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

利用 FPGA 实现 RV32I 指令集中除系统控制指令之外的其余指令。

利用单周期方式实现 RV32I 的控制通路及数据通路，并能够顺利通过功能仿真。

通过对单周期 CPU 的实际设计来理解 CPU 内部的控制及数据通路原理，并掌握基本的系统级测试和 Debug 技术。

2.实验原理

2.1 RV32 指令集

本实验中需要实现的 RV32I 指令含包含以下三类：

_ 整数运算指令：可以是对两个源寄存器操作数，或一个寄存器一个立即数操作数进行计算后，结果送入目的寄存器。运算操作包括带符号数和无符号数的算术运算、移位、逻辑操作和比较后置位等。

_ 控制转移指令：条件分支包括 beq，bne 等等，根据寄存器内容选择是否跳转。无条件跳转指令会将本指令下一条指令地址 PC+4 存入 rd 中供函数返回时使用。

_ 存储器访问指令：对内存操作是首先寄存器加立即数偏移量，以计算结果为地址读取/写入内存。读写时可以是按 32 位字，16 位半字或 8 位字节来进

行读写。读写时区分无符号数和带符号数。注意：RV32I 为 Load/Store 型架构，内存中所有数据都需要先 load 进入寄存器才能进行操作，不能像 x86 一样直接对内存数据进行算术处理。

2.2 单周期电路设计

所谓单周期 CPU 是指 CPU 在每一个时钟周期中需要完成一条指令的所有操作，即每个时钟周期完成一条指令。每条指令的执行过程一般需要以下几个步骤：

1. 取指令：使用本周期新的 PC 从指令存储器中取出指令，并将其放入指令寄存器（IR）中

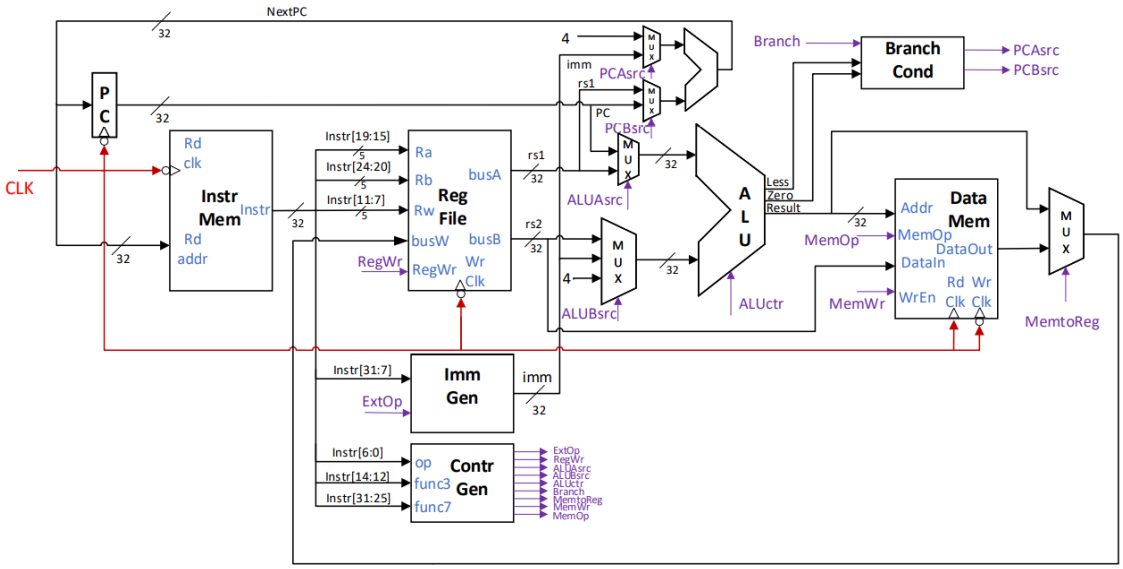
2. 指令译码：对取出的指令进行分析，生成本周期执行指令所需的控制信号，并计算下一条指令的地址

