实验六 CPU 运算器的设计与实现

山东大学网络空间安全学院

计算机组成与设计 实验报告

21 级 唐诗 202100460120

21 级 于洁 202100460119

21 级 李昕 202100460065

2022 年 11 月 24日

目录

一、实验任务.....2
二、实验说明.....2
二、 实验环境.....3
三、 思路分析.....3
四、 设计数码管.....3
五、实验实现.....4
六、完整代码.....10
七、实验结果展示.....17

一、实验任务

本次实验需要实现一个 CPU 的运算器。简易运算器由三个 8 位寄存器（R0、R1、R2）和一个算术逻辑单元(ALU)构成，其中 ALU 应该至少支持加法、减法、乘法，与、或、或非、与非、比较的功能。

输入由开关控制；每一步运算后，相应标志位（标志位设置同实验四）的情况通过 LED 灯表示；运算结果以十进制通过数码管显示。读取数据的结果以十进制通过数码管显示。时钟信号、复位信号等控制信号允许用开关控制。

注：1. 本次实验涉及的数据皆为补码。

2.实验说明给出的默认指令集由定长指令构成，其中指令的操作码为变长操作码。

二、实验说明

实验使用八位二进制串 ($b_7b_6b_5b_4b_3b_2b_1b_0$) 表示指令，与开发板上的八个开关对应。实验使用四位二进制补码作为输入，与开发板上的四个开关对应。三个寄存器 R0，R1，R2 分别对应二进制地址码 00、01、10。

运算指令格式 高四位($b_7b_6b_5b_4$)为操作码，次低两位（b_3b_2）为地址码（指明存放第一个操作数的寄存器），最低两位(b_1b_0)为地址码（指明存放第二个操作数的寄存器）。

注：运算结果始终默认存放至寄存器 R2 因此不在指令中显式指出。

有符号乘法的结果若超过 8 比特应当解释为溢出。操作码对应的运算操作由表 1 说明其中 A 表示第一个操作数，B 表示第二个操作数。

例：指令 0010-0001 表示计算寄存器 R0 的值加上寄存器 R1 的值。指令 0110-0110 表示计算寄存器 R1 的值减去寄存器 R2 的值。

表 1 运算指令的操作码说明

运算指令的操作码（b7b6b5b4）	运算操作
0000	A&B
0001	A|B
0010	A+B
0110	A-B
0111	有符号数比较
1100	~(A|B)
1101	~(A&B)
1110	A*B

存储指令格式 最高两位（b_7b_6）为操作码置 10，次高两位（b_5b_4）为写入寄存器的地址码，低四位（$b_3b_2b_1b_0$）为待写入数据的二进制补码。

读取指令格式 最高六位（$b_7b_6b_5b_4b_3b_2$）为操作码置 111100，最低两位（b_1b_0）为地址码置 10。要求该指令可读取寄存器 R2 的值并以十进制通过数码管显示。

注：不强制要求支持读取并显示 R0，R1 寄存器的数据。

二、 实验环境

Vivado 2019.1 开发工具 (xc7a100tcsg324 -1)

NEXYS A7 开发板

三、 思路分析

本实验的输入指令有三种，分别是运算指令、存储指令和读取指令。

在CPU模块中，首先判断是否是存储指令，如果最高两位操作码为10，则是存储指令。

如果是存储指令，则执行case语句，通过判断指令的次高两位，判断数据要放入的寄存器号，并对四位输入进行符号扩展，使用op[3]对高四位进行填充，并使用非阻塞赋值将其赋值给对应的寄存器。

如果不是存储指令，则判断高四位op[7:4]是否为对应的运算指令或读取指令。如果条件判断为真，执行相应运算。如果条件判断为假，指令最高六位操作码为111100，则执行读取指令。进行ALU实例化，计算出算术指令要放入寄存器中的数据值。分析标志位，通过分类讨论，将不可能置1的情况（包括存储指令、读取指令等）置0。在计算模块中，我们调用了八位加法器、有符号乘法等模块。

最后，通过连接数码管以十进制形式输出数据。

四、 设计数码管

1.flag的设计

我们用abs表示读入的数n的绝对值，先把n的符号位赋给flag,再复制八份，若n为0，不操作，若n为1，通过n与全1串异或按位取反，再加上flag的最低位，abs为求得的n的补码，所以n的绝对值的计算、赋值代码为

```
assign abs = (n^flag)+flag[0]
```

2.灯管的切换

将设定时钟计数上限为CNT_MAX，用100MHz时钟规定从0到 99999计数即为1ms，达到最大值时利用判断语句将其置0，否则不断累加。

我们在右侧LED小灯和LED数码管中分别显示二进制和十进制，便于通过二进制验证输出结果。不读数时，data_out赋0，数码管不显示。只有读取R2中数据时，才将alu中标志位数据传给XF，通过左侧的led灯显示结果，否则保持XF为0。

代码如下：

```
        if (b[8] && b[7] && b[6] && b[5] && b[4] && !b[3] && !b[2]) begin
            addr = b[1:0];
            data_out = mymem[addr];
            if (addr == 2) XF = 1t;
            else XF = 0;
        end
        else begin
            data_out = 0; XF = 0;
        end
```

