《计算机组成与结构》
课程实验手册

2017 年 11 月 23 日星期四

1　实验平台的安装和使用

1.1实验平台的安装

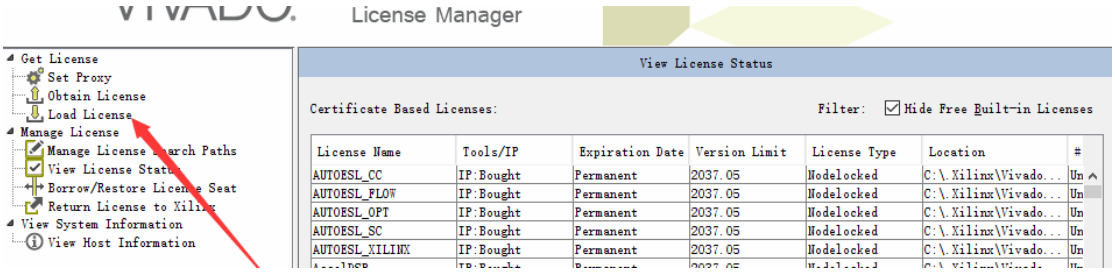
Step1：解压位于 XXX 的安装文件，文件很大，有 20G，需耐心等待。

Step2：在解压后的文件夹根目录中找到 Xsetup.exe 文件双击进行进行安装，安装时间依赖电脑性能，估计在半小时左右。有部分需要选择的地方按照如下所示的图片进行选择，没有图片的地方按照默认选择。

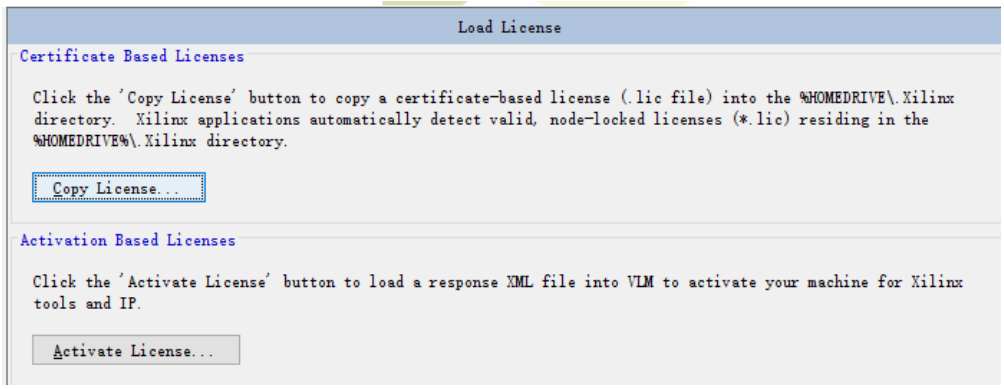
Step3：安装完成后会弹出 vivado License Manager 的程序，如下图所示。

如果没有自动弹出此程序，请在开始菜单->所有程序中选择这个程序打开。

Step4：加载证书文件。



双击如图箭头处的 Load License，会出现如下界面。打开 Copy License，选择在 XXXX 的 lic 文件，确定，就完成了实验平台的安装。



1.2实验平台的使用

在 vivado 中，有两种文件，一种是设计文件，一种是仿真文件，通过给设计文件加上一定的激励，就得到了仿真文件，仿真文件用于对电路进行仿真。
设计文件模块的编写如下：

```
module   设计模块名称(变量 1，变量 2，变量 3,.....);
    input  变量 1,变量 2.....;
   output  变量 m.....;
    语句 1;
    语句 2;
    .....
endmodule
```

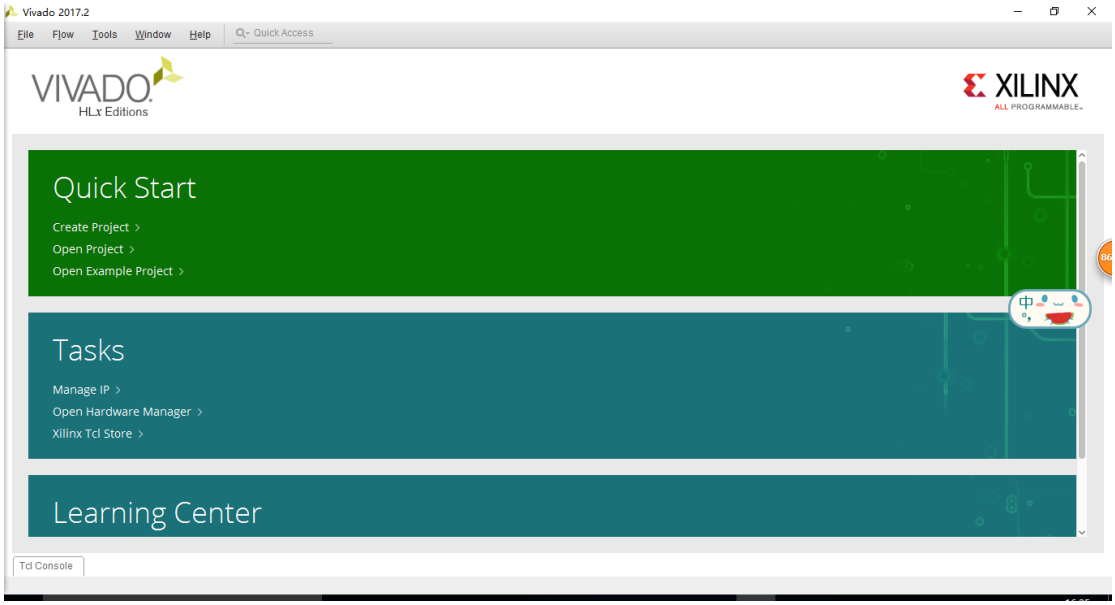
仿真文件模块的编写如下：

```
module  仿真文件模块名称 ;
    // Inputs
    变量类型  变量 1;
    变量类型  变量 2;
    .....
    // Outputs
    变量类型  变量 m;
    .....
    // Instantiate the Unit Under Test (UUT)
        设计模块类型   模块变量名(
            .形参 1（实参 1）,
            .....
            .形参 i(实参 i)      );
        语句 1；
        语句 2；
        .....
endmodule
```

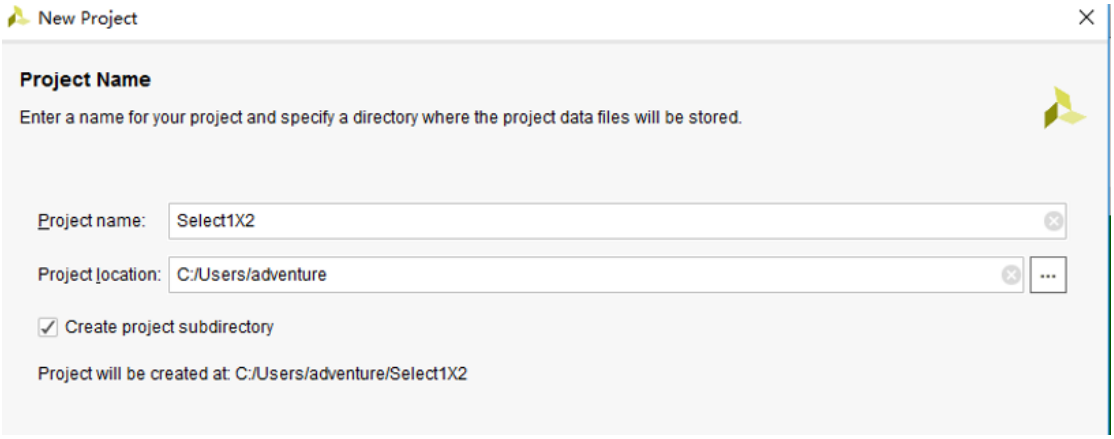
其中，仿真文件中使用设计文件，即绿色部分，如下：

door uut (.x(x),.y(y),.z(z),.f(f));

- Step1：打开 vivado 软件（注意是 vivado，不是 vivado HTL），会进入到如图所示界面。

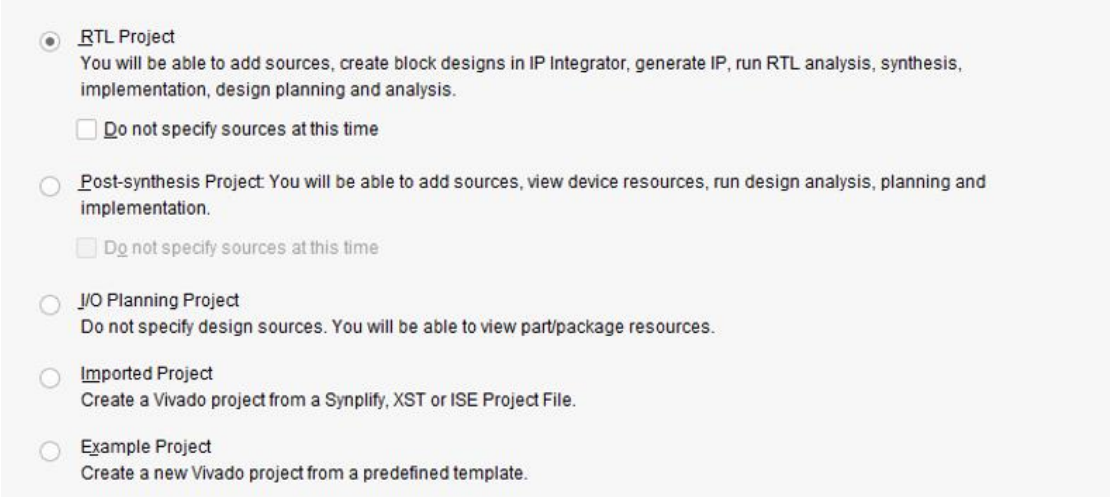


- Step2：建立工程，点击 Quick Start 的 Create Project 按钮，可以快速创建工程。基本所有的设置都可以使用默认的设置。快速建立工程的基本步骤是：命名工程，选择工程类别，选择仿真版的型号，最后建立工程。命名工程如下图所示：



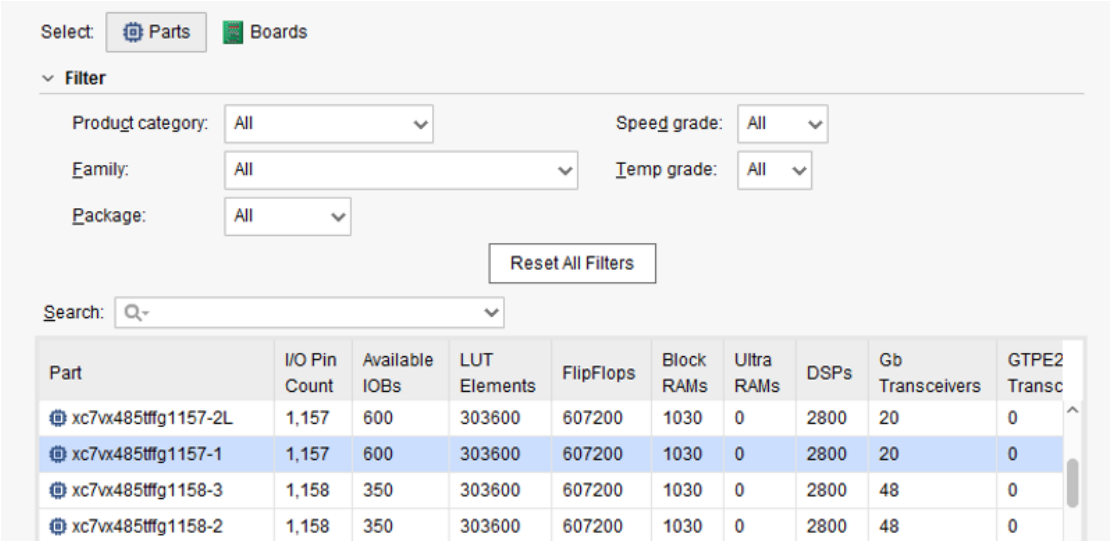
在“Create project subdirectory”处勾选，就表示在已选位置基础上创建一个子目录，不勾就表示不创建。一般情况下按大类来分，比如米尔的文件夹，zingsk 的文件夹，zybo 的文件夹等，所以最好是勾选上。

选择工程类别时，选择 RTL 项目，如下图所示：



最好勾选 RTL Project 下的 Do not specify sources at this time，勾选之后可以在创建工程时跳过创建源文件的过程，加快创建速度。如果不勾选，就会进入具体设置，有硬件语言的类型，ip 的选择等等。此处建议勾选，因为这些可以在工程中设置，没有必要提前设置。

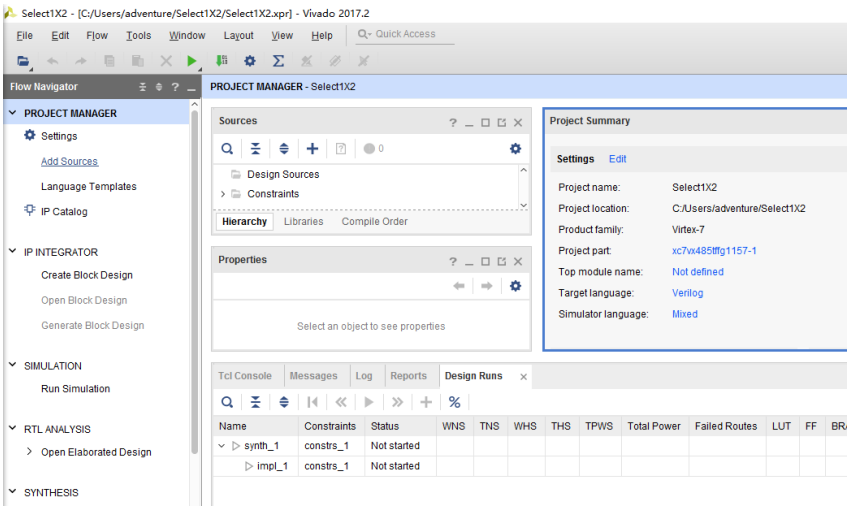
最后是选择仿真版的型号。在不需要仿真版时可以随便选一个，之后用到仿真版的时候再进行修改。选择仿真版的界面如下图所示：