3. 读取操作数：从寄存器堆中读取寄存器操作数，并完成立即数的生成

4. 运算：利用 ALU 对操作数进行必要的运算

5. 访问内存：包括读取或写入内存对应地址的内容

6. 寄存器写回：将最终结果写回到目的寄存器中



2.3 PC 生成

Branch	Zero	Less	PCAsrc	PCBsrc	NextPC
000	×	×	0	0	PC + 4
001	×	×	1	0	PC + imm
010	×	×	1	1	rs1 + imm
100	0	×	0	0	PC + 4
100	1	×	1	0	PC + imm
101	0	×	1	0	PC + imm
101	1	×	0	0	PC + 4
110	×	0	0	0	PC + 4
110	×	1	1	0	PC + imm
111	×	0	1	0	PC + imm
111	×	1	0	0	PC + 4

2.4 控制信号生成

指令	op[6:2]	func3	func7[5]	ExtOP	RegWr	Branch	MemtoReg	MemWr	MemOP
lui	01101	×	×	001	1	000	0	0	×
auipc	00101	×	×	001	1	000	0	0	×
addi	00100	000	×	000	1	000	0	0	×
slti	00100	010	×	000	1	000	0	0	×
sltiu	00100	011	×	000	1	000	0	0	×
xori	00100	100	×	000	1	000	0	0	×
ori	00100	110	×	000	1	000	0	0	×
andi	00100	111	×	000	1	000	0	0	×
slli	00100	001	0	000	1	000	0	0	×
srli	00100	101	0	000	1	000	0	0	×
srai	00100	101	1	000	1	000	0	0	×
add	01100	000	0	×	1	000	0	0	×
sub	01100	000	1	×	1	000	0	0	×
sll	01100	001	0	×	1	000	0	0	×
slt	01100	010	0	×	1	000	0	0	×
sltu	01100	011	0	×	1	000	0	0	×
xor	01100	100	0	×	1	000	0	0	×
srl	01100	101	0	×	1	000	0	0	×
sra	01100	101	1	×	1	000	0	0	×
or	01100	110	0	×	1	000	0	0	×
and	01100	111	0	×	1	000	0	0	×
jal	11011	×	×	100	1	001	0	0	×
jalr	11001	000	×	000	1	010	0	0	×
beq	11000	000	×	011	0	100	×	0	×
bne	11000	001	×	011	0	101	×	0	×
blt	11000	100	×	011	0	110	×	0	×
bge	11000	101	×	011	0	111	×	0	×
bltu	11000	110	×	011	0	110	×	0	×
bgeu	11000	111	×	011	0	111	×	0	×
lb	00000	000	×	000	1	000	1	0	000
lh	00000	001	×	000	1	000	1	0	001
lw	00000	010	×	000	1	000	1	0	010
lbu	00000	100	×	000	1	000	1	0	100
lhu	00000	101	×	000	1	000	1	0	101
sb	01000	000	×	010	0	000	×	1	000
sh	01000	001	×	010	0	000	×	1	001
sw	01000	010	×	010	0	000	×	1	010