五、实验实现

1. ALU模块：

ALU接口定义：

```
module alu(
    input[3:0] operator,
    input[7:0] A,
    input[7:0] B,
    output[7:0]r,//运算结果
    output ZF,//0 标志位，运算结果为 0(全零)则置 1，否则置 0
           CF,//进借位标志位
           OF,//溢出标志位
           SF, //符号标志位，与 r 的最高位相同
           PF  //奇偶校验位，若为r有奇数个1，置1，否则置0
);
```

（1）各种标志位的分析：（ZF、OF、CF、SF、PF）

（1）ZF：0 标志位，运算结果为 0(全零)则置 1，只要有一位不是1则置 0

```
assign ZF = (r == 0);
```

（2）SF： 符号标志位，与 r 的最高位相同

```
assign SF = OF ^ r[7];
```

（3）CF： 进借位标志位.，观察结果最高位，如果最高位为 0 说明没有发生进借位，CF置0。 只要有一个高位进位或者符号位进位就为溢出，并对加法和减法分别设置。
在加法中，如果是无符号运算，若向最高位进位，那么CF=1；如果是有符号运算，若从最高位借位，则将第二个操作数取反后，应该对高位没有进位。

```
adder f0(A[7], C[7], 0, ccf, cf);
assign CCF = ccf | ((cf == 1) & (r[7] == 0));
assign CF = (CCF^(operator[3:1]==3'b001))&f;
```

（4）PF：奇偶校验位，对r的每一位执行异或操作，若为r有奇数个1，置1，否则置0F。

```
assign PF = r[7]^r[6]^r[5]^r[4]^r[3]^r[2]^r[1]^r[0];
```

（5）OF：八位全加器或八位乘法器调用时的“溢出变量”of赋给OF，OF=of。
溢出判断： ①加法：正数+正数=负数；负数+负数=正数。
②减法：正数-负数=负数；负数-正数=正数。则判断为溢出
加法时，先判断a,b符号是否相同，结果赋给x1,再判断a,s符号是否不同，把结果赋给x2，x1与x2相与判断是否溢出，若溢出，overflow置为1。

```
assign x1 = ~(a[7] ^ b[7]); //判断a,b的符号是否相同
assign x2 = a[7] ^ s[7];    //判断a,s的符号是否不同
assign overflow = x1 & x2;  //利用a,b,s的符号判溢出
```

我们设计的ALU模块可以对八位以内的操作数A、B进行各种运算。

（2）加法器和乘法器：

在一位加法器模块基础上，将8个全加器串联，将本位两个一位二进制数（ia,ib）和来自低位的进位(cin)相加。输出向高位的进位(cout),输出本位和(sum)，构建八位全加器。

以此加法器为基础，构建八位有符号乘法器：

首先，判断a、b的符号是否相同，输入包含两个八位有符号乘数a、b，以及符号位sign（sign=a[7]^b[7]）,若相同则不做处理，若不同则对无符号乘法的结果取补码。输出运算结果s和溢出位of。前面已经在ALU中对负操作数取补码，记录符号后，下面按照无符号乘法器进行运算，再依据符号位考虑是否将低八位取反作为运算结果s

（3）ALU部分的具体实现：

这一部分，我们采用了双层嵌套语句。operator=0010时，进行加法操作，正常调用八位全加器，b=B,使得r=A+b,0001时），实现加法功能。

减法时，先对b(0010时b=B；0011时b=1)取补码，再调用八位全加器，使得r=a+b。

case部分表示如下。

```
        always @(*)begin
          case (operator)
              4'b0010:  //A+B
                  begin
                      C=B;
                      m1=0;
                        m2=0;
                  end
              4'b0110: //A-B
                  begin
                      C=B2;
                      m1=0;
                      m2=0;
                  end
              4'b1110:  //A*B
                  begin
                      C=0;
                      if(A[7]==1'b1)m1=A2;
                      else m1=A;
                      if(B[7]==1'b1)m2=B2;
                      else m2=B;
                  end
              4'b0111:  //小于置位
                  begin
                      C=B2;
                      m1=0;
                      m2=0;
                  end
          endcase
      end
```

其中，对于乘法以外的操作，m1、m2只需要置零。对于加减操作，A、C作为加法器操作数，A不进行操作，对于B和C，根据操作数，C为B本身或B的补码或1或1的补码。对于乘法操作，C置零，m1、m2是乘法器操作数，根据A、B，m1、m2分别传入A、B或其补码。

```
        adder_8bit(A,C,0,C1,of1);
        mult(m1,m2,A[7]^B[7],C2,of2);
```