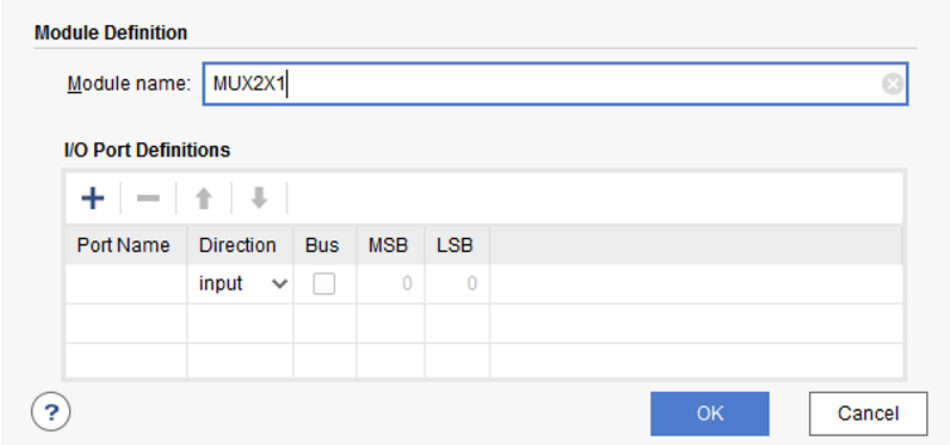
完成之后，创建工程就完成了。

- Step3：设计文件的编写。

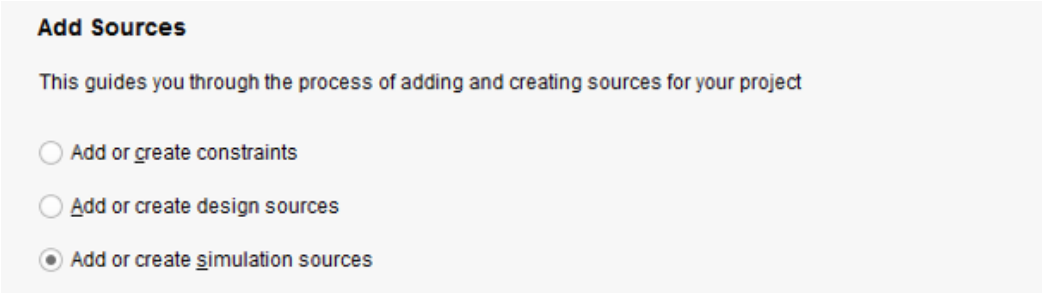
首先我们要新建一个设计文件



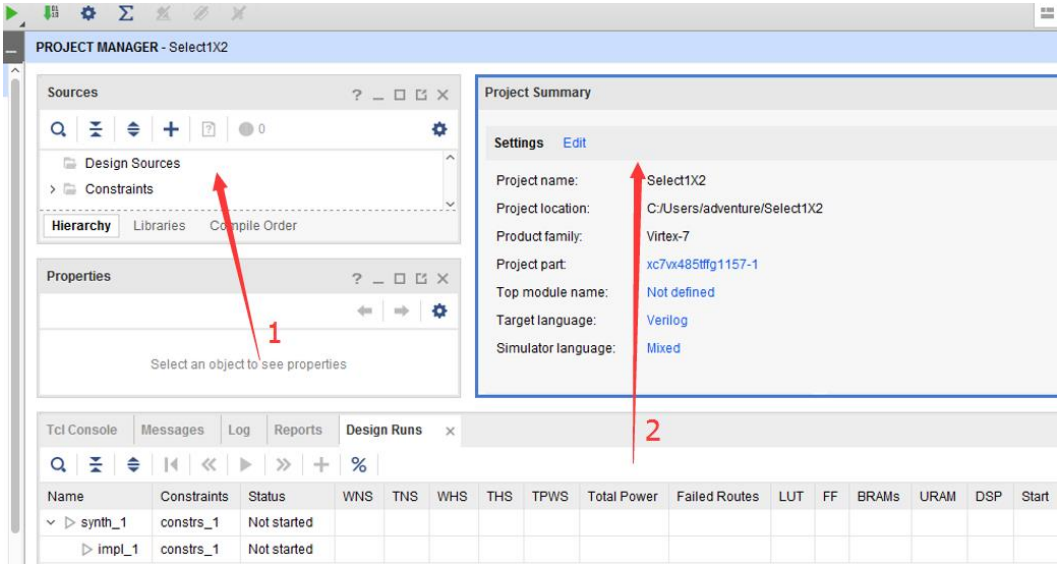
新建工程后会出现上图所示的界面，点击 Project manager 中的 add sources 就可以新建文件。新建文件的过程是给新文件命名，I/O 接口定义（可跳过），选择文件类型。



上图包含了文件命名和 I/O 接口定义，输入了文件名之后可以直接点选 OK。



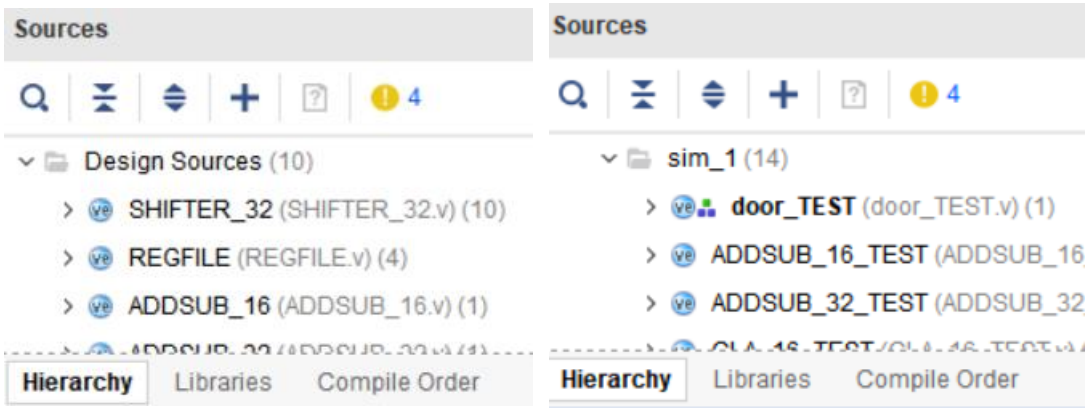
上图是选择文件类型的界面，第一个选项是同时生成设计文件和仿真文件，第二个选项是生成设计文件，第三个选项是生成仿真文件，在这里我们选择第二个文件类型，点击确认就新建了工程。



新建工程后就可以在如上图所示的 1 区域找到新建的文件，双击新建的文件后就可以在如上图所示 2 区域编辑文件。将实例代码复制到对应文件就完成了项目的编写。

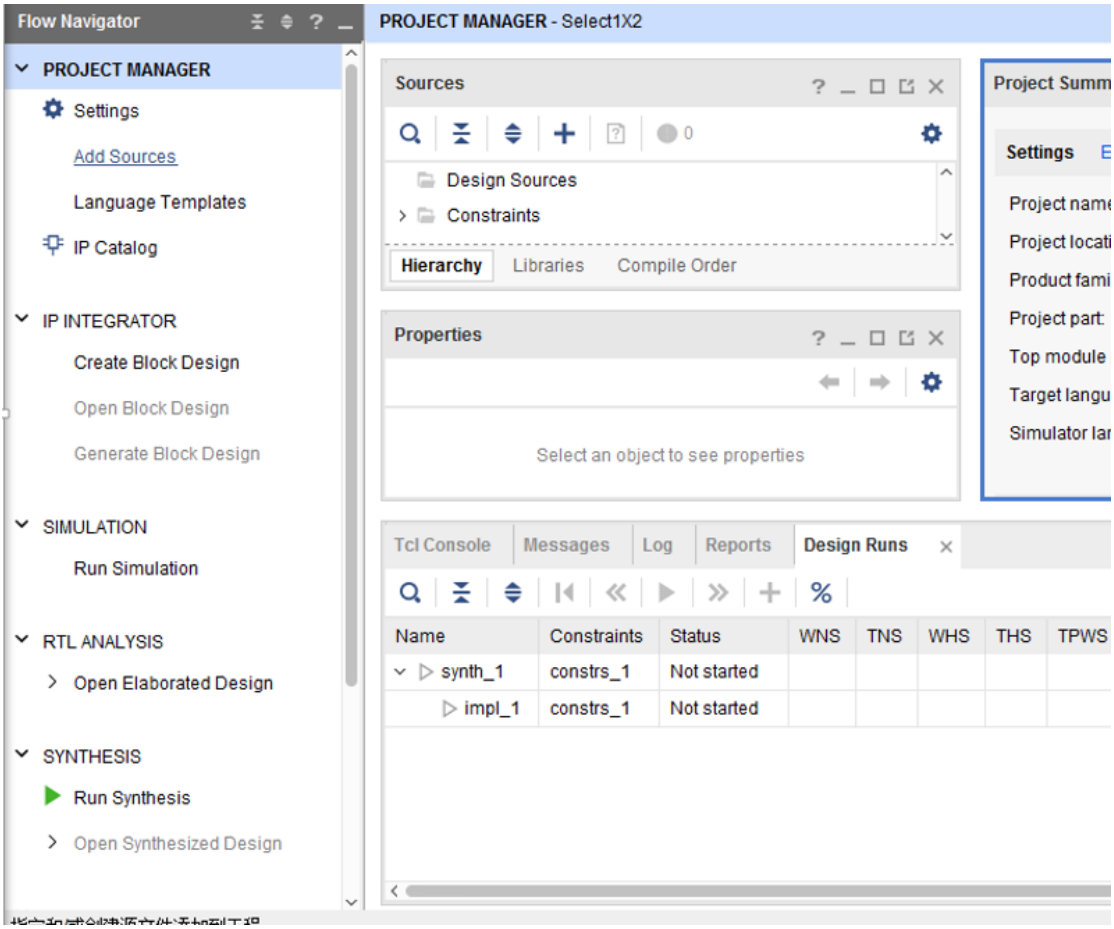
- Step3：RTL 图和仿真。

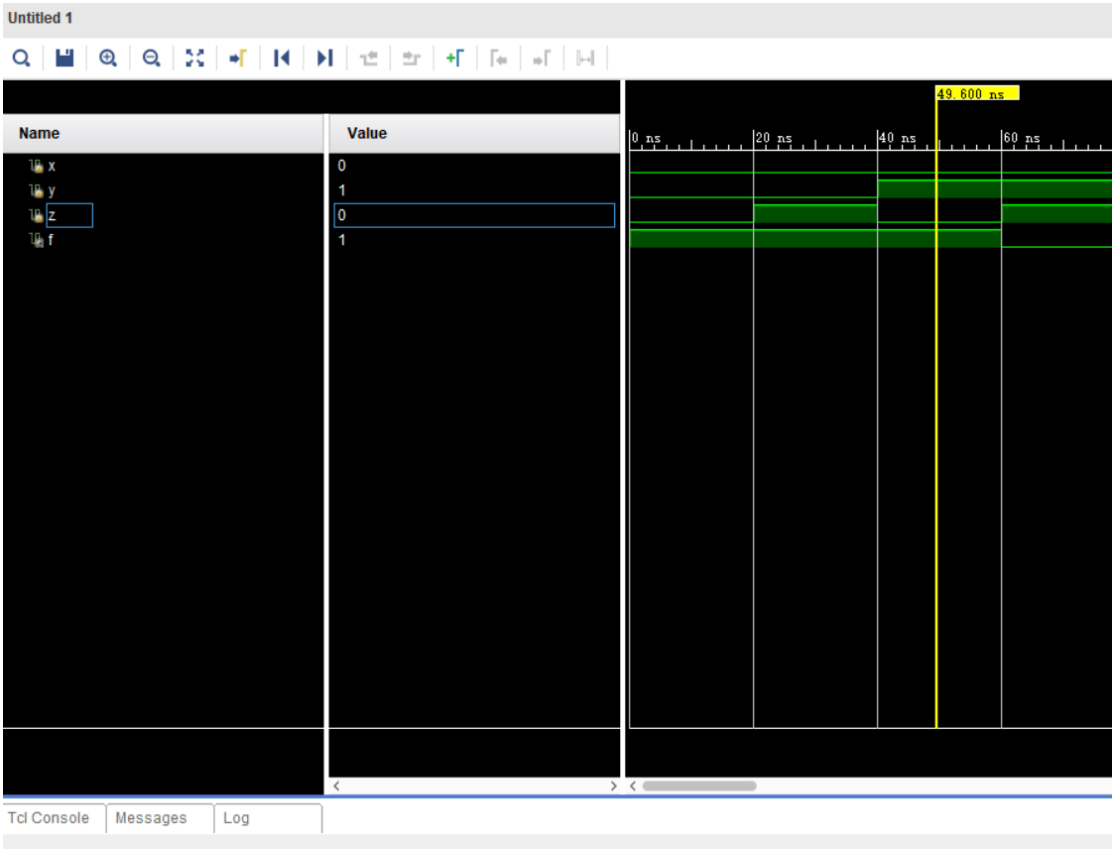
  如图所示，在 Source 区有三个文件夹，分别是 Design Sources，Constraints 和 Simulation Sources，Design Sources 放置了所有的设计代码，Simulation Sources 放置了所有的测试代码。



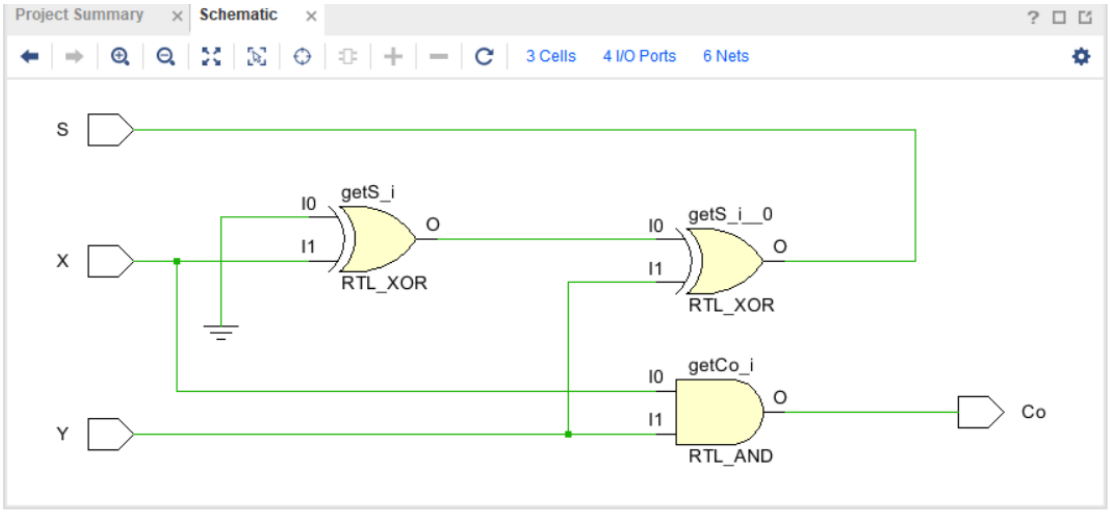
类似于 c 语言的 main 函数，测试文件也有顶层文件的说法，只有设置成顶层测试文件的测试文件才能够进行测试，可以右击你想测试的测试文件选择 Set As Top，将测试文件测试成顶层文件进行测试。特别的，当只有一个测试文件时，该测试文件会被默认成顶层文件，不需要进行多余的设置。

如下图所示，点击左侧的 RTL ANALYSIS 的 open Elaborated Design 就可以看到 RTL 图，如果存在仿真代码，则可点击 Run Simulation 的 run behavioral simulation 看到仿真图。





仿真图例如下图所示，这是一个仿真的波形图，这个界面的最右端为输入和输出的变量名称，中间为当前时间变量的准确值，我们可以左键波形图的任意位置来获得任意位置的变量的准确值。



如上图所示，这就是模仿出来的 RTL 图，可以清楚的看到电路的组织结构。

2 实验内容

2.1总体要求

本课程实验手册总共分为 9 个实验，分别是基本门电路的实现、多路选择器的实现、加减法器的实现、移位器的实现、寄存器堆的实现、存储器的整体封装、ALU 的整体封装、实现支持异常和中断的单周期 CPU 和解决数据冒险的带流水线的 CPU。

本课程实验总共分为 2 个检查点，分别是 12 月 5 日左右，12 月 15 日左右。

第一次检查基本门电路的实现、多路选择器的实现、加减法器的实现、移位器的实现、寄存器堆的实现、存储器的整体封装、ALU 的整体封装，即前 7 个实验。检查内容为实验报告需要提交的内容，包括模块的源代码，仿真代码，RTL 图和仿真波形图。

