄存器rj和通用寄存器rd的值进行比较，如果两者不等则跳转到目标地址，否则不跳转 |

### 4 实验内容与步骤

1. **画出模块框图，标出输入输出端口。**

****

1. **主要设计代码及注释（填空处代码请标红）。**

always @(posedge clk) begin

if (reset) begin

pc <= 32'h1bfffffc; //trick: to make nextpc be 0x1c000000 during reset

end

else begin

pc <= nextpc;

end

end

assign inst\_sram\_we = 1'b0;

assign inst\_sram\_addr = pc;

assign inst\_sram\_wdata = 32'b0;

assign inst = inst\_sram\_rdata;

assign op\_31\_26 = inst[31:26];

assign op\_25\_22 = inst[25:22];

assign op\_21\_20 = inst[21:20];

assign op\_19\_15 = inst[19:15];

assign rd = inst[ 4: 0];

assign rj = inst[ 9: 5];

assign rk = inst[14:10];

assign i12 = inst[21:10];

assign i16 = inst[25:10];

decoder\_6\_64 u\_dec0(.in(op\_31\_26 ), .co(op\_31\_26\_d ));

decoder\_4\_16 u\_dec1(.in(op\_25\_22 ), .co(op\_25\_22\_d ));

decoder\_2\_4 u\_dec2(.in(op\_21\_20 ), .co(op\_21\_20\_d ));

decoder\_5\_32 u\_dec3(.in(op\_19\_15 ), .co(op\_19\_15\_d ));

assign inst\_add\_w = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'h0] & op\_21\_20\_d[2'h1] & op\_19\_15\_d[5'h00];

assign inst\_addi\_w = op\_31\_26\_d[6'h00] & op\_25\_22\_d[4'ha];

assign inst\_ld\_w = op\_31\_26\_d[6'h0a] & op\_25\_22\_d[4'h2];

assign inst\_st\_w = op\_31\_26\_d[6'h0a] & op\_25\_22\_d[4'h6];//在这里实现inst\_st\_w指令的代码//确定当前指令是否为存储字指令，仅在操作码和功能位同时满足特定条件时，inst\_st\_w才会为真。

assign inst\_bne = op\_31\_26\_d[6'h17];

assign src2\_is\_imm = inst\_addi\_w | inst\_ld\_w | inst\_st\_w;//在这里实现立即数选择信号//判断当前指令是否需要将第二个源操作数作为立即数，从而为后续的运算或存储提供正确的操作数信息。

assign res\_from\_mem = inst\_ld\_w;

assign gr\_we = inst\_add\_w | inst\_ld\_w | inst\_addi\_w;

assign mem\_we = inst\_st\_w;

assign src\_reg\_is\_rd = inst\_bne | inst\_st\_w;

assign rf\_raddr1 = rj;

assign rf\_raddr2 = src\_reg\_is\_rd ? rd :rk;

regfile u\_regfile(

.clk (clk ),

.raddr1 (rf\_raddr1),//

.rdata1 (rj\_value),

.raddr2 (rf\_raddr2),//

.rdata2 (rkd\_value),

.we (gr\_we ),

.waddr (rd ),//

.wdata (rf\_wdata )

);//在空出的括号里完成引脚匹配

assign br\_offs = {{14{i16[15]}},i16} << 2;//在这里完成br\_offs信号的生成 //偏移量需要先左移两位再与PC相加 //{{14{i16[15]}}, i16}：这是符号扩展，将i16的高位14位用i16[15]（最高位）填充，从而将16位立即数扩展为32位。这样可以正确处理负数偏移量。<< 2：左移两位，相当于乘以4，因为在RISC架构中，指令地址是按字节对齐的，左移两位可以将偏移量转换为字节地址。

assign br\_target = pc + br\_offs;

assign rj\_eq\_rd = (rj\_value == rkd\_value);

assign br\_taken = valid && inst\_bne && !rj\_eq\_rd;

assign nextpc = br\_taken ? br\_target : pc + 4;//在这里实现nextpc信号的生成 //如果br\_taken为真，nextpc将被赋值为br\_target，这是分支目标地址。如果br\_taken为假，nextpc将被赋值为当前程序计数器pc加4，表示顺序执行下一条指令（在RISC架构中，指令通常是4字节对齐的）。

assign imm = {{20{i12[11]}},i12[11:0]};

assign alu\_src1 = rj\_value;

assign alu\_src2 = src2\_is\_imm ? imm : rkd\_value;//在这里实现alu\_src2信号的生成 //如果需要立即数，就使用imm，否则使用寄存器值rkd\_value