当操作码operator=0010或operator=0110，调用加法器，将运算结果C1存入out；

当操作码operator=1110，将运算结果C2存入out；

其他情况将在后面详细介绍。

通过以上各种情况得来的out最终赋给输出r。下面就是这一部分的代码，其中f是用于后续CF判断的标记。

```
          always @(*)begin  //数据预处理
           if(operator==4'b0010||operator==4'b0110)begin  //判断是否为加法器
               out=C1;
               of=of1;
               f=1;
           end
           else if(operator==4'b1110)begin  //判断是否为乘法
               out=C2;
              of=of2;
               f=1;
           end
           else if(operator==4'b0000)begin  //A&B
              out=A&B;
               of=0;
           end
           else if(operator==4'b0001)begin  //A|B
               out=A|B;
               of=0;
               f=0;
           end
           else if(operator==4'b1100)begin   //~(A|B)
               out=~(A|B);
               of=0;
               f=0;
             end
  else if(operator==4'b0111)begin   //小于置位
               out=C1[7];
               of=0;
               f=0;
           end
   else if(operator==4'b1101)begin   //~(A & B)
                       out=~(A & B);
               of=0;
               f=0;
           end
       end

       assign r=out;
       assign CCF = ccf | ((cf == 1) & (r[7] == 0));
       assign OF=of;
       assign ZF = (r == 0);
      assign SF = OF^r[7];
       assign PF = r[7]^r[6]^r[5]^r[4]^r[3]^r[2]^r[1]^r[0];
       assign CF = (CCF^(operator[3:1]==3'b001))&f;
         endmodule
```

2. CPU模块：

代码如下：

```
CPU接口定义如下：
 module CPU(
     input clk, // 输入时钟
     input [9:0] b, // 10位指令
     output reg [7:0] data_out, //  输出看一下结果
```

```
        output reg [4:0] XF,
        output reg [7:0] led_id,
        output reg [6:0] out_led
    );
        reg [7:0] mymem [2:0]; // 定义 32 个 8 位存储器（mymem[i]表示第 i 个存储器）
        wire [7:0] zc;
          reg [4:0] lt;
        reg [1:0] addr;
        wire ZF,CF,OF,SF,PF;
        wire [3:0] code;
        assign code = b[7:4];
  endmodule
```

由于指令读取和ALU运算同时进行，会造成了“数据竞争”。所以，我们构建了一个十位的二进制变量b,额外加入了两位保护位，加入两位额外的操作码当b[8]=1时，指令有效并执行，否则无论后续指令为何都不进行任何操作。b[9]是一个时钟信号，当b[9]的上升沿到来时，R2寄存器才被写入。此外，多出一位R2寄存器的写控制信号在CPU进行指令扩展或以此为底层实例化其他模块时，更加方便。

```
        always@ (posedge b[9]) begin
            if (b[8] && !b[7] || b[8] && b[7] && b[6] && !b[5] && !b[4])
                mymem[2] = zc;
                lt = {ZF,CF,OF,SF,PF};
        end
```

其次，我们构建了一个八位的二进制变量data_out，在右侧八个小灯上显示二进制位，便于更方便地显示具体的二进制表示。另外我们构建了八位的led_id，七位的out_led还有clk。

我们借助内置的E3时钟信号来存储和读取数据。

当b[8:6]=3'b110时，我们写入数据，对b[3:0]先进行符号扩展为八位，之后存入地址码为addr=b[5:4]的寄存器mymem[addr]中；

```
            if (b[8] && b[7] && !b[6]) begin                //10存储操作
                if (b[5:4]) mymem[1] = {b[3],b[3],b[3],b[3],b[3:0]};
                if (!b[5:4]) mymem[0] = {b[3],b[3],b[3],b[3],b[3:0]};
            end
```

当b[8:2]=7'b1111100时，我们读出数据,将地址码为addr=b[1:0]的寄存器 mymem[addr]中存的数据传给输出data_out。

通过ALU模块进行运算。两个操作数分别是mymem[b[3:2]]和mymem[b[1:0]]，把运算结果存入一个暂存寄存器zc中。

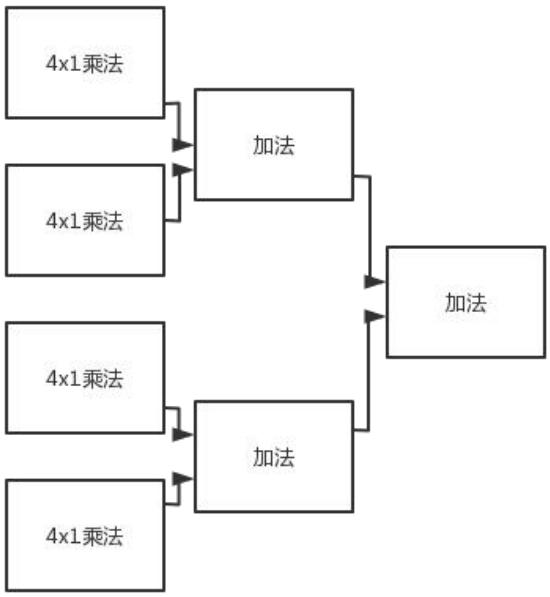
```
        wire [1:0] A,B;
          assign A = b[3:2];
        assign B = b[1:0];
        alu(code,mymem[A],mymem[B],zc,ZF,CF,OF,SF,PF);
```