指令	op[6:2]	func3	func7[5]	ALUAsrc	ALUBsrc	ALUctr	ALU 动作
lui	01101	×	×	×	01	0011	拷贝立即数
auipc	00101	×	×	1	01	0000	PC + 立即数
addi	00100	000	×	0	01	0000	rs1 + imm
slti	00100	010	×	0	01	0010	slt rs1, imm
sltiu	00100	011	×	0	01	1010	sltu rs1, imm
xori	00100	100	×	0	01	0100	rs1 ⊕ imm
ori	00100	110	×	0	01	0110	rs1 | imm
andi	00100	111	×	0	01	0111	rs1 & imm
slli	00100	001	0	0	01	0001	rs1 << imm[4:0]
srli	00100	101	0	0	01	0101	rs1 >> imm[4:0]
srai	00100	101	1	0	01	1101	rs1 >>> imm[4:0]
add	01100	000	0	0	00	0000	rs1 + rs2
sub	01100	000	1	0	00	1000	rs1 - rs2
sll	01100	001	0	0	00	0001	rs1 << rs2[4:0]
slt	01100	010	0	0	00	0010	slt rs1, rs2
sltu	01100	011	0	0	00	1010	sltu rs1, rs2
xor	01100	100	0	0	00	0100	rs1 ⊕ rs2
srl	01100	101	0	0	00	0101	rs1 >> rs2[4:0]
sra	01100	101	1	0	00	1101	rs1 >>> rs2[4:0]
or	01100	110	0	0	00	0110	rs1 | rs2
and	01100	111	0	0	00	0111	rs1 & rs2
jal	11011	×	×	1	10	0000	PC + 4
jalr	11001	000	×	1	10	0000	PC + 4
beq	11000	000	×	0	00	0010	根据 rs1，rs2 设置 Zero
bne	11000	001	×	0	00	0010	根据 rs1，rs2 设置 Zero
blt	11000	100	×	0	00	0010	根据 rs1，rs2 设置带符号 Less
bge	11000	101	×	0	00	0010	根据 rs1，rs2 设置带符号 Less
bltu	11000	110	×	0	00	1010	根据 rs1，rs2 设置无符号 Less
bgeu	11000	111	×	0	00	1010	根据 rs1，rs2 设置无符号 Less
lb	00000	000	×	0	01	0000	rs1 + imm
lh	00000	001	×	0	01	0000	rs1 + imm
lw	00000	010	×	0	01	0000	rs1 + imm
lbu	00000	100	×	0	01	0000	rs1 + imm
lhu	00000	101	×	0	01	0000	rs1 + imm
sb	01000	000	×	0	01	0000	rs1 + imm
sh	01000	001	×	0	01	0000	rs1 + imm
sw	01000	010	×	0	01	0000	rs1 + imm

2.5 寄存器

32 个 32 位寄存器，即时读取，写入时注意时序即可。也可用上次实验的。

2.5 立即数

同样的，也可以利用立即数生成器 imm Generator 生成所有的立即数。注意，所有立即数均是符号扩展，且符号位总是 instr[31]:

```
assign immI = {{20{instr[31]}}, instr[31:20]};
```

12

```
assign immU = {instr[31:12], 12'b0};
assign immS = {{20{instr[31]}}, instr[31:25], instr[11:7]};
assign immB = {{20{instr[31]}}, instr[7], instr[30:25], instr[11:8], 1'b0};
assign immJ = {{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0};
```

2.5 ALU

即上次实验的 ALU。

3.实验环境

- 软件环境

Quartus 17.1 Lite 以及 头歌

4.实验步骤和结果

4.1 顶层模块

```
assign rs1 = imemdataout[19:15];
assign rs2 = imemdataout[24:20];
assign rd = imemdataout[11:7];

assign imemclk=~clock;//指令 下降沿读取
assign dmemrdclk=clock;//数据 上升沿读取
assign dmemwrclk=~clock;//数据 下降沿写入

Control mycontrol(imemdataout[6:2],imemdataout[14:12],imemdataout[30],ExtOP,RegWr,Branch,MemtoReg,MemWr
regfile myregfile(rs1,rs2,rd,wrdata,RegWr,~clock,rs1data,rs2data);//下降沿写入 读
immediate myimm(imemdataout,ExtOP,imm);
PC_ myPC(PC,imm,rs1data,Branch,zero,less,NextPC);
ALU myALU(dataa,datab,ALUctr,less,zero,aluresult);

always @(negedge clock)
if(reset)
   PC=0;
else
   PC=NextPC;


always @(*)
begin
if(ALUAsrc==0)
   dataa=rs1data;
else
   dataa=PC;
if(ALUBsrc==2'b00)
   datab=rs2data;
else if(ALUBsrc==2'b01)
   datab=imm;
else if(ALUBsrc==2'b10)
   datab=4;
if(MemtoReg)
   wrdata=dmemdataout;
else
   wrdata=aluresult;
end

assign imemaddr=PC;
assign dbgdata=PC;
assign dmemwe=MemWr;
assign dmemop=MemOP;
assign dmemdatain=rs2data;
assign dmemaddr=aluresult;
```

