数字电路与数字系统

实验报告

实　　验: CPU 数据通路

姓　名: 周心同

学　　号: 201220069

目录

1.实验目的

在单周期 CPU 的实现之前，先完成 CPU 数据通路中的三个重要部分：寄存器堆、ALU 和数据存储器，并通过功能仿真测试。

2.实验原理

2.1 寄存器堆

RV32I 共 32 个 32bit 的通用寄存器 x0~x31(寄存器地址为 5bit 编码），其中寄存器 x0 中的内容总是 0，无法改变。其他寄存器的别名和寄存器使用约定参见表 10 1。需要注意的是，部分寄存器在函数调用时是由调用方（Caller）来负责保存的，部分寄存器是由被调用方（Callee）来保存的。在进行 C 语言和汇编混合编程时需要注意。

表 10-1: RV32I 中通用寄存器的定义与用法

Register	Name	Use	Saver
x0	zero	Constant 0	–
x1	ra	Return Address	Caller
x2	sp	Stack Pointer	Callee
x3	gp	Global Pointer	–
x4	tp	Thread Pointer	–
x5~x7	t0~t2	Temp	Caller
x8	s0/fp	Saved/Frame pointer	Callee
x9	s1	Saved	Callee
x10~x11	a0~a1	Arguments/Return Value	Caller
x12~x17	a2~a7	Arguments	Caller
x18~x27	s2~s11	Saved	Callee
x28~x31	t3~t6	Temp	Caller

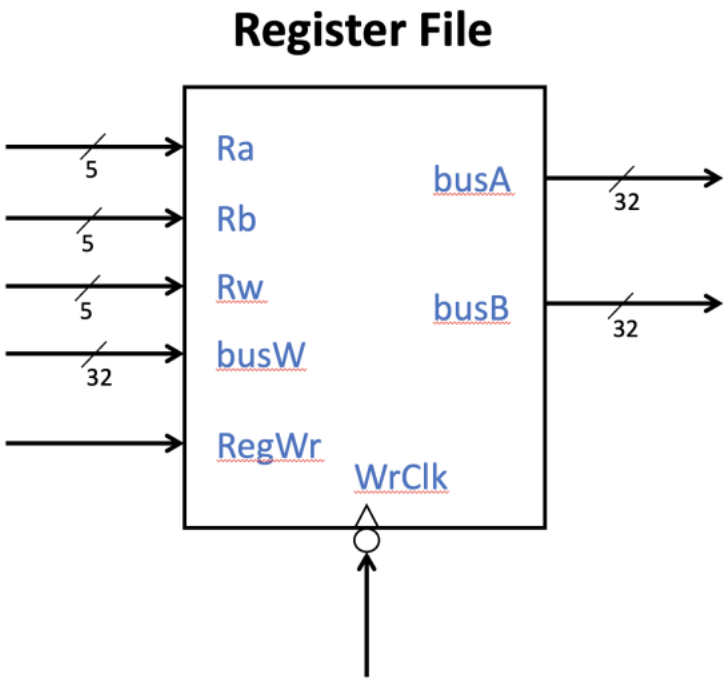

图 10-1: 寄存器堆接口示意图

上图描述了寄存器堆的接口，该寄存器堆中有 32 个 32bit 的寄存器。寄存器堆需要支持同时两个读操作和一个写操作。因此需要有 2 个读地址 Ra 和 Rb，分别对应 RISC-V 汇编中的 rs1 和 rs2。写地址为 Rw，对应 rd。地址均是 5 位。写入数据 busW 为 32 位，写入有效控制为一位高电平有效的 RegWr 信号。寄存器堆的输出是 2 个 32 位的寄存器数据，分别是 busA 和 busB。寄存器堆有一个控制写入的时钟 WrClk。在时序上我们可以让读取是非同步的，即地址改变立刻输出。写入可以在时钟下降沿写入。注意，寄存器 x0 需要特殊处理，不论何时都是全零。

2.2 ALU

ALU 是 CPU 中的核心数据通路部件之一，它主要完成 CPU 中需要进行的算术逻辑运算，我们在前面的实验中已经实现了一个简单的 ALU。在本实验中只需要对该 ALU 稍加改造即可。针对 RV32I 的运算需求，我们对 ALU 的控制信号进行了重