我们还对十进制输出结果进行了优化，“+”不显示，在选择灯的部分，只有输入读取指令时，数码管才显示数据，否则不显示。在时钟信号下，四个数码管以肉眼不可分辨的速度按顺序分别显示负号和三位十进制数。

3. 其他设计

除了上述在ALU和CPU模块提到的优化，我们还进行了以下优化：

流水线操作：改进后的代码利用两个四位加法器进行运算，这样循环两次，分别计算两个数前四位的和以及后四位的和，提升乘法器性能。




```
//八位有符号乘法器：
    module mult(
        input[7:0] a,
        input[7:0] b,
        input sign,
        output[7:0] s,
        output of
        );
        reg[15:0] tmp,sum1;
        reg [15:0] mux0,mux1,mux2,mux3,mux4,mux5,mux6;
        wire [15:0] sum,cout1,cout2,cout3,cout4,cout5,sum2;
        always@(*)begin
            tmp={{8{1'b0}}, b};
            if(a[0])mux0=tmp; else mux0=16'd0; // 若 a 的当前位为 1 则运算
            tmp=tmp<<1;
            if(a[1])mux1=tmp; else mux1=16'd0;
            tmp=tmp<<1;
            if(a[2])mux2=tmp; else mux2=16'd0;
            tmp=tmp<<1;
            if(a[3])mux3=tmp; else mux3=16'd0;
            tmp=tmp<<1;
            if(a[4])mux4=tmp; else mux4=16'd0;
            tmp=tmp<<1;
            if(a[5])mux5=tmp; else mux5=16'd0;
            tmp=tmp<<1;
            if(a[6])mux6=tmp; else mux6=16'd0;
        end
        adder_16bit f0(mux0,mux1,0,cout1,co1);
        adder_16bit f1(cout1,mux2,0,cout2,co2);
        adder_16bit f2(cout2,mux3,0,cout3,co3);
        adder_16bit f3(cout3,mux4,0,cout4,co4);
        adder_16bit f4(cout4,mux5,0,cout5,co5);
        adder_16bit f5(cout5,mux6,0,sum,co6);
```

```
      always@(*)begin
          if(sign==1)sum1=(~sum)+1;
            else sum1=sum;
      end
      assign sum2=sum1;
      assign of=(sum2[15:7]!=9'b000000000)&&(sum2[15:7]!=9'b111111111);
        assign s=sum2[7:0];

  endmodule
```

（2）存储器扩充

另外，我们通过增加输入指令的长度，扩充了存储器。指令在存储器中顺序存储，执行时可用立即数寻址或寄存器寻址来确定目标指令。存储器前半部分存数据，后半部分存储指令。我们增设ram，rom，增加了input use[4:0]。use[4]表示是否启用存储器，use[3]表示是 用立即数寻址还是寄存器寻址。use[2]表示使用ram还是rom，use[1:0]表示存储指令，存储修改数据，访问数据，执行指令。

```
reg [7:0] rom[4:0],ram[4:0],D;
reg vrom[4:0];  //判断当前rom是否已使用过
reg [7:0]temp;
reg [7:0] address1,address2;  //存储指令的地址
reg [3:0] address2;  //所需要显示或执行的地址
reg F;
assign code = b[7:4];
always@ (posedge D[9]) begin
    if (D[8:7]==2'b10|| D[8:4]==5'b11100 )
        mymem[2] = zc;
        lt = {ZF,CF,OF,SF,PF};
end
always@ (posedge clk) begin
    D=b;
    address2=b[7:4];
    if(!use[4]||F) begin
    if (D[8:2]==7'b1111100) begin
            addr = D[1:0];
            data_out = mymem[addr];
            if (addr == 2) XF = lt;
            else XF = 0;
        end
        if (D[8:6]==3'b110) begin
            if (D[5:4]) mymem[1] = {D[3],D[3],D[3],D[3],D[3:0]};
            if (!D[5:4]) mymem[0] = {D[3],D[3],D[3],D[3],D[3:0]};
        end
    if(use[4])begin
        if(use[3]==1)address2=mymem[b[7:6]];
        case(use[2:0])
            4'b000: begin //存指令
               ram[address1|5'b10000]=b[7:0];
               temp=1;
               while((address1&temp)!=0)begin
                   address1=address1^temp;
                   temp=temp<<1;
               end
               address1=address1|temp;
               if(address1==5'b11110)address1=0;
           end
```

```
            4'b001: ram[address2]=b[3:0];
            4'b010:data_out=ram[address2];
            4'b011: begin
          D[7:0]=ram[address2|5'b10000];
          F=1;
            end

             4'b100: begin
                 if(address2!=4'b1111)begin
                     rom[address2|5'b10000]=b[7:0];
                     temp=1;
                     while((address2&temp)!=0)begin
                         address2=address2^temp;
                         temp=temp<<1;
                     end
                    address2=address2|temp;
                 end
             end

             4'b101: begin
                 if(vrom[address2]==0)begin
                     rom[address2]=b[3:0];
                     vrom[address2]==1;
                 end
             end
             4'b110: data_out=rom[address2];
             4'b111: begin
                 D[7:0]=rom[address2|5'b10000];
                 F=1;
             end
       else begin
            data_out = 0; XF = 0;
           end
    end
           endcase
    end
end
```