这是第一版

```
assign rs1 = imemdataout[19:15];
assign rs2 = imemdataout[24:20];
assign rd = imemdataout[11:7];

assign imemclk=~clock;//指令 下降沿读取
assign dmemrdclk=clock;//数据 上升沿读取
assign dmemwrclk=~clock;//数据 下降沿写入

Control A(imemdataout[6:2],imemdataout[14:12],imemdataout[30],ExtOP,RegWr,Branch,MemtoReg,MemWr,MemOP,ALUAsrc,ALUBsrc,ALUctr);
regfile myregfile(rs1,rs2,rd,wrdata,RegWr,~clock,rs1data,rs2data);//下降沿写入 读取
immediate B(imemdataout,ExtOP,imm);
PC_ C(PC,imm,rs1data,Branch,zero,less,NextPC,reset);
ALU D(dataa,datab,ALUctr,less,zero,aluresult);

always @(negedge clock)
   PC<=NextPC;

always @(*)
begin
if(MemtoReg)
   wrdata=dmemdataout;
else
   wrdata=aluresult;
if(ALUAsrc==0)
   dataa=rs1data;
else
   dataa=PC;
if(ALUBsrc==2'b00)
   datab=rs2data;
else if(ALUBsrc==2'b01)
   datab=imm;
else if(ALUBsrc==2'b10)
   datab=4;
end

assign imemaddr=NextPC;
assign dbgdata=PC;
assign dmemwe=MemWr;
assign dmemop=MemOP;
assign dmemdatain=rs2data;
assign dmemaddr=aluresult;
```

这是第二版

4.2 PC

```
module PC_(
input [31:0]PC,
input [31:0]imm,
input [31:0]rs1data,
input [2:0]Branch,
input zero,
input less,
output reg [31:0] NextPC
);

always @(*)
begin
if(Branch==0) NextPC=PC+4;
else if(Branch==1) NextPC=PC+imm;
else if(Branch==2) NextPC=rs1data+imm;
else if(Branch==4 && zero==0) NextPC=PC+4;
else if(Branch==4 && zero==1) NextPC=PC+imm;
else if(Branch==5 && zero==0) NextPC=PC+imm;
else if(Branch==5 && zero==1) NextPC=PC+4;
else if(Branch==6 && less==0) NextPC=PC+4;
else if(Branch==6 && less==1) NextPC=PC+imm;
else if(Branch==7 && less==0) NextPC=PC+imm;
else if(Branch==7 && less==1) NextPC=PC+4;


end

endmodule
```

这是第一版

```
module PC_(
input [31:0]PC,
input [31:0]imm,
input [31:0]rs1data,
input [2:0]Branch,
input zero,
input less,
output reg [31:0] NextPC,
input reset
);
initial begin

NextPC=0;
end

always @(*)
begin
if(reset) NextPC=0;
else if(Branch==0) NextPC=PC+4;
else if(Branch==1) NextPC=PC+imm;
else if(Branch==2) NextPC=rs1data+imm;
else if(Branch==4 && zero==0) NextPC=PC+4;
else if(Branch==4 && zero==1) NextPC=PC+imm;
else if(Branch==5 && zero==0) NextPC=PC+imm;
else if(Branch==5 && zero==1) NextPC=PC+4;
else if(Branch==6 && less==0) NextPC=PC+4;
else if(Branch==6 && less==1) NextPC=PC+imm;
else if(Branch==7 && less==0) NextPC=PC+imm;
else if(Branch==7 && less==1) NextPC=PC+4;
end
endmodule
```