这行代码用于生成alu\_src2信号，其含义如下：

assign alu\_result = alu\_src1+alu\_src2;

assign data\_sram\_we = mem\_we;

assign data\_sram\_addr = alu\_result;

assign data\_sram\_wdata = rkd\_value;

assign rf\_wdata = res\_from\_mem ? data\_sram\_rdata : alu\_result;//在这里完成写回寄存器值的选择

endmodule

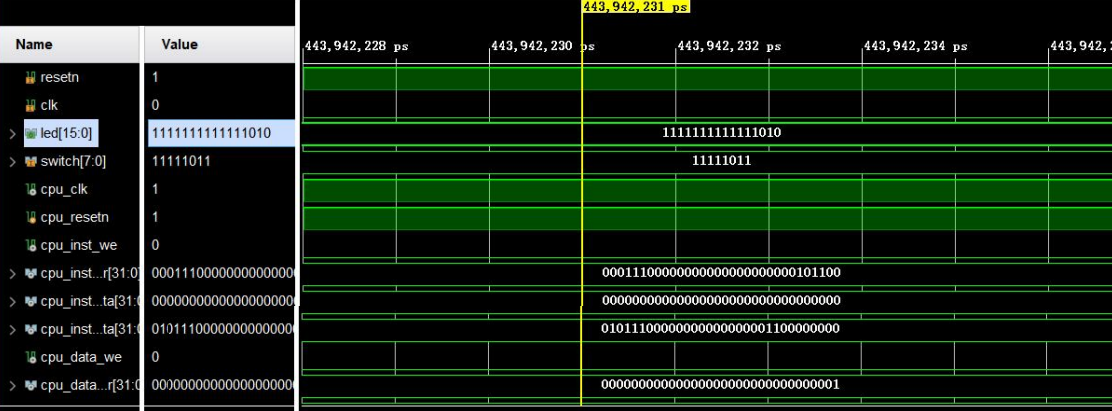
1. **测试所用汇编程序**

表3.2 五条指令单周期CPU测试所用汇编程序详述

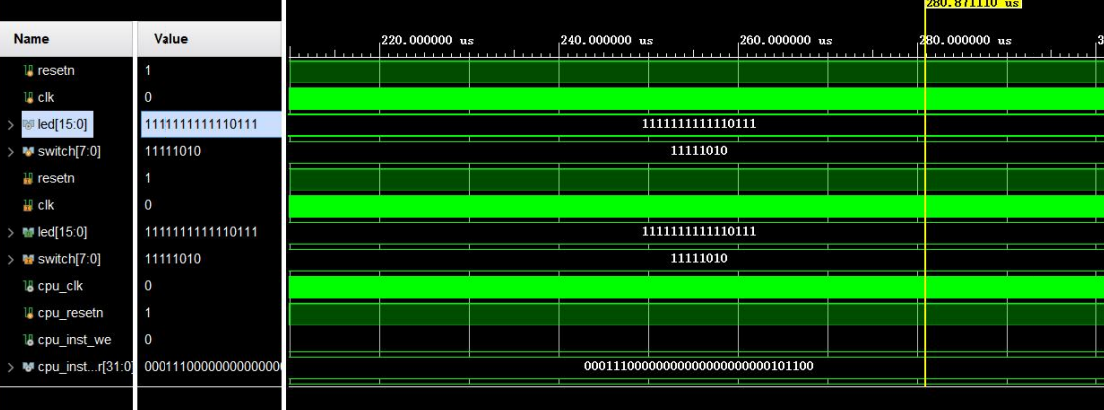
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令地址 | 汇编指令 | 结果描述 | 机器指令的机器码 | |
| 16进制 | 2进制 |
| 1c000000 | addi.w $t0,$zero,0x0 | [$t0] = 0x0 | 0280000c | 0000\_0010\_1000\_0000\_0000\_0000\_0000\_1100 |
| 1c000004 | addi.w $t1,$zero,0x1 | [$t1] = 0x1 | 0280040d | 0000\_0010\_1000\_0000\_ 0000\_0100\_0000\_1101 |
| 1c000008 | addi.w $s0,$zero,0x0 | [$s0]=0x0 | 02800017 | 0000\_0010\_1000\_0000\_ 0000\_0000\_0000\_0111 |
| 1c00000c | addi.w $s1,$zero,0x1 | [$s0]=0x1 | 02800418 | 0000\_0010\_1000\_0000\_ 0000\_0100\_0001\_1000 |