六、完整代码

```
 module adder(
    input ia, //1位二进制加数
    input ib,//1 位二进制加数
    input cin,//低位来的进位信号
    output cout,//向高位的进位信号
    output sum//1 位和数
    );
wire   x1;
wire   x2;
wire   x3;
assign x1 = ia ^ ib;
assign x2 = cin & x1;
assign x3 = ia & ib;
assign sum = x1 ^ cin;
```

```
  assign cout = x2 | x3;
  endmodule
//八位全加器：（有符号）
  module adder_8bit(
  input[7:0] a,
  input[7:0] b,
  input cin,
  output[7:0] s,
  output overflow
        );
        wire x1, x2;
  adder f0(a[0], b[0], cin, co1, s[0]);
  adder f1(a[1], b[1], co1, co2, s[1]);
  adder f2(a[2], b[2], co2, co3, s[2]);
  adder f3(a[3], b[3], co3, co4, s[3]);
  adder f4(a[4], b[4], co4, co5, s[4]);
  adder f5(a[5], b[5], co5, co6, s[5]);
  adder f6(a[6], b[6], co6, co7, s[6]);
  adder f7(a[7], b[7], co7, co8, s[7]);
  assign x1 = ~(a[7] ^ b[7]); //判断a,b的符号是否相同
  assign x2 = a[7] ^ s[7];    //判断a,s的符号是否不同
  assign overflow = x1 & x2;  //利用a,b,s的符号判溢出
  endmodule
//八位加法器：（无符号）
  module unsigned_adder_8bit(
      input[7:0] a,
      input[7:0] b,
      input cin,
      output[7:0] s,
      output overflow
      );
      adder f0(a[0], b[0], cin, co1, s[0]);
               adder f1(a[1], b[1], co1, co2, s[1]);
      adder f2(a[2], b[2], co2, co3, s[2]);
      adder f3(a[3], b[3], co3, co4, s[3]);
      adder f4(a[4], b[4], co4, co5, s[4]);
      adder f5(a[5], b[5], co5, co6, s[5]);
      adder f6(a[6], b[6], co6, co7, s[6]);
      adder f7(a[7], b[7], co7, overflow, s[7]);
    endmodule
//十六位全加器：
  module adder_16bit(
      input[15:0] a,
      input[15:0] b,
      input cin,
      output[15:0] s,
      output overflow
      );
      wire co;
      unsigned_adder_8bit f1(a[7:0], b[7:0], 0, s[7:0], co);
      unsigned_adder_8bit f2(a[15:8], b[15:8], co, s[15:8], overflow);
   endmodule
//八位有符号乘法器：
  module mult(
      input[7:0] a,
      input[7:0] b,
```

```
    input sign,
    output[7:0] s,
    output of
    );
    reg[15:0] tmp,sum1;
    reg [15:0] mux0,mux1,mux2,mux3,mux4,mux5,mux6;
    wire [15:0] sum,cout1,cout2,cout3,cout4,cout5,sum2;
    always@(*)begin
        tmp={{8{1'b0}}, b};
        if(a[0])mux0=tmp; else mux0=16'd0; // 若 a 的当前位为 1 则运算
        tmp=tmp<<1;
        if(a[1])mux1=tmp; else mux1=16'd0;
        tmp=tmp<<1;
        if(a[2])mux2=tmp; else mux2=16'd0;
        tmp=tmp<<1;
        if(a[3])mux3=tmp; else mux3=16'd0;
        tmp=tmp<<1;
        if(a[4])mux4=tmp; else mux4=16'd0;
        tmp=tmp<<1;
        if(a[5])mux5=tmp; else mux5=16'd0;
        tmp=tmp<<1;
        if(a[6])mux6=tmp; else mux6=16'd0;
    end
    adder_16bit f0(mux0,mux1,0,cout1,co1);
    adder_16bit f1(cout1,mux2,0,cout2,co2);
    adder_16bit f2(cout2,mux3,0,cout3,co3);
    adder_16bit f3(cout3,mux4,0,cout4,co4);
    adder_16bit f4(cout4,mux5,0,cout5,co5);
    adder_16bit f5(cout5,mux6,0,sum,co6);
    always@(*)begin
        if(sign==1)sum1=(~sum)+1;
          else sum1=sum;
    end
    assign sum2=sum1;
    assign of=(sum2[15:7]!=9'b000000000)&&(sum2[15:7]!=9'b111111111);
      assign s=sum2[7:0];

endmodule
//ALU:
module alu(
    input[3:0] operator,
    input[7:0] A,
    input[7:0] B,
    output[7:0]r,//运算结果
    output ZF,//0 标志位，运算结果为 0(全零)则置 1，否则置 0
            CF,//进借位标志位
            OF,//溢出标志位
            SF, //符号标志位，与 r 的最高位相同
            PF
);
    reg[7:0] A1,B1, C, m1,m2,out;
    reg of,f;
    wire[7:0] A2,B2,C1,C2;
    wire of1,of2, ccf,cf,CCF;
    always@(*)begin
        A1=~A;
```

```
        B1=~B;
    end
    adder_8bit(A1,0,1,A2,);
    adder_8bit(B1,0,1,B2,);
      always @(*)begin
        case (operator)
            4'b0010:  //A+B
                begin
                    C=B;
                    m1=0;
                      m2=0;
                end
            4'b0110: //A-B
                begin
                    C=B2;
                    m1=0;
                    m2=0;
                end
            4'b1110:  //A*B
                begin
                    C=0;
                    if(A[7]==1'b1)m1=A2;
                    else m1=A;
                    if(B[7]==1'b1)m2=B2;
                    else m2=B;
                end
            4'b0111:  //小于置位
                begin
                    C=B2;
                    m1=0;
                    m2=0;
                end
        endcase
    end
    adder_8bit(A,C,0,C1,of1);
    mult(m1,m2,A[7]^B[7],C2,of2);
    adder f0(A[7],C[7],0,ccf,cf);
     always @(*)begin  //数据预处理
        if(operator==4'b0010||operator==4'b0110)begin  //判断是否为加法器
            out=C1;
            of=of1;
            f=1;
        end
        else if(operator==4'b1110)begin  //判断是否为乘法
            out=C2;
           of=of2;
            f=1;
        end
        else if(operator==4'b0000)begin  //A&B
           out=A&B;
            of=0;
        end
        else if(operator==4'b0001)begin  //A|B
            out=A|B;
            of=0;
            f=0;
```

```
        end
        else if(operator==4'b1100)begin   //~(A|B)