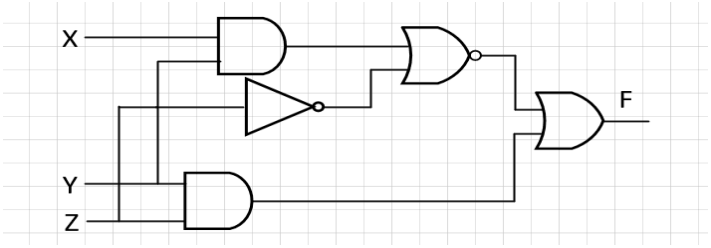
第二次检查支持异常和中断的单周期 CPU，检查内容同上，包括模块的源代码，仿真代码 RTL 图和仿真波形图。

解决数据冒险的带流水线的 CPU 在最后提交的试验报告中检查，依据自己能力可以选做解决其他冒险的流水线 CPU，做了会依据完成情况给分。

2.2各任务要求

2.2.1 实验一 门电路的实现

门电路的电路图如图所示：



需完成设计代码和仿真代码的实现，要求仿真代码实现所有真值的覆盖，并在提交报告时给出设计代码、仿真代码、RTL 图和仿真波形图。

设计代码：

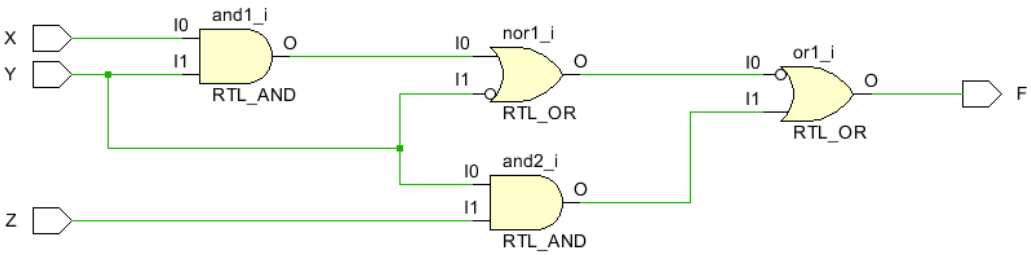
```
module gate_circuit(X, Y, Z, F);
    input X, Y, Z;
    output F;
```

```
    and and1 (XandY, X, Y);
    and and2 (YandZ, Y, Z);
    not not1 (Y_n, Y);
    nor nor1 (XandY_nor_Y_n, XandY, Y_n);
    or or1(F, XandY_nor_Y_n, YandZ);
endmodule
```

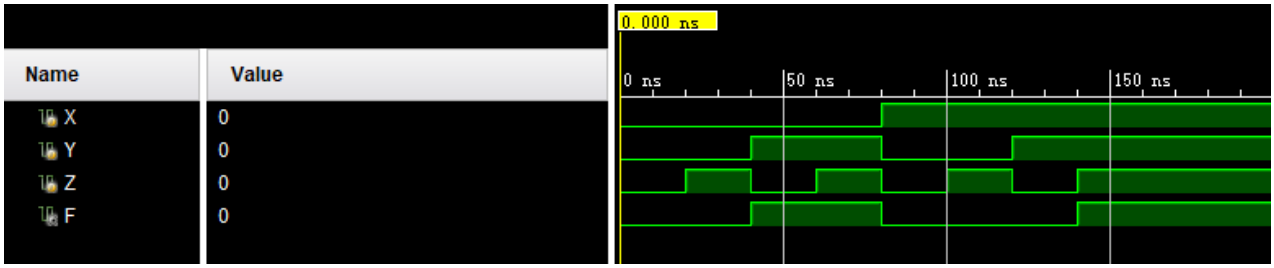
仿真代码：

```
module gate_circuit_TEST;
    reg X, Y, Z;
    wire F;
    gate_circuit gate_circuit_test(.X(X), .Y(Y), .Z(Z), .F(F));
        initial begin;
            X=0;Y=0;Z=0;#20;
            X=0;Y=0;Z=1;#20;
            X=0;Y=1;Z=0;#20;
            X=0;Y=1;Z=1;#20;
            X=1;Y=0;Z=0;#20;
            X=1;Y=0;Z=1;#20;
            X=1;Y=1;Z=0;#20;
            X=1;Y=1;Z=1;#20;
        end
endmodule;
```

RTL:



仿真图：



2.2.2 实验二 多路选择器

包括 5 位二选一选择器，32 位二选一选择器，32 位 4 选一选择器，模块命名格式和每个选择器的输入输出参数要求分别为：

- 5 位二选一选择器（MUX2X5） 模块参数列表：module MUX2X5 (A0,A1,S,Y);

设计代码：

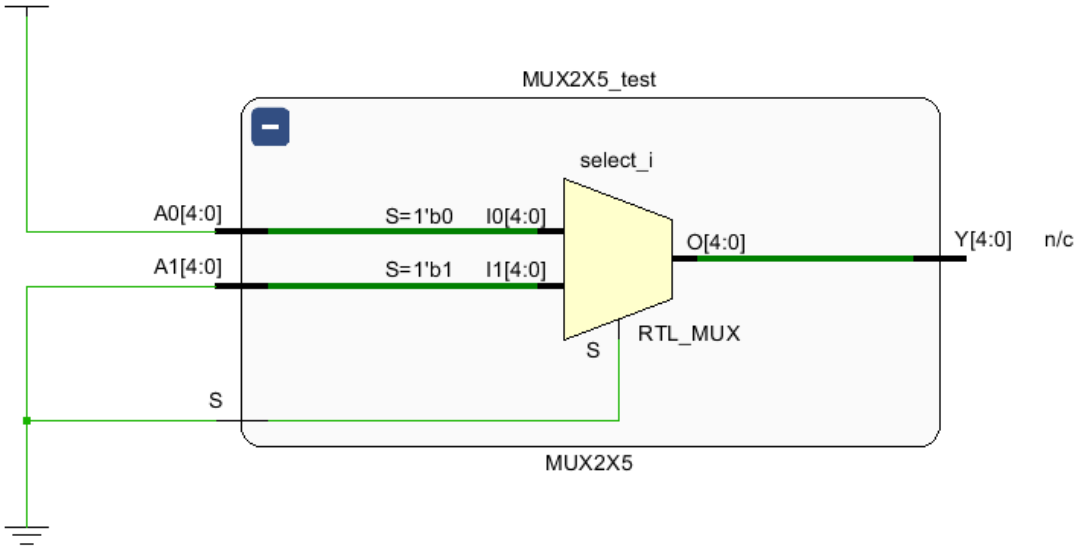
```
module MUX2X5(A0,A1,S,Y);
    input [4:0] A0,A1;
    input S;
    output [4:0] Y;
    function [4:0] select;
        input [4:0] A0,A1;
        input S;
            case (S)
                0:select = A0;
                1:select = A1;
            endcase
    endfunction
    assign Y = select(A0,A1,S);
endmodule
```

仿真代码：

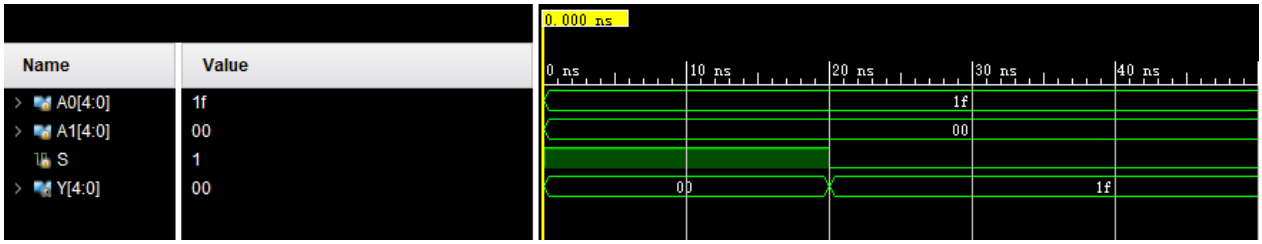
```
module MUX2X5_TEST;
    reg [4:0] A0,A1;
    reg S;
    wire [4:0] Y;
```

```
    MUX2X5 MUX2X5_test(.A0(A0), .A1(A1), .S(S), .Y(Y));
        initial begin;
          A0 = 'b11111;
          A1 = 'b00000;
          S = 1;
          #20;
      S = 0;
          end
endmodule
```

RTL



仿真图：



- 32 位二选一选择器（MUX2X32） 模块参数列表：module MUX2X32 (A0,A1,S,Y);

设计代码：

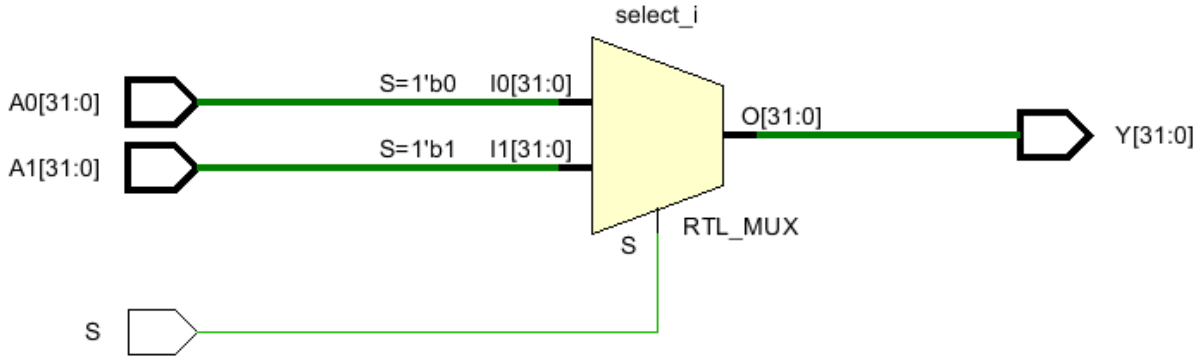
```
module MUX2X32(A0,A1,S,Y);
```

```
    input [31:0] A0,A1;
    input S;
    output [31:0] Y;
    function [31:0] select;
        input [31:0] A0,A1;
        input S;
            case (S)
                0:select = A0;
                1:select = A1;
            endcase
    endfunction
    assign Y = select(A0,A1,S);
endmodule
```

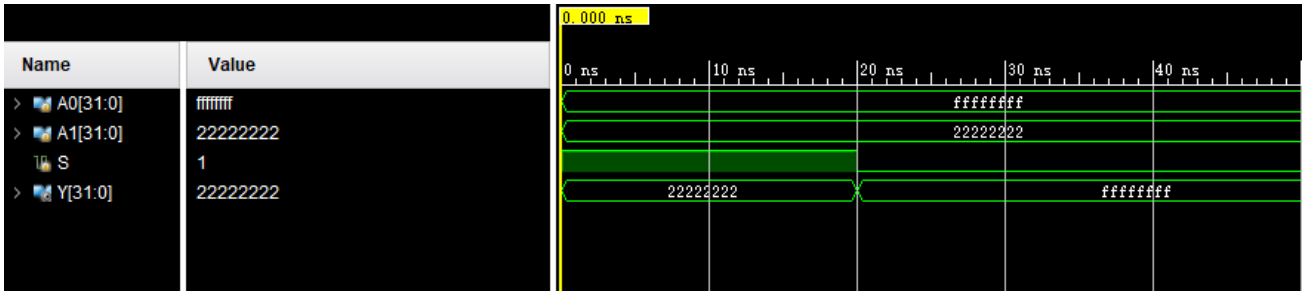
仿真代码：

```
module MUX2X32_TEST;
    reg [31:0] A0,A1;
    reg S;
    wire [31:0] Y;
    MUX2X32 MUX2X32_test(.A0(A0), .A1(A1), .S(S), .Y(Y));
        initial begin;
          A0 = 'b11111111111111111111111111111111;
          A1 = 'b00100010001000100010001000100010;
          S = 1;
          #20;
          S = 0;
        end
endmodule
```

RTL:



仿真图：



- 5 位四选一选择器（MUX4X32） 模块参数列表：module MUX4X32 (A0,A1,A2,A3,S,Y);

设计代码：

```
module MUX4X32(A0,A1,A2,A3,S,Y);
    input [31:0] A0,A1,A2,A3;
    input [1:0] S;
    output [31:0] Y;
    function[31:0] select;
        input [31:0] A0,A1,A2,A3;
        input [1:0]S;
            case(S)
                2'b00:select = A0;
```

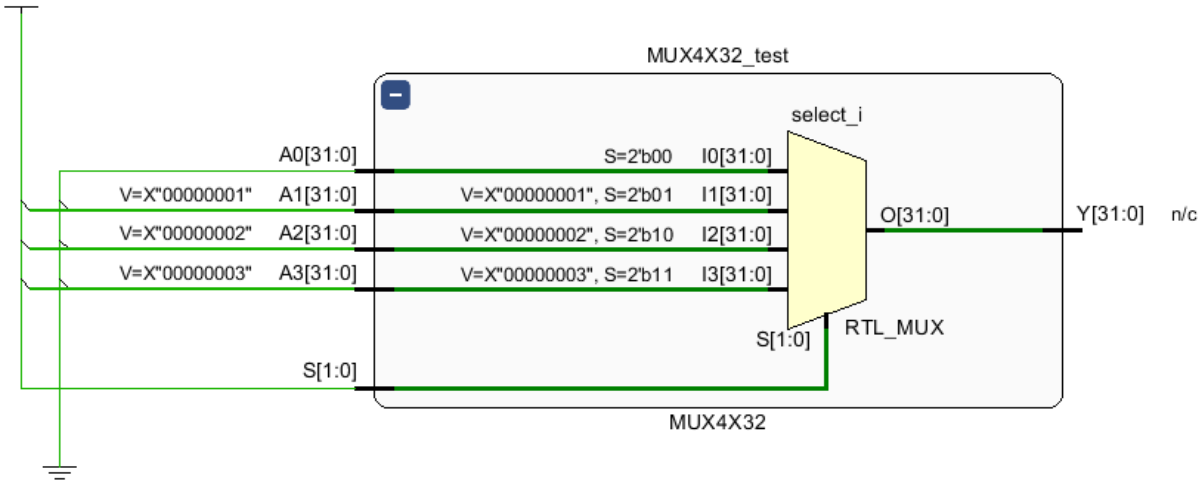
```
                2'b01:select = A1;
                2'b10:select = A2;
                2'b11:select = A3;
            endcase
    endfunction
    assign Y = select(A0,A1,A2,A3,S);
endmodule
```

仿真代码：

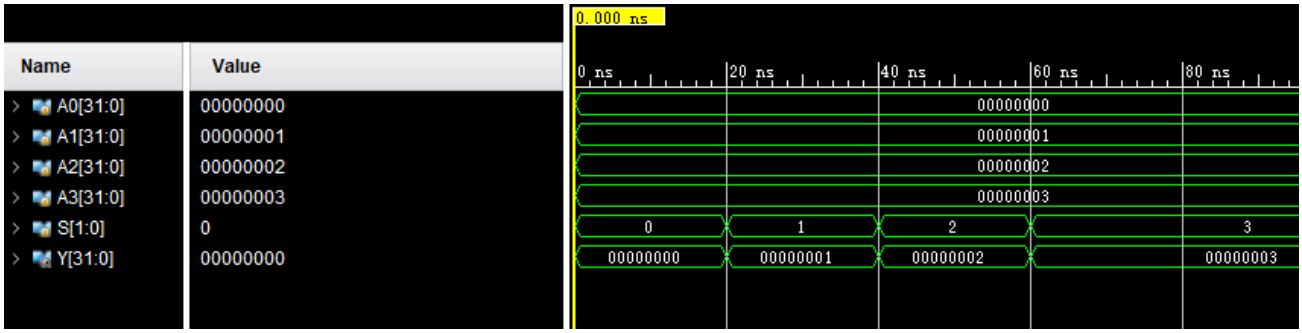
```
module MUX4X32_TEST;
    reg [31:0] A0, A1, A2, A3;
    reg [1:0] S;
    wire [31:0] Y;
    MUX4X32
MUX4X32_test( .A0(A0), .A1(A1), .A2(A2), .A3(A3), .S(S) , .Y(Y));
        initial begin
            A0 = 'b00000000000000000000000000000000;
            A1 = 'b00000000000000000000000000000001;
            A2 = 'b00000000000000000000000000000010;
            A3 = 'b00000000000000000000000000000011;
            S = 'b00;
            #20;
```

```
            S = 'b01;
            #20;
            S = 'b10;
            #20;
            S = 'b11;
        end
endmodule
```