新定义，如表 10.2 所示。该 ALU 的逻辑图如图 10 2 所示。ALU 对输入数据并行地进行加减法、移位、比较大小、异或等操作。最终 ALUout 输出是通过一个八选一选择器选择不同运算部件的结果，选择器的控制端可以用 ALUctr[2:0] 直接生成。ALU 其他部件的控制信号需要的控制信号包括：A/L 控制移位器进行算术移位还是逻辑移位，L/R 控制是左移还是右移，U/S 控制比较大小是带符号比较还是无符号比较，S/A 控制是加法还是减法。这些控制信号需要按照所需进行的操作对应设置，请同学们自行设计。注意：比较大小或判断相等时应使用减法操作。

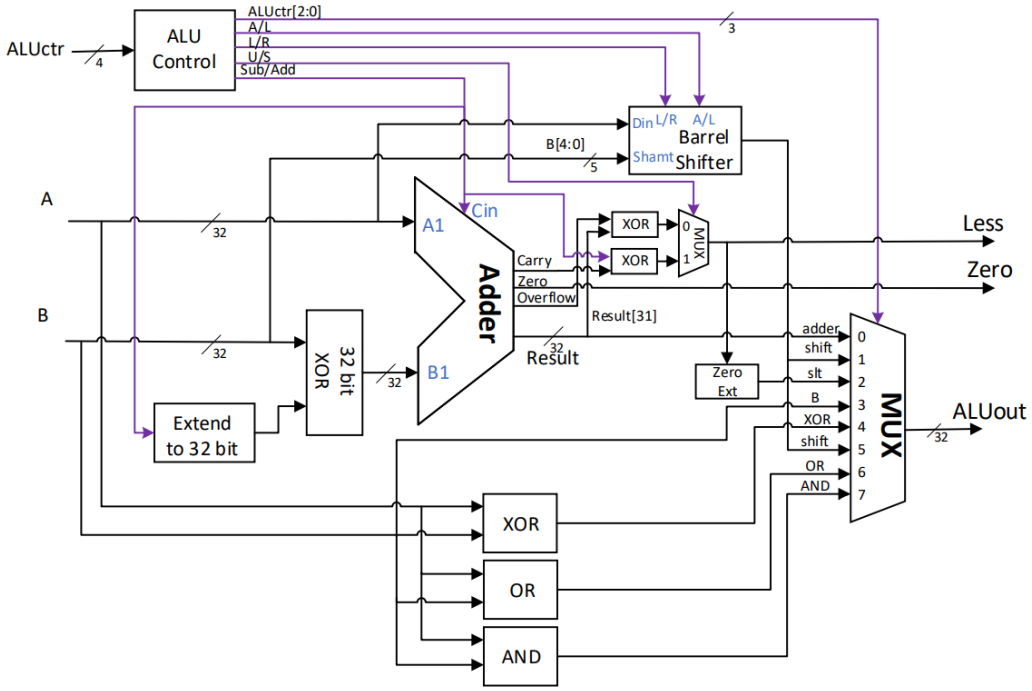

图 10-2: ALU 电路示意图

2.3 数据存储器

这里我用的方法是：利用 4 片 8bit RAM 拼接成一个 32bit 的存储器。用 4 片 8bit 的 RAM 拼接成一个 32bit 的 RAM，每个 8bitRAM 负责 32 比特的给定部分。例如，RAM0 负责提供地址低两位为 00 的数据，RAM1 负责提供地址低两位为 01 的数据，以此类推。如果需要一次性读写 32bit 数据，我们将对应的数据和地址前 30 位连接到 4 片 RAM 上即可同时对 4 片 RAM 进行操作，一次读写 4 × 8 =32bit 数据。如果只需要写入 8bit 数据，可以根据地址低两位来控制 RAM 写使能端口，只对一片 RAM 进行写入。