            out=~(A|B);
            of=0;
            f=0;
          end
 else if(operator==4'b0111)begin   //小于置位
            out=C1[7];
            of=0;
            f=0;
        end
  else if(operator==4'b1101)begin   //~(A & B)
                    out=~(A & B);
            of=0;
            f=0;
        end
     end

     assign r=out;
     assign CCF = ccf | ((cf == 1) & (r[7] == 0));
     assign OF=of;
     assign ZF = (r == 0);
    assign SF = OF^r[7];
     assign PF = r[7]^r[6]^r[5]^r[4]^r[3]^r[2]^r[1]^r[0];
     assign CF = (CCF^(operator[3:1]==3'b001))&f;
       endmodule
//CPU:
 module CPU(
    input clk, // 输入时钟
    input [9:0] b, // 10位指令
    output reg [7:0] data_out, //  输出看一下结果
    output reg [4:0] XF,
    output reg [7:0] led_id,
    output reg [6:0] out_led
 );
    reg [7:0] mymem [2:0]; // 定义 32 个 8 位存储器（mymem[i]表示第 i 个存储器）
    wire [7:0] zc;
      reg [4:0] lt;
    reg [1:0] addr;
    wire ZF,CF,OF,SF,PF;
    wire [3:0] code;
    assign code = b[7:4];
    always@ (posedge clk) begin
        if (b[8] && b[7] && !b[6]) begin                //10存储操作
            if (b[5:4]) mymem[1] = {b[3],b[3],b[3],b[3],b[3:0]};
            if (!b[5:4]) mymem[0] = {b[3],b[3],b[3],b[3],b[3:0]};
        end
        if (b[8] && b[7] && b[6] && b[5] && b[4] && !b[3] && !b[2]) begin  //111100读
取操作
            addr = b[1:0];
            data_out = mymem[addr];
            if (addr == 2) XF = lt;
            else XF = 0;
        end
        else begin
            data_out = 0; XF = 0;
```

```
        end
    end

    always@ (posedge b[9]) begin
        if (b[8] && !b[7] || b[8] && b[7] && b[6]&&~(b[5]&b[4]) )   //0开头的指令或者11**且不为1111时才会给标志位赋值
            mymem[2] = zc;
            lt = {ZF,CF,OF,SF,PF};
    end
    wire [1:0] A,B;
      assign A = b[3:2];
    assign B = b[1:0];
    alu(code,mymem[A],mymem[B],zc,ZF,CF,OF,SF,PF);
    wire [7:0] n;
    assign n = mymem[addr];
      parameter CLK_COUNT = 249999;//时钟计数上限
    reg [31:0] count;//计数
    reg [1:0] id;//id0~3对应左到右四个数码管
    wire flag;  //flag标记补码是否表示负数
    assign flag=n[7];
//8位2进制，十进制至多3位
    reg [7:0] n1;    //百位
    reg [7:0] n2;    //十位
    reg [7:0] n3;    //个位
    reg [7:0] abs;
    reg [7:0] tmp1;
    reg [1:0] tmp2;
    reg [31:0] tmp3;
    always @(*)
    case(flag) //求正数的个十百位
    1'b0:
        begin
           n1 = n / 100 % 10;
           n2 = n / 10 % 10;
           n3 = n % 10;
        end
    1'b1:
        begin
           tmp1=1;
           abs=~n;  //求负数的个十百位
            while((abs&tmp1)!=0)begin
                abs=abs^tmp1;
               tmp1=tmp1<<1;
           end
           abs=abs|tmp1;
           n1 = abs / 100 % 10;
          n2 = abs / 10 % 10;
          n3 = abs % 10;
        end
    endcase
    always @ (posedge clk)//时钟上升沿
    begin //根据时钟信号控制切换显示的数码管
         if (count == CLK_COUNT) begin
      count <= 0;
      id <= (id + 1); //切换
   end
```

```
    else count <= count+1;
end

    //数码管显示
     always @ (*)
     begin
         if (b[8] && b[7] && b[6] && b[5] && b[4] && !b[3] && !b[2])
         begin
             if (id == 0)
               begin
                 led_id <= 8'b1111_0111;//第四位的灯
                 if(n[7]==1)
                 out_led<=7'b1111110;//负号
                 else out_led<=7'b1111111;//不显示
             end
             else if (id == 1)//其余三个管显示逻辑相同
             begin
                 led_id <= 8'b1111_1011;
                 begin
                 case(n1)
                   4'b0000: out_led = 7'b0000001;    //0
                   4'b0001: out_led = 7'b1001111;    //1
                4'b0010: out_led = 7'b0010010;    //2
                4'b0011: out_led = 7'b0000110;    //3
                4'b0100: out_led = 7'b1001100;    //4
                4'b0101: out_led = 7'b0100100;    //5
                4'b0110: out_led = 7'b0100000;    //6
                4'b0111: out_led = 7'b0001111;    //7
                4'b1000: out_led = 7'b0000000;    //8
                4'b1001: out_led = 7'b0000100;    //9
                  endcase
                  end
             end
             else if (id == 2)
             begin
                 led_id <= 8'b1111_1101;
                 begin
                 case(n2)
                 4'b0000: out_led = 7'b0000001;    //0
              4'b0001: out_led = 7'b1001111;    //1
              4'b0010: out_led = 7'b0010010;    //2
              4'b0011: out_led = 7'b0000110;    //3
              4'b0100: out_led = 7'b1001100;    //4
              4'b0101: out_led = 7'b0100100;    //5
              4'b0110: out_led = 7'b0100000;    //6
              4'b0111: out_led = 7'b0001111;    //7
              4'b1000: out_led = 7'b0000000;    //8
              4'b1001: out_led = 7'b0000100;    //9
                 endcase
                 end
             end
             else if (id == 3)
             begin
                 led_id <= 8'b1111_1110;
                 begin
                 case(n3)
```