RTL



仿真图：



提交实验报告要求：单个多路选择器的设计代码和仿真代码的源代码，RTL

图和仿真波形图，要求仿真波形图覆盖 S 的所有取值情况。

2.2.3 实验三 加减法器

实现 32 位加减法器的设计，要求能够实现是否加减法结果是否为 0 的判断。模块命名格式和每个选择器的输入输出参数要求分别为：

- 32 位加减法器（ADDSUB32） 模块参数列表：module ADDSUB32 (X, Y, Sub, S, Cout);

设计代码：

```
module ADDSUB_32(X, Y, Sub, S, Cout);
    input [31:0] X,Y;
    input Sub;
    output [31:0]S;
    output Cout;
    wire Cout1;
    CLA_32 adder0 (X, Y^{32{Sub}}, Sub, S, Cout);
endmodule
```

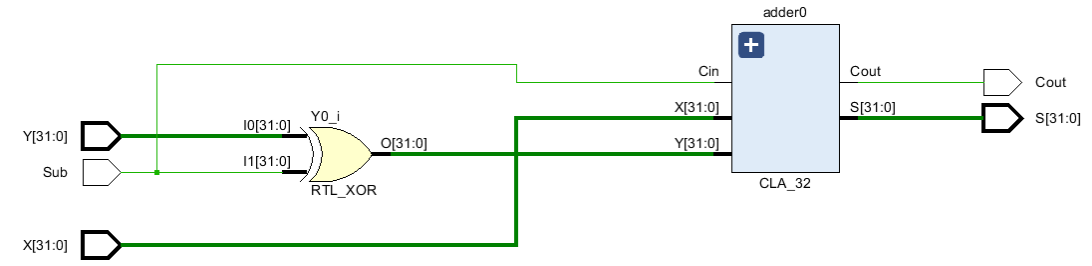
仿真代码：

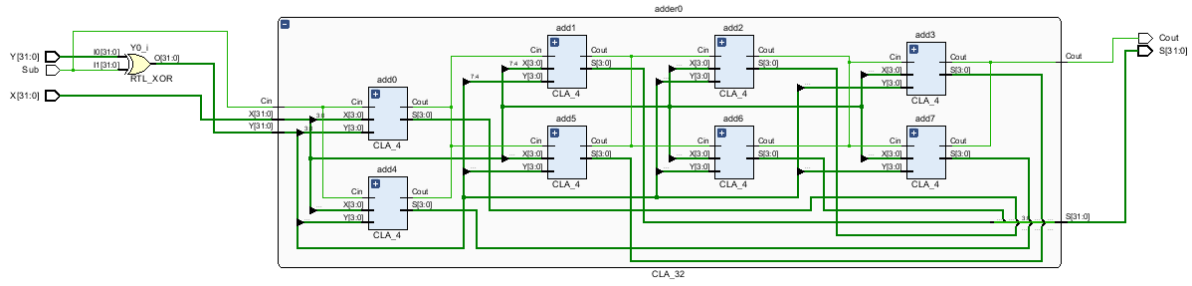
```
module ADDSUB_32_TEST;
reg [31:0] X,Y;
reg Sub;
wire [31:0]S;
wire Cout;
ADDSUB_32 addsubtest ( .X(X), .Y(Y), .Sub(Sub), .S(S), .Cout(Cout));
    initial begin
      X = 'b00000000000011110000000000001111;
      Y = 'b00000000000000110000000000000011;
      Sub = 1;
      #20;
      Sub = 0;
```

end

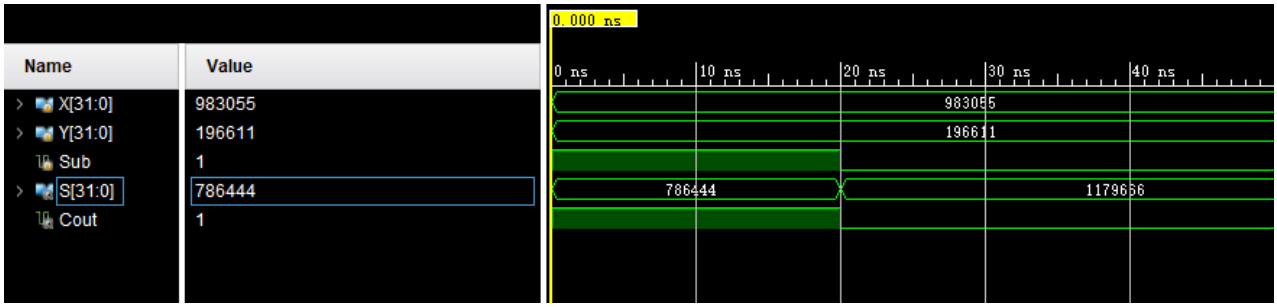
endmodule

RTL：





仿真图：



提交实验报告要求：32 位加减法的设计代码和仿真代码的源代码，RTL 图和仿真波形图。

2.2.4 实验四 移位器

实现 32 位移位器的设计，要求能够实现是算数和逻辑的左右移。模块命名格式和每个选择器的输入输出参数要求分别为：

- 32 移位器（SHIFTER32） 模块参数列表：module SHIFTER32 (X, Sa, Arith, Right, Sh);

Sa 代表移位位数，Arith 代表算术还是逻辑，默认 1 为算数，Right 代表左移还是右移，默认 1 为右移，Sh 为结果。

设计代码：

```
module SHIFTER_32(X,Sa,Arith,Right,Sh);
    input [31:0] X;
    input [4:0] Sa;
    input Arith,Right;
    output [31:0] Sh;
    wire [31:0] T4,S4,T3,S3,T2,S2,T1,S1,T0;
    wire a = X[31]&Arith;
    wire [15:0]e = {16{a}};
    parameter z = 16'b0;
    wire [31:0] L1u,L1d,L2u,L2d,L3u,L3d,L4u,L4d,L5u,L5d;

    assign L1u = {X[15:0],z};
    assign L1d = {e,X[31:16]};
    MUX2X32 M1l(L1u,L1d,Right,T4);
    MUX2X32 M1D(X,T4,Sa[4],S4);

    assign L2u = {S4[23:0],z[7:0]};
    assign L2d = {e[7:0],S4[31:8]};
    MUX2X32 M2l(L2u,L2d,Right,T3);
    MUX2X32 M2D(S4,T3,Sa[3],S3);

    assign L3u = {S3[27:0],z[3:0]};
    assign L3d = {e[3:0],S3[31:4]};
    MUX2X32 M3l(L3u,L3d,Right,T2);
    MUX2X32 M3D(S3,T2,Sa[2],S2);

    assign L4u = {S2[29:0],z[1:0]};
```

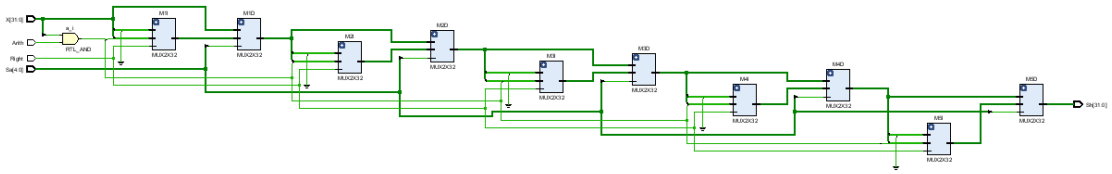
```
        assign L4d = {e[1:0],S2[31:2]};
        MUX2X32 M4l(L4u,L4d,Right,T1);
        MUX2X32 M4D(S2,T1,Sa[1],S1);

        assign L5u = {S1[30:0],z[0]};
        assign L5d = {e[0],S1[31:1]};
        MUX2X32 M5l(L5u,L5d,Right,T0);
        MUX2X32 M5D(S1,T0,Sa[0],Sh);
Endmodule
```

仿真代码：

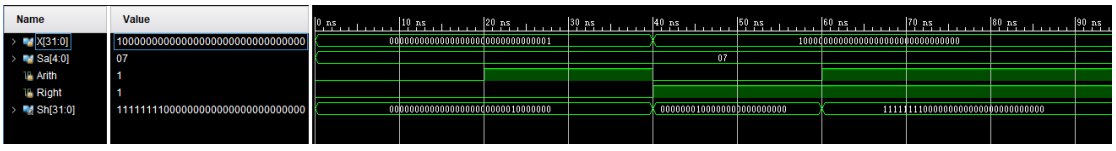
```
    module SHIFTER_32_TEST;
    reg [31:0] X;
    reg [4:0] Sa;
    reg Arith,Right;
    wire [31:0] Sh;
    SHIFTER_32 shifter_test( .X(X), .Sa(Sa), .Arith(Arith), .Right(Right), .Sh(Sh));
        initial begin
          X = 'b00000000000000000000000000000001;
          Sa = 'b00111;
          Arith = 0;
          Right = 0;

          #20;

          X = 'b00000000000000000000000000000001;
          Sa = 'b00111;
          Arith = 1;
          Right = 0;
```

```
        #20;

        X = 'b10000000000000000000000000000000;
        Sa = 'b00111;
        Arith = 0;
        Right = 1;

        #20;

        X = 'b10000000000000000000000000000000;
        Sa = 'b00111;
        Arith = 1;
        Right = 1;

    end
endmodule
```

RTL：



仿真图：



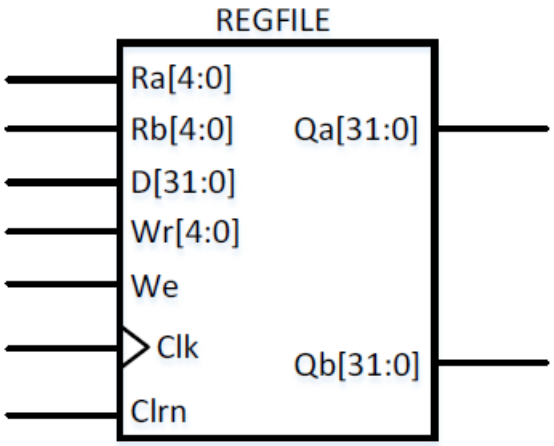
提交实验报告要求：32 移位器的设计代码和仿真代码的源代码，RTL 图和仿真波形图。

2.2.5 实验五 寄存器堆

实现对寄存器堆的设计，要求有写使能信号，有 32 个寄存器。模块命名格

式和每个选择器的输入输出参数要求分别为：

- 寄存器堆（REGFILE） 模块参数列表：module REGFILE (Ra, Rb, D, Wr, We, Clk, Clrn, Qa, Qb);通用寄存器堆的接口模型如下所示



设计代码：

```
module REGFILE(Ra, Rb, D, Wr, We, Clk, Clrn, Qa, Qb);
    input [4:0] Ra, Rb, Wr;
    input [31:0] D;
    input We, Clk, Clrn;
    output [31:0] Qa, Qb;
    wire [31:0] Ymux,
Q0_re32,Q1_re32,Q2_re32,Q3_re32,Q4_re32,Q5_re32,Q6_re32,Q7_re32,
Q8_re32,Q9_re32,Q10_re32,Q11_re32,Q12_re32,Q13_re32,Q14_re32,Q15_re32,
Q16_re32,Q17_re32,Q18_re32,Q19_re32,Q20_re32,Q21_re32,Q22_re32,Q23_re32,
Q24_re32,Q25_re32,Q26_re32,Q27_re32,Q28_re32,Q29_re32,Q30_re32,Q31_re32;
    DEC5T32E dec(Wr, We, Y_mux);
    REG32(D, Y_mux, Clk, Clrn,

Q31_re32,Q30_re32,Q29_re32,Q28_re32,Q27_re32,Q26_re32,Q25_re32,Q24_re32,

Q23_re32,Q22_re32,Q21_re32,Q20_re32,Q19_re32,Q18_re32,Q17_re32,Q16_re32,
    Q15_re32,Q14_re32,Q13_re32,Q12_re32,Q11_re32,Q10_re32,Q9_re32,Q8_re32,
    Q7_re32,Q6_re32,Q5_re32,Q4_re32,Q3_re32,Q2_re32,Q1_re32,Q0_re32
    );
    MUX32X32 select1 (
```

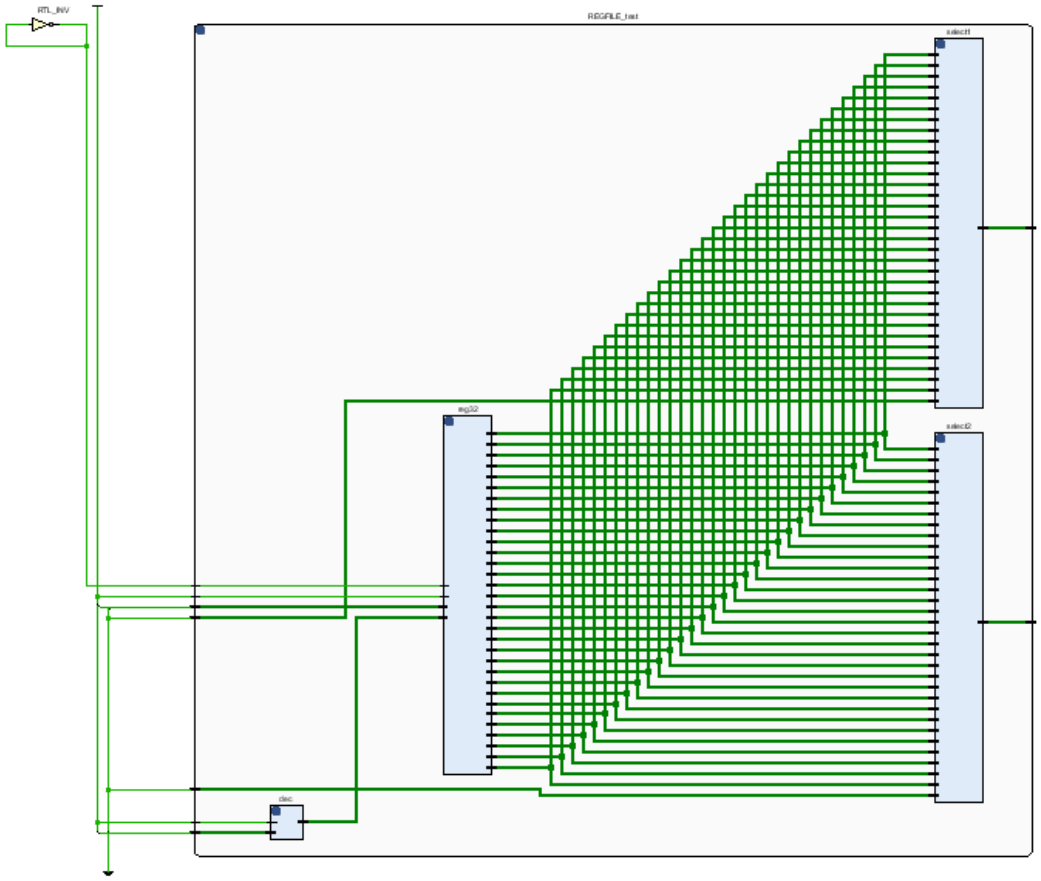
```
        Q0_re32,Q1_re32,Q2_re32,Q3_re32,Q4_re32,Q5_re32,Q6_re32,Q7_re32,
        Q8_re32,Q9_re32,Q10_re32,Q11_re32,Q12_re32,Q13_re32,Q14_re32,Q15_re32,

Q16_re32,Q17_re32,Q18_re32,Q19_re32,Q20_re32,Q21_re32,Q22_re32,Q23_re32,

Q24_re32,Q25_re32,Q26_re32,Q27_re32,Q28_re32,Q29_re32,Q30_re32,Q31_re32,
        Ra,Qa
        );
        MUX32X32 select2 (
        Q0_re32,Q1_re32,Q2_re32,Q3_re32,Q4_re32,Q5_re32,Q6_re32,Q7_re32,
        Q8_re32,Q9_re32,Q10_re32,Q11_re32,Q12_re32,Q13_re32,Q14_re32,Q15_re32,

Q16_re32,Q17_re32,Q18_re32,Q19_re32,Q20_re32,Q21_re32,Q22_re32,Q23_re32,

Q24_re32,Q25_re32,Q26_re32,Q27_re32,Q28_re32,Q29_re32,Q30_re32,Q31_re32,
        Rb,Qb
        );
    Endmodule
```