3.实验环境

- 软件环境

Quartus 17.1 Lite

- 硬件环境

开发板：DE10 Standard

FPGA：Intel Cyclone V SE 5CSXFC6D6F31C6N

4.实验步骤和结果

4.1 ALU

根据 alu 功能要求实现不同的功能。

特别的，对于带符号比较，我先比较了第一位。

less 和 zero 因为是 wire 型，所以用一个 reg 型的 less2 和 zero2 来在 always 语句里面赋值，然后再用 assign 赋值给 less 和 zero。

```
//add your code here
reg less2;
reg zero2;

always @(*)
begin
    case(ALUctr[2:0])
    0:
    begin
    if(ALUctr[3]==0) aluresult=dataa+datab;
    else aluresult=dataa-datab;
    end
    1:aluresult=dataa<<datab[4:0];
    2:if(ALUctr[3]==0)
        begin
        if(dataa[31]==1)
            begin
            if(datab[31]==0) aluresult=1;
            else
                begin
                    if(dataa<datab) aluresult=1;
                    else aluresult=0;
                end
            end
        else
            begin
            if(datab[31]==1) aluresult=0;
            else
                begin
                    if(dataa<datab) aluresult=1;
                    else aluresult=0;
                end
            end
            less2=aluresult;
        end
```

```
        end
    else
        begin
            if(dataa<datab) aluresult=1;
            else aluresult=0;
            less2=aluresult;
        end
    3:aluresult=datab;
    4:aluresult=dataa^datab;
    5:
    if(ALUctr[3]==0)
    aluresult=dataa>>5;
    else
    begin
    aluresult=( {32{dataa[31]}} << ( 6'd32 - {1'b0, datab[4:0]} ) ) | ( dataa >> datab[4:0] ) ;
    end
    6:aluresult=dataa|datab;
    7:aluresult=dataa&datab;


    endcase
    if(ALUctr[2:0]==2)
        if(dataa==datab) zero2=1;
        else  zero2=0;
    else
        if(aluresult==0) zero2=1;
        else zero2=0;

end

assign less=less2;
assign zero=zero2;
```

4.2 数据存储器

依照前文所述，利用 4 片 8bit RAM 拼接成一个 32bit 的存储器。写入的时候根据写入的字节数决定写入的 RAM 个数。读取的时候根据读取要求，先读取相应数量的 RAM，然后根据拓展要求进行对应的拓展即可。

```
always @(posedge wrclk)//write
begin
    if(we)
    begin
       case(memop)
       0:
       begin
       ram[addr]  <= datain[7:0];
       end
       1:
       begin
        ram[addr]  <= datain[7:0];
        ram[addr+1]<= datain[15:8];
       end
       2:
       begin
        ram[addr]  <= datain[7:0];
        ram[addr+1]<= datain[15:8];
        ram[addr+2]<= datain[23:16];
        ram[addr+3]<= datain[31:24];
       end
       endcase

    end
end
```

```
reg [7:0] ram [4096:0];

always @(posedge rdclk)//read
begin
    case(memop)
    0:
    begin
    dataout[7:0]<=ram[addr];
    if(ram[addr][7])
            dataout[31:8]<=24'b111111111111111111111111;
        else
            dataout[31:8]<=24'b000000000000000000000000;
    end
    1:
    begin
    dataout[7:0]<=ram[addr];
    dataout[15:8]<=ram[addr+1];
    if(ram[addr+1][7])
            dataout[31:16]<=16'b1111111111111111;
        else
            dataout[31:16]<=16'b0000000000000000;
    end
    2:dataout<={ram[addr+3],ram[addr+2],ram[addr+1],ram[addr]};
    4:
    begin
    dataout[7:0]<=ram[addr];
    dataout[31:8]<=24'b000000000000000000000000;
    end
    5:
    begin
    dataout[7:0]<=ram[addr];
    dataout[15:8]<=ram[addr+1];
    dataout[31:16]<=16'b0000000000000000;
    end
    endcase
end
```

5. 实验中遇到的问题及解决办法

5.1. 算术右移时用 for 循环不好综合

通过或赋值将算术右移的几位给补上

```
aluresult=( {32{dataa[31]}} << ( 6'd32 - {1'b0, datab[4:0]} ) ) | ( dataa >> datab[4:0] ) ;
```

6.实验启示和建议

这个实验比较简单，跟实验九和实验十一都不是一个难度的，可以考虑把实验十一的内容再划分一小块过来。