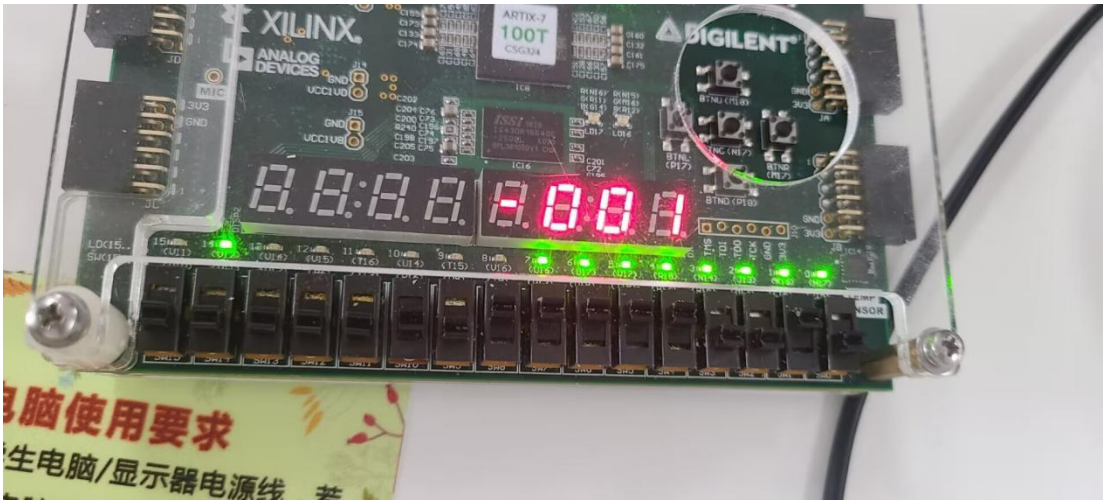
```
                4'b0000: out_led = 7'b0000001;    //0
            4'b0001: out_led = 7'b1001111;    //1
            4'b0010: out_led = 7'b0010010;    //2
            4'b0011: out_led = 7'b0000110;    //3
            4'b0100: out_led = 7'b1001100;    //4
            4'b0101: out_led = 7'b0100100;    //5
            4'b0110: out_led = 7'b0100000;    //6
            4'b0111: out_led = 7'b0001111;    //7
            4'b1000: out_led = 7'b0000000;    //8
            4'b1001: out_led = 7'b0000100;    //9
                endcase
                end
             end
        end
        else begin
            led_id <= 8'b1111_0000;
           out_led = 7'b1111111;
        end
    end
endmodule


endmodule
```