仿真代码：

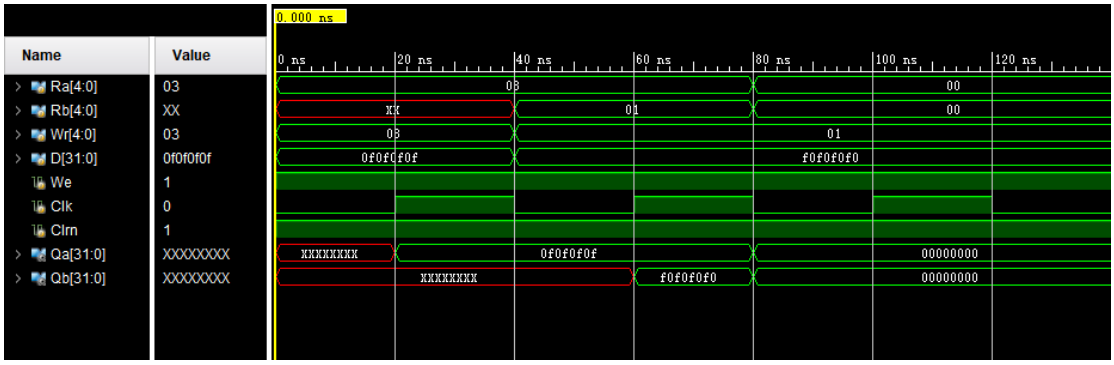
RTL：

```
module REGFILE_TEST;
    reg [4:0] Ra, Rb, Wr;
    reg [31:0] D;
    reg We, Clk, Clrn;
    wire [31:0] Qa, Qb;
    always #20 Clk = ~Clk;
    REGFILE
REGFILE_test(.Ra(Ra), .Rb(Rb), .D(D), .Wr(Wr), .We(We), .Clk(Clk), .Clrn(Clr
n), .Qa(Qa), .Qb(Qb));
    initial begin;
        Clk = 0;
        Clrn = 1;
```

```
        We = 1;
        Ra = 'b00011;
        Wr = 'b00011;
        D = 'b00001111000011110000111100001111;

        #40;
        Wr = 'b00001;
        Rb = 'b00001;
        D = 'b11110000111100001111000011110000;

        #40;
        Ra = 'b00000;
        Rb = 'b00000;
    end
endmodule
```



仿真图：

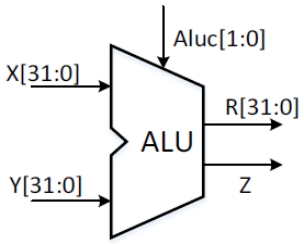


提交实验报告要求： 通用寄存器堆的设计代码和仿真代码的源代码，RTL 图和仿真波形图。

2.2.6 实验六 ALU 封装

完成 ALU 的封装，要求 ALU 能够进行加减和与或四种运算，并且能够实现是否为 0 的判断。模块命名格式和每个选择器的输入输出参数要求分别为：

- 寄存器堆（ALU） 模块参数列表：module ALU (X, Y, Aluc, R, Z);ALU 的接口模型如下所示



其中 Aluc 代表选择的功能，真值表如下

Aluc编码	实现功能	运算类型
00	加	算术运算
01	减	
10	按位与	逻辑运算
11	按位或	

提交实验报告要求： ALU 的设计代码和仿真代码的源代码，RTL 图和仿真波形图。

设计代码：

```
module ALU(X,Y,Aluc,R,Z,V);
```

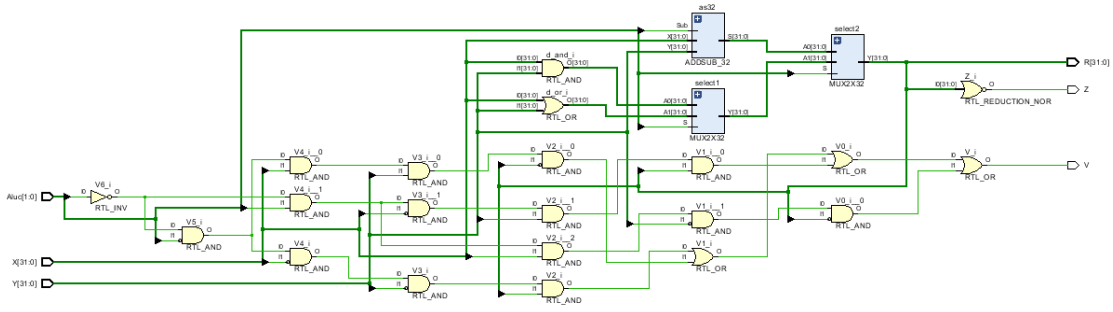
```
        input [31:0] X,Y;
        input [1:0] Aluc;
        output [31:0] R;
        output Z,V;
        wire   [31:0] d_as, d_and, d_or, d_and_or;
        ADDSUB_32 as32(X, Y, Aluc[0], d_as);
        assign d_and = X&Y;
        assign d_or = X|Y;
        MUX2X32 select1(d_and, d_or, Aluc[0], d_and_or);
        MUX2X32 select2(d_as, d_and_or, Aluc[1], R);
        assign Z = ~|R;
        assign V =
(~Aluc[1]&~Aluc[0]&~X[31]&~Y[31]&R[31])|(~Aluc[0]&~Aluc[1]&X[31]&Y[31]&~R[31
])|(~Aluc[1]&Aluc[0]&~X[31]&Y[31]&R[31])|(~Aluc[1]&Aluc[0]&X[31]&~Y[31]&~R[31]
);
    endmodule
```

仿真代码：

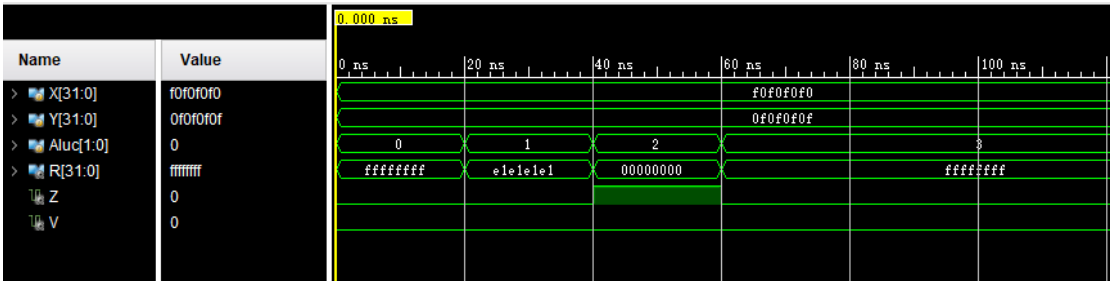
```
    module ALU_TEST;
        reg [31:0]X,Y;
        reg [1:0]Aluc;
        wire[31:0] R;
        wire Z,V;
        ALU ALU_test(.X(X),.Y(Y),.Aluc(Aluc),.R(R),.Z(Z),.V(V));
            initial begin;
```

```
        X = 'b11110000111100001111000011110000;
        Y = 'b00001111000011110000111100001111;
        Aluc = 'b00;

        #20;

        Aluc = 'b01;

        #20;

        Aluc = 'b10;

        #20;
        Aluc = 'b11;
    end
endmodule
```

RTL 图：



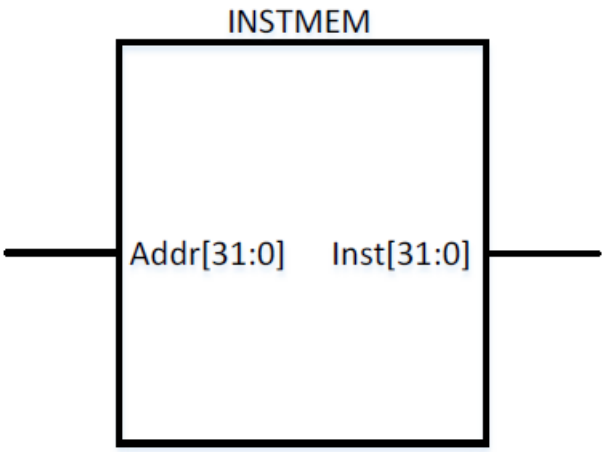
仿真图：



2.2.7 实验七 存储器

完成指令存储器的封装和数据存储器的封装。模块命名格式和每个选择器的输入输出参数要求分别为：

- 指令存储器（INSTMEM） 模块参数列表：module INSTMEM (Addr, Inst);指令存储器的接口模型如下所示



指令存储器要求可存储 32 条指令，其具体存储方式为：将输入的 32 位地址信号 Addr 的第 6 位至第 2 位的 5 位地址字段映射到 INSTMEM 模块内部定义的 Rom[31:0] 的 32 个指令编码之一进行持续输出。注意，在模块 INSTMEM 给出的代码中，这 32 条指令中的每条指令编码都使用符号 X 来暂时占用位数，而在实际使用过程中应使用指令的实际编码进行替换。

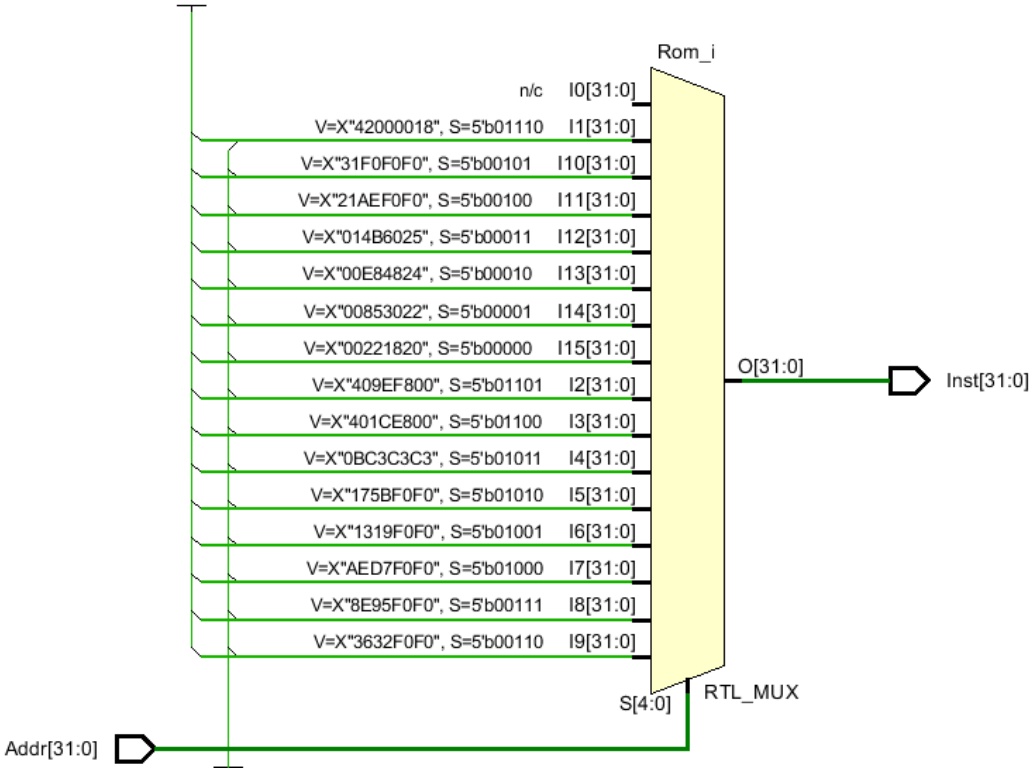
设计代码：

```
module INSTMEM(Addr, Inst);

    input [31:0] Addr;

    output [31:0] Inst;

    wire [31:0] Rom [31:0];

    assign Rom[5'h00] = 32'b100011_00000_00001_00000_000000_01100;//取出数放在$1=3
```

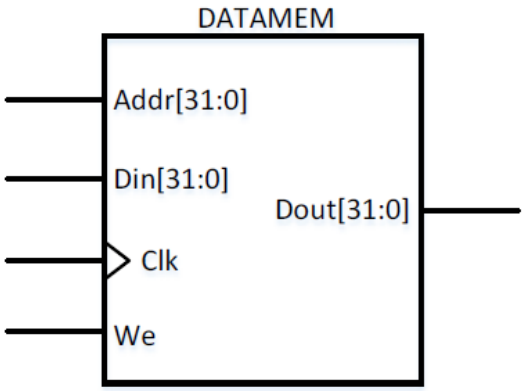
```
    assign Rom[5'h01] = 32'b100011_00000_00010_00000_000000_01000;//取出数放在$2=2
    assign Rom[5'h02] = 32'b000000_00001_00010_00011_000001_00000;//$3=$1+$2=5
    assign Rom[5'h03] = 32'b000000_00001_00010_00100_000001_00010;//$4=$1-$2=1
    assign Rom[5'h04] = 32'b000000_00011_00010_00101_000001_00100;//$5=$2&$3=0
    assign Rom[5'h05] = 32'b000000_00011_00010_00110_000001_00101;//$6=$2|$3=7
    assign Rom[5'h06] = 32'b001100_00011_00111_0000000000000010;//$7=$3&2=0
    assign Rom[5'h07] = 32'b001101_00011_01000_0000000000000010;//$8=$3|2=7
    assign Rom[5'h08] = 32'b001000_00110_01001_0000000000000010;//$9=$6+2=9
    assign Rom[5'h09] = 32'b100011_01001_01010_0000000000001000;//取出数放在$10=$9+1000=100.01=4
    assign Rom[5'h0A] = 32'b000100_00110_01000_0000000000000001;//$6、$8 相等则跳转到下面第二条
    assign Rom[5'h0B] = 32'b100011_00000_01011_00000_000000_00100;//取出数放在$11=1
    assign Rom[5'h0C] = 32'b100011_00000_01100_00000_000000_00100;//取出数放在$12=1
    assign Rom[5'h0D] = 32'b000101_00010_00001_0000000000000001;//$1、$2 不相等则跳转到下面第二条
    assign Rom[5'h0E] = 32'b100011_00000_01101_00000_000000_00100;//取出数放在$13=1
    assign Rom[5'h0F] = 32'b100011_00000_01110_00000_000000_00100;//取出数放在$14=1
    assign Rom[5'h10] = 32'b000010_00000_00000_00000_000000_01110;//跳转到上数第二条指令;
//      assign Rom[5'h11] = 32'hXXXXXXXX;
//      assign Rom[5'h12] = 32'hXXXXXXXX;
//      assign Rom[5'h14] = 32'hXXXXXXXX;
//      assign Rom[5'h15] = 32'hXXXXXXXX;
//      assign Rom[5'h16] = 32'hXXXXXXXX;
//      assign Rom[5'h17] = 32'hXXXXXXXX;
//      assign Rom[5'h18] = 32'hXXXXXXXX;
//      assign Rom[5'h19] = 32'hXXXXXXXX;
//      assign Rom[5'h1A] = 32'hXXXXXXXX;
//      assign Rom[5'h1B] = 32'hXXXXXXXX;
//      assign Rom[5'h1C] = 32'hXXXXXXXX;
//      assign Rom[5'h1D] = 32'hXXXXXXXX;
//      assign Rom[5'h1E] = 32'hXXXXXXXX;
//      assign Rom[5'h1F] = 32'hXXXXXXXX;
```

```
    assign Inst = Rom [Addr[6:2]];

endmodule
```