| 1c000010 | ld.w $a0,$zero,1024 | [$a0]=[$zero+1 024] | 28900004 | 0010\_1000\_0000\_0000\_ 0000\_0000\_0000\_0100 |
| 1c000014 | add.w $t2,$t1,$t1 | [$t2]=[$t1]+[$t 0] | 0010358e | 0000\_0000\_0001\_0000\_ 0011\_0101\_1000\_1110 |
| 1c000018 | addi.w $t0,$t1,0x0 | [$t0]=[$t1] | 028001ac | 0000\_0010\_1000\_0000\_  0000\_0001\_1010\_1100 |
| 1c00001c | addi.w $t1,$t2,0x0 | [$t1]=[$t2] | 028001cd | 0000\_0010\_1000\_0000\_ 0000\_0001\_1100\_1101 |
| 1c000020 | add.w $s0,s0,s1 | [s0]=[$s0|+[$s 1] | 001062f7 | 0000\_0000\_0001\_0000\_ 0110\_0010\_1111\_0111 |
| 1c000024 | bne $s0,$a0,loop | If([s0]!=[a0] loop | 5ffff2e4 | 0101\_1111\_1111\_1111\_ 1111\_0010\_1110\_0100 |
| 1c000028 | st.w $t2,$zero,1025 | 将 f(n)的值输 出到数码管上 | 2990100e | 0010\_1001\_1001\_0000\_ 0001\_0000\_0000\_1110 |
| 1c00002c | bne $s1,$zero,end | 测试完毕，进 入死循环 | 5c000300 | 0101\_1100\_0000\_0000\_ 0000\_0011\_0000\_0000 |

1. **仿真截图。**

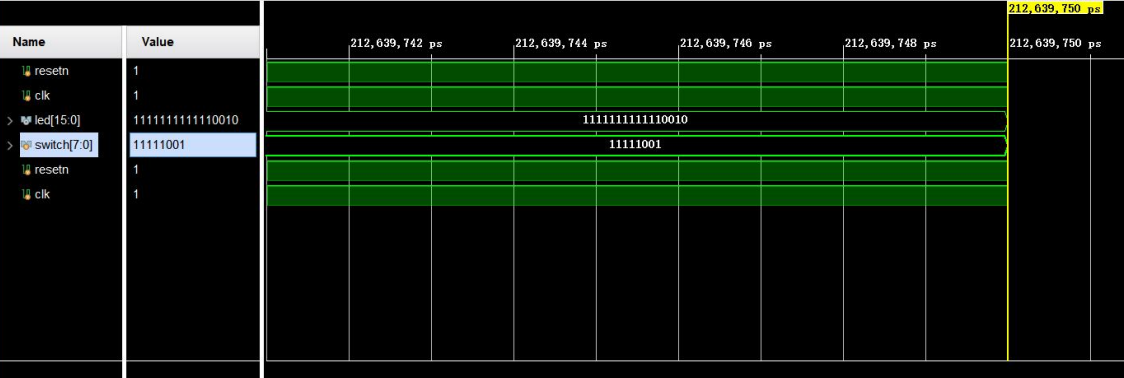
①switch = ~(8'h4);f(4)=5;led1111111111010 