这是第二版

4.3 立即数

```
module immediate(
input [31:0] instr,
input [2:0] ExtOP,
output reg [31:0]imm
);

always @(*)
begin
if(ExtOP == 3'b000) imm={{20{instr[31]}}, instr[31:20]};
else if(ExtOP == 3'b001) imm={instr[31:12], 12'b0};
else if(ExtOP == 3'b010) imm={{20{instr[31]}}, instr[31:25], instr[11:7]};
else if(ExtOP == 3'b011) imm={{20{instr[31]}}, instr[7], instr[30:25], instr[11:8], 1'b0};
else if(ExtOP == 3'b100) imm={{12{instr[31]}}, instr[19:12], instr[20], instr[30:21], 1'b0};
end


endmodule
```

4.4 控制信号

```
always @(*)
begin
if(op==5'b01101)
        begin ExtOP = 3'b001;RegWr=1;Branch=3'b000;MemtoReg=0;MemWr=0;ALUBsrc=1;ALUctr=3; end
else if(op==5'b00101)
        begin ExtOP = 3'b001;RegWr=1;Branch=3'b000;MemtoReg=0;MemWr=0;ALUAsrc=1;ALUBsrc=1;ALUctr=0; end
else if(op==5'b00100)
        begin
        ExtOP = 3'b000;RegWr=1;Branch=3'b000;MemtoReg=0;MemWr=0;
        ALUAsrc=0;ALUBsrc=1;
        if(func3==0) ALUctr=0;
        else if(func3==2) ALUctr=2;
        else if(func3==3) ALUctr=10;
        else if(func3==4) ALUctr=4;
        else if(func3==6) ALUctr=6;
        else if(func3==7) ALUctr=7;
        else if(func3==1 && func7==0 ) ALUctr=1;
        else if(func3==5 && func7==0 ) ALUctr=5;
        else if(func3==5 && func7==1 ) ALUctr=13;
        end
else if(op==5'b01100)
        begin
        RegWr=1;Branch=3'b000;MemtoReg=0;MemWr=0;ALUAsrc=0;ALUBsrc=0;
        if({func3[2:0],func7}==4'b0000) ALUctr=4'b0000;
        else if({func3[2:0],func7}==4'b0001) ALUctr=4'b1000;
        else if({func3[2:0],func7}==4'b0010) ALUctr=4'b0001;
        else if({func3[2:0],func7}==4'b0100) ALUctr=4'b0010;
        else if({func3[2:0],func7}==4'b0110) ALUctr=4'b1010;
        else if({func3[2:0],func7}==4'b1000) ALUctr=4'b0100;
        else if({func3[2:0],func7}==4'b1010) ALUctr=4'b0101;
        else if({func3[2:0],func7}==4'b1011) ALUctr=4'b1101;
        else if({func3[2:0],func7}==4'b1100) ALUctr=4'b0110;
        else if({func3[2:0],func7}==4'b1110) ALUctr=4'b0111;
        end
else if(op==5'b11011)
        begin ExtOP = 3'b100;RegWr=1;Branch=3'b001;MemtoReg=0;MemWr=0;ALUAsrc=1;ALUBsrc=2;ALUctr=0; end
else if(op==5'b11001)
        begin ExtOP = 3'b000;RegWr=1;Branch=3'b010;MemtoReg=0;MemWr=0;ALUAsrc=1;ALUBsrc=2;ALUctr=0; end
else if(op==5'b11000)
        begin ExtOP = 3'b011;RegWr=0;MemWr=0;
        ALUAsrc=0;ALUBsrc=0;
        if(func3==3'b000) Branch=3'b100;
        else if(func3==3'b001) Branch=3'b101;
        else if(func3==3'b100) Branch=3'b110;
        else if(func3==3'b101) Branch=3'b111;
        else if(func3==3'b110) Branch=3'b110;
        else if(func3==3'b111) Branch=3'b111;

        if(func3[1]==1) ALUctr=4'b1010;
        else ALUctr=4'b0010;
        end
else if(op==5'b00000)
        begin ExtOP=3'b000;RegWr=1;Branch=3'b000;MemtoReg=1;MemWr=0;MemOP=func3;ALUAsrc=0;ALUBsrc=1;ALUctr=0;end
else if(op==5'b01000)
        begin ExtOP=3'b010;RegWr=0;Branch=3'b000;MemWr=1;MemOP=func3;ALUAsrc=0;ALUBsrc=1;ALUctr=0;end
end
endmodule
```