七、实验结果展示

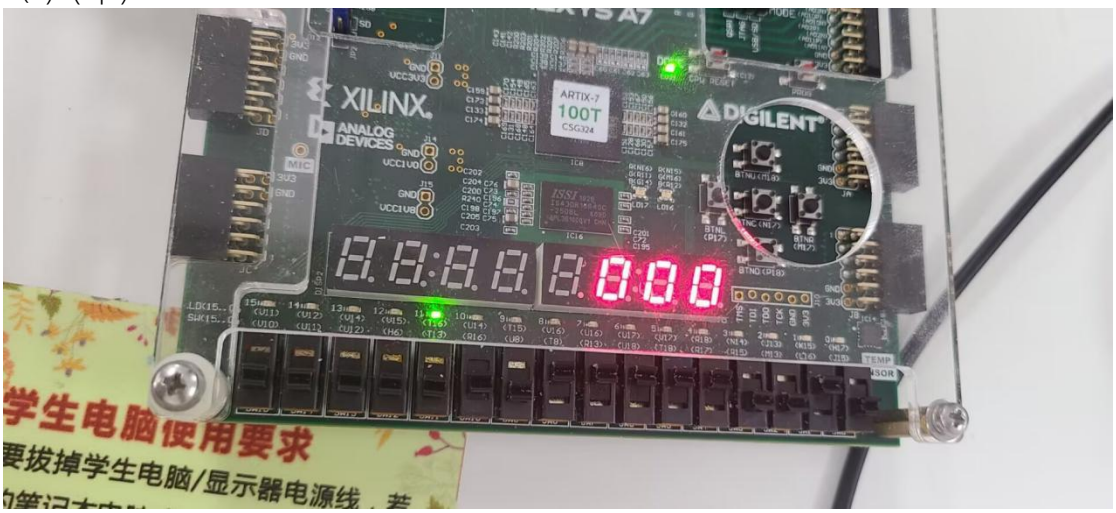
实验结果均为-5和4运算的结果

（1）-5*4

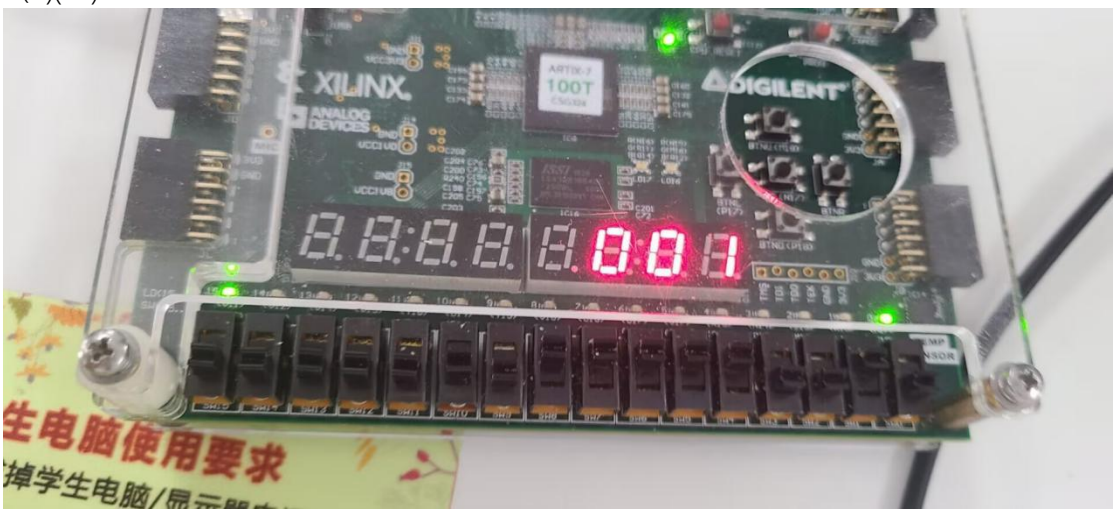
（2）-5+4


XILINX
ANALOG DEVICES
DIGILENT
ARTIX-7
100T
CSG324
电脑使用要求
学生电脑/显示器电源线

(3)~(A|B)


XILINX
ANALOG DEVICES
DIGILENT
ARTIX-7
100T
CSG324
学生电脑使用要求
要拔掉学生电脑/显示器电源线

(4)(-5)&4


XILINX
DIGILENT
生电脑使用要求

(5)-5-4


XILINX
DIGILENT
学生电脑使用要求
.请不要拔掉学生电脑/显示器电源线，若
自己的笔记本电脑或开发板，找空余接口

(6)~(A&B)


NEXYS A7
XILINX
DIGILENT
学生电脑使用要求
1.请不要拔掉学生电脑/显示器电源线，若

(7)-5|4

(8) 有符号数比较，-5<4,置零

(9) （9）输入数据


XILINX
DIGILENT
0 0 9
学生电
不要拔掉学生电脑/显示器电源线，若
的笔记本电脑或开发板，找空余接口


XILINX
ANALOG DEVICES
DIGILENT
ARTIX-7
100T
0 0 9
学生电脑
1.请不要拔掉学生电脑/显示器电源线，若
接入自己的笔记本电脑或开发板，找空余接口