RTL:



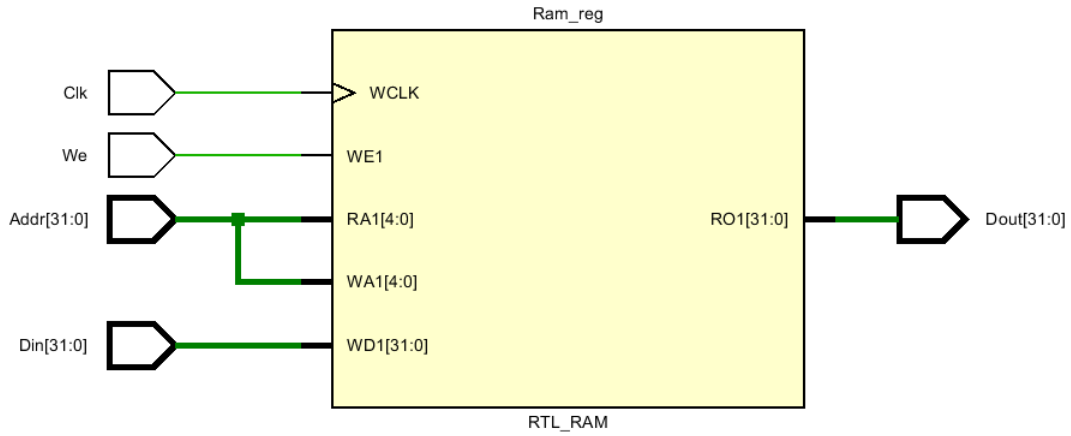
- 数据存储器（DATAMEM）模块参数列表：module DATAMEM (Addr, Din, Clk, We, Dout);
  数据存储器的接口模型如下所示



设计代码：

```
module DATAMEM(Addr, Din, Clk, We, Dout);

    input [31:0] Addr, Din;

    input Clk, We;
```

```
    output [31:0] Dout;
    reg [31:0] Ram [31:0];
    assign Dout = Ram [Addr[6:2]];
    always @(posedge Clk) begin
        if(We) Ram[Addr[6:2]] <= Din;
    end
    integer i;
    initial begin
        for (i = 0;i < 32 ; i = i + 1)
            Ram[i] = i;
    end
endmoduleRTL:
```



提交实验报告要求： ALU 的设计代码和 RTL 图。

2.2.8 实验八 支持简单异常和中断处理的单周期 CPU

要求实现一个简单的带有一个外部中断请求和一种异常情况（ALU 运算结果溢出）的 CPU。要求实现 15 种指令，指令及其机器指令如下表所示。

指令	[31:26]	[25:21]	[20:16]	[15:11]	[10: 6]	[5:0]	功能
add	000000	rs	Rt	rd	00000	100000	寄存器加
sub	000000	rs	Rt	rd	00000	100010	寄存器减
and	000000	rs	Rt	rd	00000	100100	寄存器与
or	000000	rs	Rt	rd	00000	100101	寄存器或
addi	001000	rs	rt	immediate			立即数加

andi	001100	rs	rt	immediate			立即数与
ori	001101	rs	rt	immediate			立即数或
lw	100011	rs	rt	offset			取数（字）
sw	101011	rs	rt	offset			存数（字）
beq	000100	rs	rt	offset			相等转移
bne	000101	rs	rt	offset			不等转移
j	000010	address					跳转
mfc0	010000	00000	Rt	rd	00000	000000	mfc0 rt, rd
mtc0	010000	00100	Rt	rd	00000	000000	mtc0 rt, rd
eret	010000	10000	00000	00000	00000	011000	Eret

CPU使用指令mfc0 rt, rd（其中rt是通用寄存器、rd是寄存器Cause或Status或EPC）从寄存器rd中读取值到寄存器rt中；使用指令mtc0 rt, rd（其中rt是通用寄存器、rd是寄存器Cause或Status或EPC）从寄存器rt中读取值到寄存器rd中；使用指令eret从异常/中断服务程序返回。

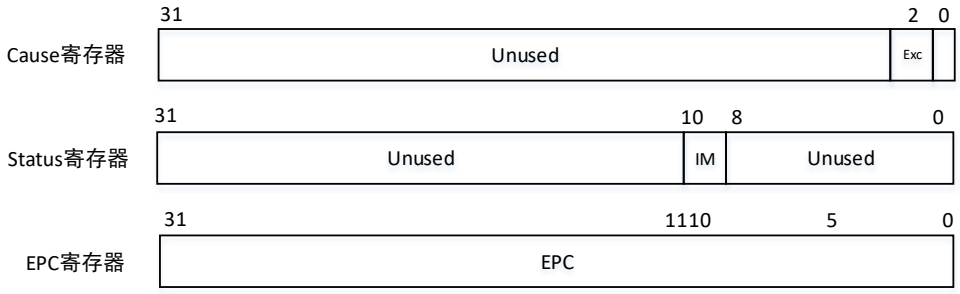
a) 要求由Status寄存器控制异常和中断是打开还是关闭；

b) 能把引起异常或中断的原因自动记录到Cause寄存器中；

c) 响应异常时自动保存当前指令的地址，响应中断时自动保存下一条指令的地址，并把一个固定地址（异常和中断处理程序的入口地址）写入PC寄存器，然后通过指令去读取Cause寄存器；

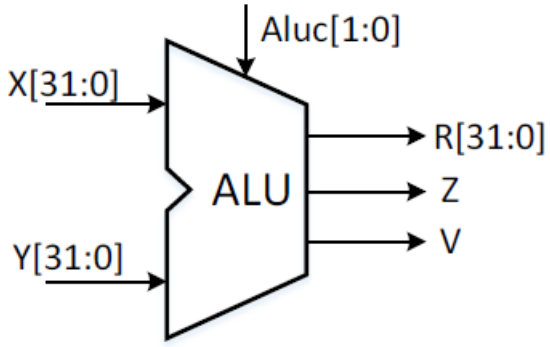
d) 如果是外部中断请求，若中断未关闭，CPU还需发出中断确认信号。

三个寄存器的定义如下：



Cause寄存器的Exc字段为第2位，如果为0，表示发生的是外部中断，否则如果为1，表示发生的是ALU计算溢出，Status寄存器的IM字段占第10-9位，其中第10位和第9位为1分别表示允许中断和异常，否则若为0，分别表示禁止中断和异常。

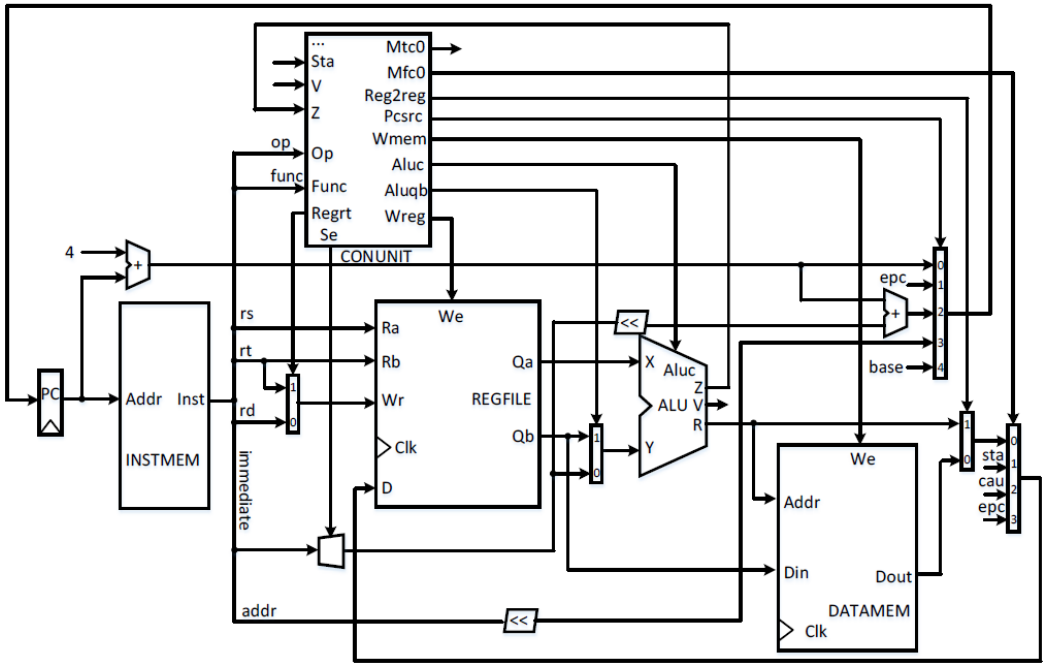
同时，为判断是否溢出，需对ALU做相应修改，使之能够输出是否溢出信号V，ALU接口图如下所示。



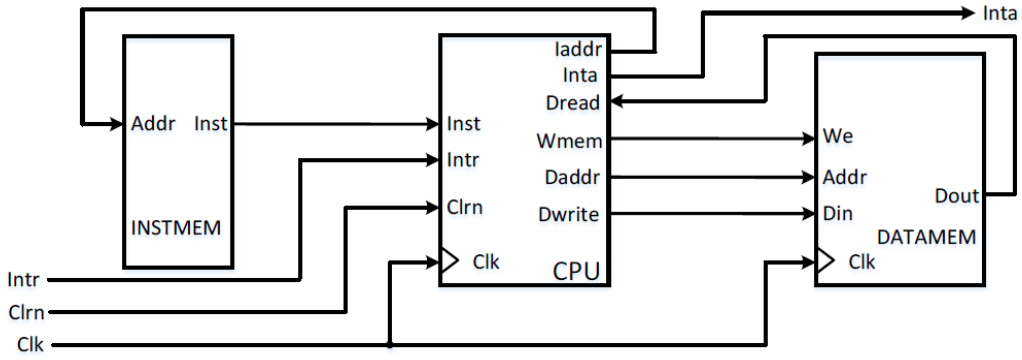
溢出条件真值表如下所示。

运算	Aluc	X[31]	Y[31]	R[31]	V
加法	00	0	0	1	1
加法	00	1	1	0	1
减法	01	0	1	1	1
减法	01	1	0	0	1

最终实验的电路图类似于下图，具体参照书上带有简单中断那一章ppt实现。



要求最后封装成如下形式



模块命名为CPU 模块输入输出参数为Module CPU（Inst,Intr,Clrn,Clk,Iaddr,Inta,Dread,Wmem,Daddr,Dwrite）；

要求能在CPU的输出端口输出结果并仿真。

提交实验报告要求： CPU 的设计代码，仿真代码，仿真波形图和 RTL 图。

注：CPU_new 为流水线 CPU，CPU2 为流水线 CPU 内模块，CPU_simple 为简单 CPU，CPU1 位简单 CPU 内模块

CPU 设计代码

```
module CPU1(Clk, Clrn, Inst, Dread, Iaddr, Daddr, Dwrite, Wmem);
    input Clk,Clrn;
    input [31:0] Inst, Dread;
    output [31:0] Iaddr, Daddr, Dwrite;
    output Wmem;

    wire [31:0] Q,Qn;
    wire [31:0]D = MUX4X32_Y;
    parameter En = 1;
//  D_FFEC32 PC (D,Clk,En,Clrn,Q,Qn);
    PC pc(Clk, En, D,Q);

    wire [31:0]CLA_32_X = Q;
    wire [31:0]CLA_32_S;
    CLA_32_plus4 cla_32_plus4(CLA_32_X,CLA_32_S);

    assign Iaddr = Q;
//    wire [31:0] Addr = Q;
//    wire [31:0] Inst;
```

```
//      INSTMEM instmem(Addr, Inst);

        wire [15:0]EXT_X = Inst[15:0];
        wire EXT_Se = Se;
        wire [31:0]EXT_Y;
        EXT16T32 ext16to32(EXT_X,EXT_Se,EXT_Y);

        wire [31:0]Sh1;
        SHIFTER32_L2 shift32_L2(EXT_Y,Sh1);

        parameter Sub = 'b0;
        wire [31:0]ADDSUB_S;
        wire Cout = 0;
        ADDSUB_32 addsub_32(Sh1, CLA_32_S, Sub, ADDSUB_S, Cout);

        wire [31:0]addr = {Q[31:28],Inst[25:0]};
        wire [31:0] Sh2;
        SHIFTER32_L2 shift32_L2_1(addr,Sh2);

        wire [31:0]MUX4X32_Y;
        wire [1:0]Pcsrc;
        wire A1;
        MUX4X32 mux4x32(CLA_32_S,A1,ADDSUB_S,Sh2,Pcsrc,MUX4X32_Y);

        wire [5:0]Op = Inst[31:26];
        wire [5:0]Func = Inst [5:0];
        wire [1:0]    Aluc;
        wire Wreg,Se,Aluqb,Regrt, Reg2reg;
        wire ALU_Z;
        CONUNIT conunit(Op, Func, ALU_Z ,Regrt, Se, Wreg, Aluqb, Aluc, Wmem,
Pcsrc, Reg2reg);

        wire [4:0]rt = Inst[20:16];
        wire [4:0]rd = Inst[15:11];
```

```
    wire [4:0]INS_REG_Y;
    MUX2X5 INS_REG(rd, rt, Regrt, INS_REG_Y);

    wire [4:0]rs = Inst[25:21];
    wire [4:0]Ra = rs;
    wire [4:0]Rb = rt;
    wire [4:0]Wr = INS_REG_Y;
    wire We = Wreg;
    wire [31:0]REG_D = ALU_DATAMEM_Y;
    wire [31:0] Qa, Qb;
    REGFILE regfile(Ra, Rb, REG_D, Wr, We, Clk, Clrn, Qa, Qb);

    wire [31:0]REG_ALU_Y;
    MUX2X32 REG_ALU(EXT_Y, Qb, Aluqb, REG_ALU_Y);

    wire [31:0]ALU_X = Qa;
    wire [31:0]ALU_Y = REG_ALU_Y;
    wire [1:0]ALU_Aluc = Aluc;
    wire [31:0]ALU_R;
    ALU alu(ALU_X,ALU_Y,ALU_Aluc,ALU_R,ALU_Z);

    assign Dwrite = Qb;
    assign Daddr = ALU_R;
    wire [31:0]ALU_DATAMEM_Y;
    MUX2X32 ALU_DATAMEM(Dread, ALU_R, Reg2reg, ALU_DATAMEM_Y);

Endmodule
```