其余都比较简单，就不列出了

5. 实验中遇到的问题及解决办法

5.1. PC 的时序处理问题

原本我是通过推测，觉得应该在下降沿读取指令，并结合上个周期从数据存储器读取的数据进行计算，并把上周期算出的数据写入 reg 和数据存储器。然后在上升沿读取数据存储器为下次计算做准备，并根据刚才下降沿计算出的改变 PC 的值。

然后根据验收时老师所说，可以在下降沿改变 PC 的值。（在上升沿可能会在计算机系统中有 bug）

讲义上也提到了这一点，一开始我没发现：（另外讲义上的那张图原来已经指出时钟都应该是上升沿还是下降沿了）

- 周期开始的下降沿将同时用于写入 PC 寄存器和读取指令存储器。由于指令存储器要在下降沿进行读取操作，而 PC 的输出要等到下降沿后才能更新，所以不能拿 PC 的输出做为指令存储器的地址。可以采用 PC 寄存器的输入，NextPC 来做为指令存储器的地址。该信号是组合逻辑，一般在上个周期末就已经准备好。

但是我一开始把 negedge clock 当成了上升沿（以为取反了，实际上是下降沿），于是给 imemaddr 赋值 PC，也过了。 可以看见，我用了阻塞赋值，所以在下降沿结束的时候，PC 的值已经更新了，所以能过。（所以说是负负得正，错了两步，一步是阻塞赋值，一步是赋值 PC）

```
always @(negedge clock)
if(reset)
   PC=0;
else
   PC=NextPC;

always @(*)
begin
if(ALUAsrc==0)
   dataa=rs1data;
else
   dataa=PC;
if(ALUBsrc==2'b00)
   datab=rs2data;
else if(ALUBsrc==2'b01)
   datab=imm;
else if(ALUBsrc==2'b10)
   datab=4;
if(MemtoReg)
   wrdata=dmemdataout;
else
   wrdata=aluresult;
end

assign imemaddr=PC;
```

但是根据老师要求，修改如下：

```
always @(negedge clock)
   PC<=NextPC;
```

```
assign imemaddr=NextPC;
assign dbgdata=PC;
```

由于要用 NextPC，所以在 PC 模块需要大改，一方面 NextPC 也要初始化，另一方面要传入 reset。

```
module PC_(
input [31:0]PC,
input [31:0]imm,
input [31:0]rs1data,
input [2:0]Branch,
input zero,
input less,
output reg [31:0] NextPC,
input reset
);
initial begin

NextPC=0;
end

always @(*)
begin
if(reset) NextPC=0;
else if(Branch==0) NextPC=PC+4;
else if(Branch==1) NextPC=PC+imm;
else if(Branch==2) NextPC=rs1data+imm;
else if(Branch==4 && zero==0) NextPC=PC+4;
else if(Branch==4 && zero==1) NextPC=PC+imm;
else if(Branch==5 && zero==0) NextPC=PC+imm;
else if(Branch==5 && zero==1) NextPC=PC+4;
else if(Branch==6 && less==0) NextPC=PC+4;
else if(Branch==6 && less==1) NextPC=PC+imm;
else if(Branch==7 && less==0) NextPC=PC+imm;
else if(Branch==7 && less==1) NextPC=PC+4;
end
```

这样也通过了。

6.实验启示和建议

这次实验比较简单，除了时序上的问题，大多都是搬运工作和打表工作。