②Case2:switch = ~(8'h5);f(5)=8;led1111111110111



③Case3:switch = ~(8'h6);f(6)=13;led1111111110010

****

1. **上板运行结果拍照。**

①右 2,3 开关上拨，即 n=6,f(6)=13，亮灯结果一致。 



②右 3 开关上拨，即 n=4,f(4)=5，亮灯结果一致。



### 5 实验数据

表3.3 5条指令单周期CPU实验数据

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 步骤 | 执行的指令 | 预期结果（pc、各寄存器等） | 实际结果（pc、各寄存器等） | 一致性 |
| 2035 | addi.w $t0,$zero,0x0 | pc = 1c000000  reset = 1  t0 = 0 | 12 号置 0  （实际上 12 号早就置零了， reset = 0 有效时，pc 恒置为 1c000000 cpu 持续地执行第 一行指令。） | 一致 |
| 2055 | addi.w $t1,$zero,0x1 | pc = 1c000004（这个信号 2035 - 2055ns 结束时才产 生效果，寄存器的写要等下 一个上升沿生效。） t1=1 | t1 = 1 | 一致 |
| 2075 | addi.w $s0,$zero,0x0 | pc = 1c000008  s0 = 0 | s0 =0 | 一致 |
| 2095 | addi.w $s1,$zero,0x1 | pc = 1c00000c  s1 = 1 | s1 = 1 | 一致 |
| 2115 | ld.w $a0,$zero,1024 | pc = 1c000010  a0 = 读到的数 | a0 = 4 | 一致 |
| 2135 | add.w $t2,$t0,$t1 | pc =1c000014  t2 = t1+t0 = 1 | t2 =1 | 一致 |
| 2155 | addi.w $t0,$t1,0x0 | pc = 1c000018  t0 = t1 = 1 | t0 = 1 | 一致 |
| 2175 | addi.w $t1,$t2,0x0 | pc = 1c00001c  t1 = t2 = 1 | t2 =1 | 一致 |
| 2195 | add.w $s0,$s0,$s | pc = 1c000020  s0++ | s0 = 1 | 一致 |
| 2215 | bne $s0,$a0,loop | pc = 1c000024  goto loop  下一刻 pc = 1c000014 |  | 一致 |
| 2235 |  |  | pc = 1c000014 | 一致 |
| 2315 | bne $s0,$a0,loop | pc = 1c000024  t0 =1  t1 = 2  t2 = 2  s0 = 2  goto loop  下一刻 pc = 1c000014 | t0 =1  t1 = 2  t2 = 2  s0 = 2 | 一致 |
| 2415 |  | pc = 1c000024  t0 =2  t1 = 3  t2 = 3  s0 = 3  goto loop  下一刻 pc = 1c000014 | t0 =2  t1 = 3  t2 = 3  s0 = 3 | 一致 |
| 2515 |  | pc = 1c000024  t0 =3  t1 = 5  t2 = 5  s0 =4  s0 = 4 = a0 不跳转  下一刻 pc = 1c000028 | t0 =3  t1 = 5  t2 = 5  s0 =4 | 一致 |
| 2535 | st.w $t2,$zero,1025 | pc = 1c000028 | pc = 1c000028 | 一致 |

### 6 实验小结（包括遇到的问题及解决办法）

问题：对查阅到的指令的具体含义感到困惑，无法准确理解它们在编程或架构中的作用。

解决方法：通过询问GPT来获得详细解释，同时进行资料搜索以获取准确的信息，我了解add.w表示加法操作、addi.w表示带立即数的加法、st.w表示存储字指令、ld.w表示加载字指令，以及bne表示不相等分支指令。这些指令在汇编语言中各有其独特的功能，通过合理运用这些知识，可以更好地理解指令的用途，并在编程中实现正确的测试结果。