CPU 封装代码

```
module CPU_simple(Clk, Clrn);
    input Clk,Clrn;
    wire [31:0] Inst, Dread;
    wire [31:0] Iaddr, Daddr, Dwrite;
    wire Wmem;
```

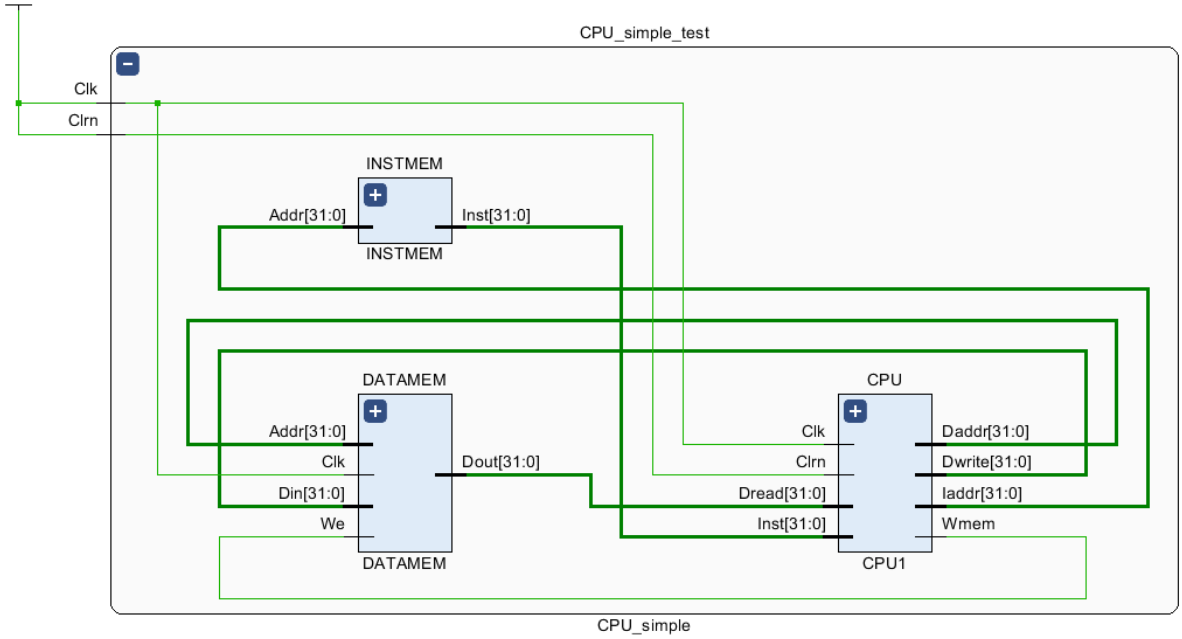
CPU1 CPU(Clk, Clrn, Inst, Dread, Iaddr, Daddr, Dwrite, Wmem);

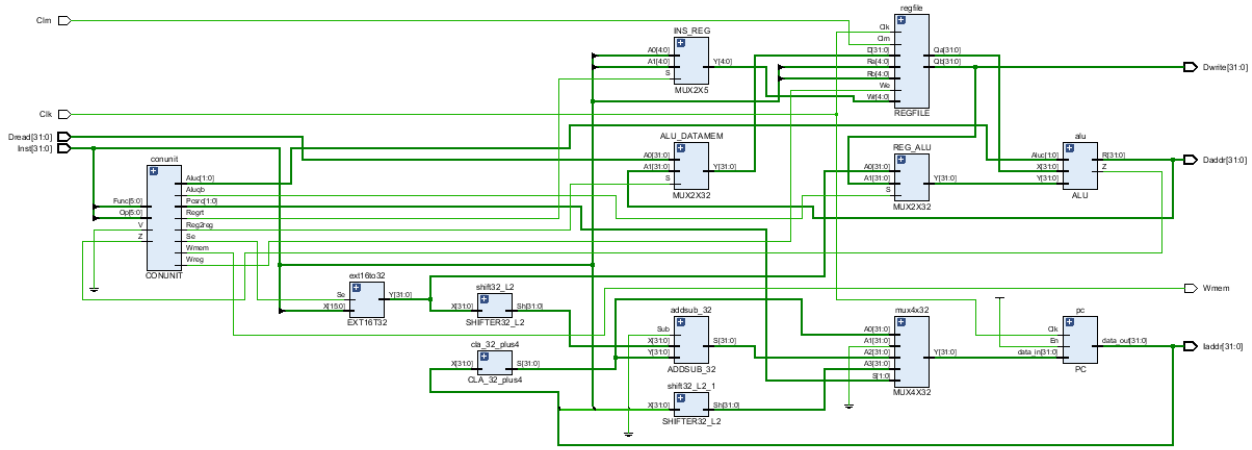
DATAMEM DATAMEM(Daddr, Dwrite, Clk, Wmem, Dread);

INSTMEM INSTMEM(Iaddr, Inst);

Endmodule

RLT 图





CPU 仿真代码

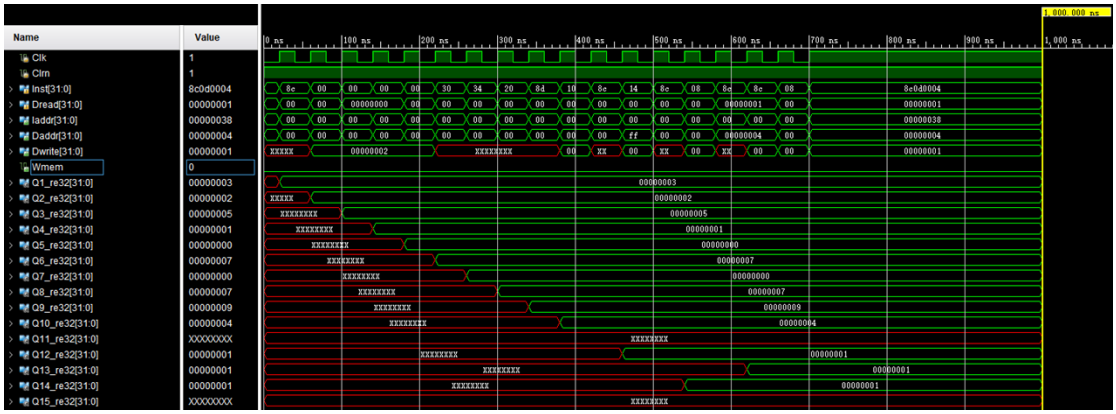
此处不用 always 的原因是防止后续无效命令覆盖了前面寄存器的值

```
module CPU_simple_TEST;
    reg Clk,Clrn;
    CPU_simple CPU_simple_test(.Clk(Clk),.Clrn(Clrn));
        initial begin;
            Clk = 0;
            Clrn = 1;
            #20;
            Clk =~Clk;
            ……
        end
endmodule
```

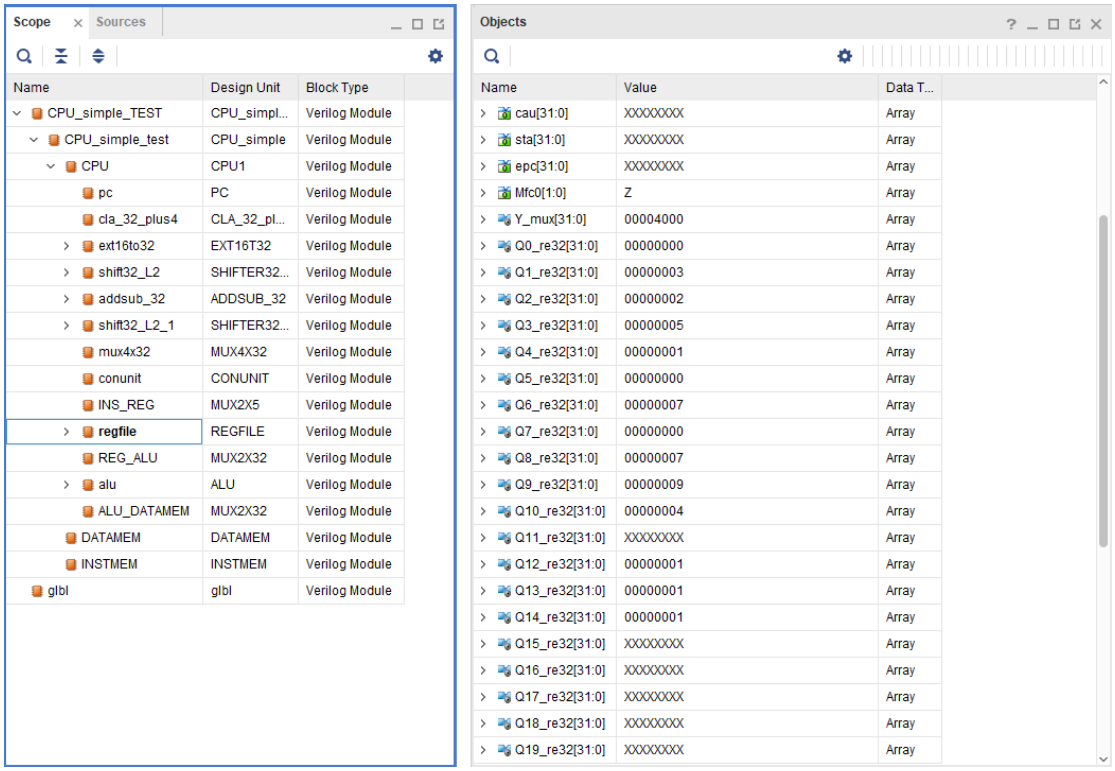
仿真图

指令：

```
assign Rom[5'h00] = 32'b100011_00000_00001_00000_000000_01100;//取出数放在$1=3
assign Rom[5'h01] = 32'b100011_00000_00010_00000_000000_01000;//取出数放在$2=2
assign Rom[5'h02] = 32'b000000_00001_00010_00011_000001_00000;//$3=$1+$2=5
assign Rom[5'h03] = 32'b000000_00001_00010_00100_000001_00010;//$4=$1-$2=1
assign Rom[5'h04] = 32'b000000_00011_00010_00101_000001_00100;//$5=$2&$3=0
assign Rom[5'h05] = 32'b000000_00011_00010_00110_000001_00101;//$6=$2|$3=7
assign Rom[5'h06] = 32'b001100_00011_00111_0000000000000010;//$7=$3&2=0
assign Rom[5'h07] = 32'b001101_00011_01000_0000000000000010;//$8=$3|2=7
assign Rom[5'h08] = 32'b001000_00110_01001_0000000000000010;//$9=$6+2=9
assign Rom[5'h09] = 32'b100011_01001_01010_0000000000001000;//取出数放在$10=$9+1000=100.01=4
assign Rom[5'h0A] = 32'b000100_00110_01000_0000000000000001;//$6、$8 相等则跳转到下面第二条
assign Rom[5'h0B] = 32'b100011_00000_01011_00000_000000_00100;//取出数放在$11=1
assign Rom[5'h0C] = 32'b100011_00000_01100_00000_000000_00100;//取出数放在$12=1
assign Rom[5'h0D] = 32'b000101_00010_00001_0000000000000001;//$1、$2 不相等则跳转到下面第二条
assign Rom[5'h0E] = 32'b100011_00000_01101_00000_000000_00100;//取出数放在$13=1
assign Rom[5'h0F] = 32'b100011_00000_01110_00000_000000_00100;//取出数放在$14=1
assign Rom[5'h10] = 32'b000010_00000_00000_00000_000000_01110;//跳转到上数第二条指令;
```



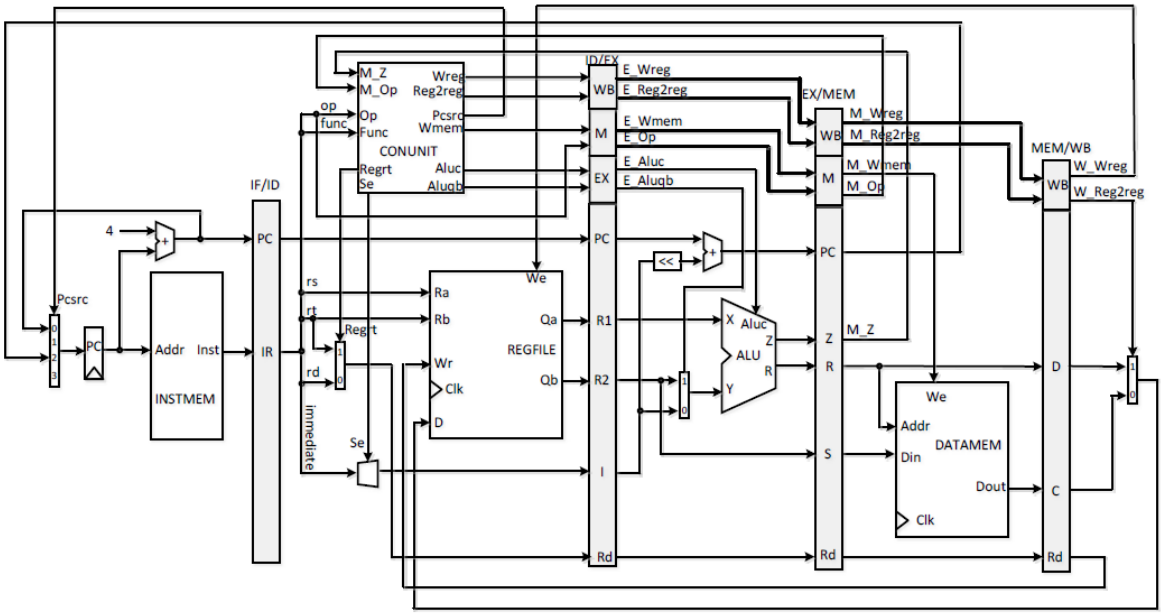
寄存器堆图



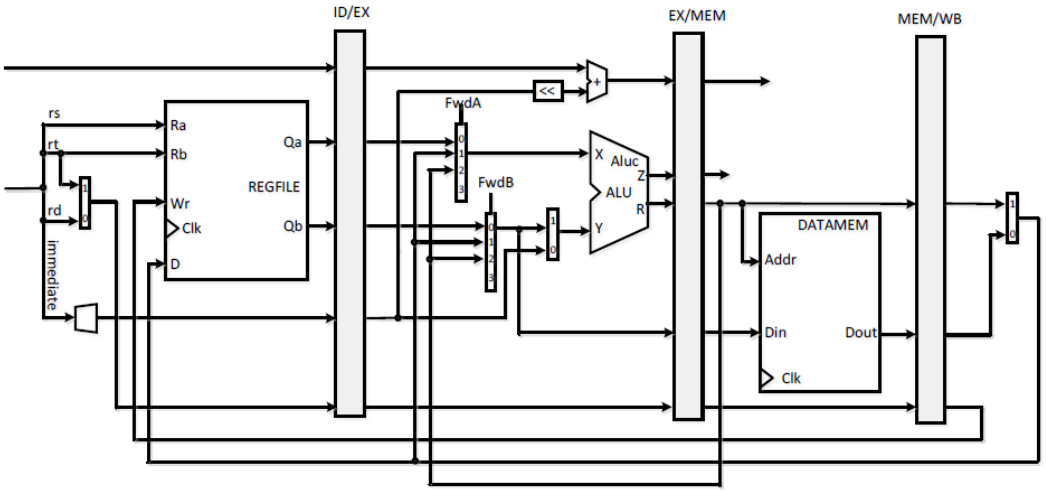
2.2.9 实验九 解决数据冲突的流水线的 CPU

实现解决数据冲突的流水线的CPU，实现的指令为除去与上一实验关于中断和异常的指令剩下的指令。

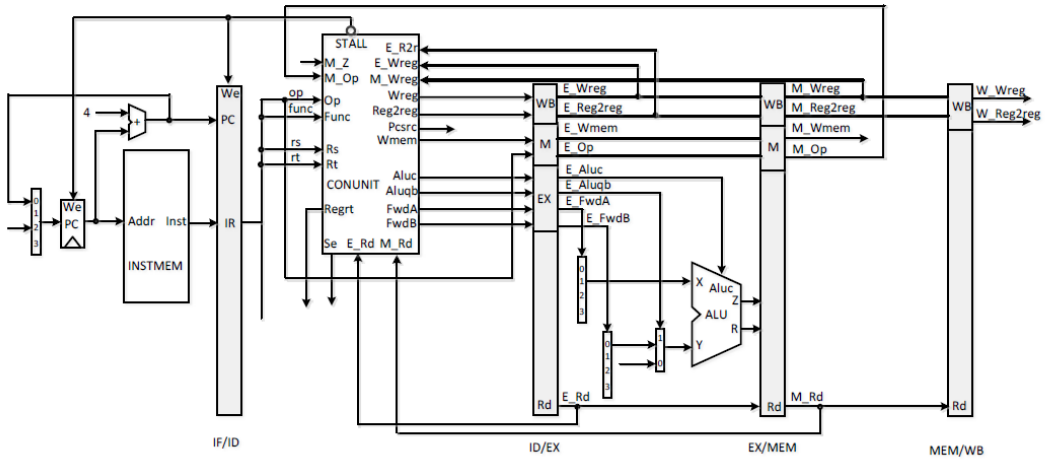
带流水线的数据通路如图所示。



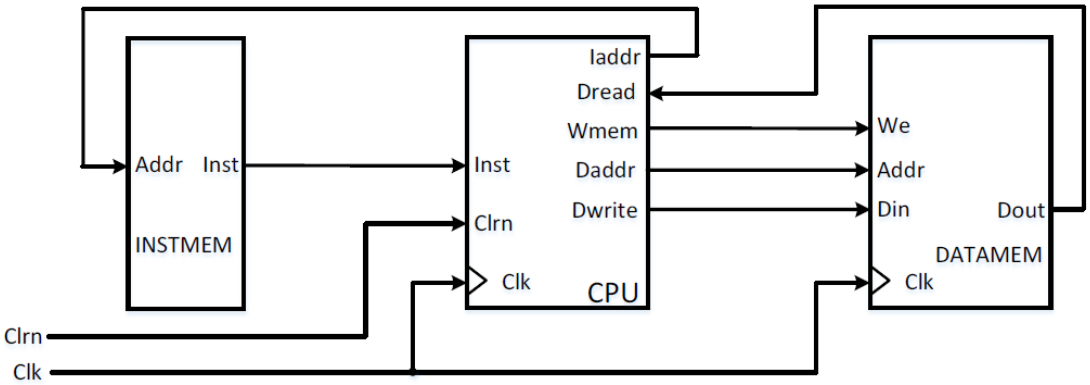
内部前推机制如图所示。



暂停流水线的方式如图所示。



要求封装成以下形式。



要求在以上电路图中的寄存器堆接入输出并仿真结果。

提交实验报告要求： CPU 的设计代码，仿真代码，仿真波形图和 RTL 图。

注：CPU_new 为流水线 CPU，CPU2 为流水线 CPU 内模块，CPU_simple 为简单 CPU，CPU1 位简单 CPU 内模块

设计代码：

```
module CPU2(Clk, Clrn, Inst[31:0], Dread[31:0], Iaddr[31:0], Daddr[31:0], Dwrite[31:0], Wmem);

    input Clk,Clrn;

    input [31:0] Inst, Dread;

    output [31:0] Iaddr, Daddr, Dwrite;

    output Wmem;


    wire [31:0] Q;
```

```
    wire [31:0] D = PC_MUX4X32_Y;
    wire    PC_En = stall;
    PC pc(Clk, PC_En, D,Q);

    wire [31:0]CLA_32_X = Q;
    wire [31:0]CLA_32_S;
    CLA_32_plus4 cla_32_plus4(CLA_32_X,CLA_32_S);

    assign Iaddr = Q;
    wire IF_ID_We = stall;
    wire [31:0]IF_ID_IR_in = Inst;
    wire [31:0]IF_ID_PC_in = CLA_32_S;
    wire [31:0]IF_ID_IR_out,IF_ID_PC_out;
    IF_ID if_id(IF_ID_We, Clk, Clrn, IF_ID_PC_in,IF_ID_PC_out,IF_ID_IR_in,IF_ID_IR_out);

    wire [15:0]EXT_X = IF_ID_IR_out[15:0];
    wire EXT_Se = Se;
    wire [31:0]EXT_Y;
    EXT16T32 ext16to32(EXT_X,EXT_Se,EXT_Y);

    wire [31:0]A_MUX4X32_Y;
    wire [31:0]A_MUX4X32_A0 = ID_EX_R1_out;
    wire [31:0]A_MUX4X32_A1 = ALU_DATAMEM_Y;
    wire [31:0]A_MUX4X32_A2 = R_out;
    wire [1:0] A_MUX4X32_S = E_FwdA;
    MUX4X32
MUX4X32_FwdA(A_MUX4X32_A0,A_MUX4X32_A1 ,A_MUX4X32_A2,,A_MUX4X32_S,A_MUX4
X32_Y);

    wire [31:0]B_MUX4X32_Y;
```

```
    wire [31:0]B_MUX4X32_A0 = ID_EX_R2_out;
    wire [31:0]B_MUX4X32_A1 = ALU_DATAMEM_Y;
    wire [31:0]B_MUX4X32_A2 = R_out;
    wire [1:0] B_MUX4X32_S = E_FwdB;
     MUX4X32
MUX4X32_FwdB(B_MUX4X32_A0,B_MUX4X32_A1 ,B_MUX4X32_A2,,B_MUX4X32_S,B_MUX4X
32_Y);

    wire [5:0]Op = IF_ID_IR_out[31:26];
    wire [5:0]Func = IF_ID_IR_out [5:0];
    wire [1:0] Pcsrc, Aluc;
    wire CON_M_Z = M_Z;
    wire [5:0]CON_M_Op = M_Op;
    wire Wreg,Se,Aluqb,Regrt,Wmem, Reg2reg;
    wire CON_V;
    wire [4:0]E_Rd = ID_EX_Rd_out;
    wire [4:0]M_Rd = EX_MEM_Rd_out;
    wire Con_E_Wreg = E_Wreg;
    wire Con_M_Wreg = M_Wreg;
    wire [4:0]Con_Rs = IF_ID_IR_out[25:21];
    wire [4:0]Con_Rt = IF_ID_IR_out[20:16];
    wire [1:0] FwdA,FwdB;
    wire E_Reg2reg_in = E_Reg2reg;
    wire Con_Z;
    wire stall;
    CONUNIT conunit(Op, Func, Con_Z    , Regrt, Se, Wreg, Aluqb, Aluc, Wmem, Pcsrc,
Reg2reg,CON_V ,CON_M_Z,
CON_M_Op,E_Rd,M_Rd,Con_E_Wreg,Con_M_Wreg,Con_Rs,Con_Rt,FwdA,FwdB,E_Reg2reg_in
,stall);

    wire [4:0]rt = IF_ID_IR_out[20:16];
```

```
wire [4:0]rd = IF_ID_IR_out[15:11];
wire [4:0]INS_REG_YY;
MUX2X5 INS_REG(rd, rt, Regrt, INS_REG_YY);

wire [4:0]rs = IF_ID_IR_out[25:21];
wire [4:0]Ra = rs;
wire [4:0]Rb = rt;
wire [4:0]Wr = MEM_WB_Rd_out;
wire We = W_Wreg;
wire [31:0]REG_D = ALU_DATAMEM_Y;
wire [31:0] Qa, Qb;
wire CLk_n = ~Clk;
REGFILE regfile(Ra, Rb, REG_D, Wr, We, CLk_n, Clrn, Qa, Qb);//改成下降沿写入

wire [31:0]ID_EX_PC_in = IF_ID_PC_out;
wire [31:0]ID_EX_R1_in = Qa;
wire [31:0]ID_EX_R2_in = Qb;
wire [31:0]I_in = EXT_Y;
wire [4:0]ID_EX_Rd_in = INS_REG_YY;
wire [31:0]ID_EX_PC_out,ID_EX_R1_out,ID_EX_R2_out,I_out;
wire [4:0]ID_EX_Rd_out;
wire [5:0]E_Op;
wire E_Wreg,E_Reg2reg,E_Wmem,E_Aluqb;
wire [1:0]E_Aluc;
wire [1:0]E_FwdA,E_FwdB;
wire [1:0]E_FwdA_in = FwdA;
wire [1:0]E_FwdB_in = FwdB;
parameter ID_EX_We = 1;
wire ID_EX_Clrn = stall;
```

```
    ID_EX id_ex(ID_EX_We,Clk,
ID_EX_Clrn,ID_EX_PC_in,ID_EX_PC_out,ID_EX_R1_in,ID_EX_R1_out,
ID_EX_R2_in,ID_EX_R2_out,I_in,I_out,
    Wreg,E_Wreg, Reg2reg,E_Reg2reg,Wmem,E_Wmem, Op,E_Op, Aluc,E_Aluc,
Aluqb,E_Aluqb, ID_EX_Rd_in,ID_EX_Rd_out,E_FwdA_in,E_FwdA,E_FwdB_in,E_FwdB);


    wire [31:0]Sh1;
    SHIFTER32_L2 shift32_L2(I_out,Sh1);


    parameter Sub = 1'b0;
    wire [31:0]ADDSUB_S;
    ADDSUB_32 addsub_32(Sh1, ID_EX_PC_out, Sub, ADDSUB_S);


    wire [31:0]REG_ALU_Y;
    wire [31:0]REG_ALU_A1 = B_MUX4X32_Y;
    wire [31:0]REG_ALU_A0 = I_out;
    MUX2X32 REG_ALU(REG_ALU_A0, REG_ALU_A1, E_Aluqb, REG_ALU_Y);


    wire [31:0]PC_MUX4X32_A0 = CLA_32_S;
    wire [31:0]PC_MUX4X32_A2 = EX_MEM_PC_out;
    wire [1:0]PC_MUX4X32_S = Pcsrc;
    wire [31:0]PC_MUX4X32_Y;
    MUX4X32
PC_MUX4X32(PC_MUX4X32_A0,,PC_MUX4X32_A2,,PC_MUX4X32_S,PC_MUX4X32_Y);


    wire [31:0]ALU_X = A_MUX4X32_Y;
    wire [31:0]ALU_Y = REG_ALU_Y;
    wire [1:0]ALU_Aluc = E_Aluc;
    wire ALU_Z,V;
    wire [31:0]ALU_R;
    ALU alu(ALU_X,ALU_Y,ALU_Aluc,ALU_R,ALU_Z,V);
```

```
    parameter EX_MEM_We = 1'b1;
    wire [31:0]EX_MEM_PC_in = ADDSUB_S;
    wire Z_in = ALU_Z;
    wire [31:0]R_in = ALU_R;
    wire [31:0]S_in = B_MUX4X32_Y;
    wire [4:0]EX_MEM_Rd_in = ID_EX_Rd_out;
    wire [5:0]M_Op;
    wire [4:0]EX_MEM_Rd_out;
    wire [31:0]EX_MEM_PC_out,R_out,S_out;
    wire M_Wreg,M_Reg2reg,M_Wmem,M_Z;
    EX_MEM ex_mem(EX_MEM_We,Clk, Clrn,
EX_MEM_PC_in,EX_MEM_PC_out,Z_in,M_Z,R_in,R_out,S_in,S_out,EX_MEM_Rd_in,EX_MEM_Rd
_out,E_Wreg,M_Wreg,E_Reg2reg,M_Reg2reg,E_Wmem,M_Wmem,E_Op,M_Op);

    assign Daddr = R_out;
    assign Dwrite = S_out;
    assign Wmem = M_Wmem;

    parameter MEM_WB_We = 1'b1;
    wire [31:0]D_in = R_out;
    wire [31:0]D_out;
    wire [31:0]C_in = Dread;
    wire [31:0]C_out;
    wire [4:0]MEM_WB_Rd_in = EX_MEM_Rd_out;
    wire [4:0]MEM_WB_Rd_out;
    wire W_Wreg,W_Reg2reg;
    MEM_WB
mem_wb(MEM_WB_We,Clk,Clrn,D_in,D_out,C_in,C_out,MEM_WB_Rd_in,MEM_WB_Rd_out,M
_Wreg,W_Wreg,M_Reg2reg,W_Reg2reg);
```

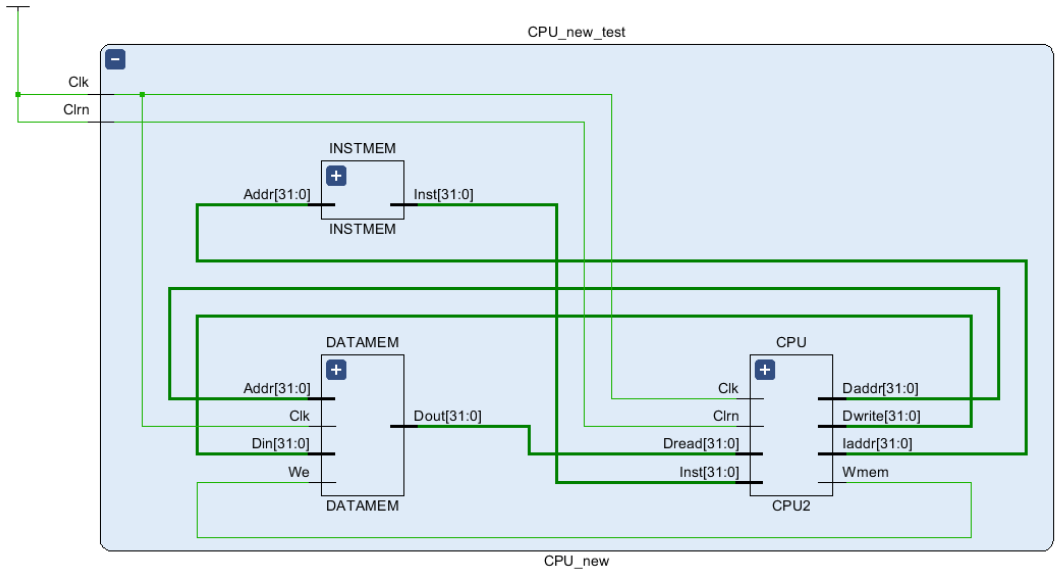
```
    wire [31:0]ALU_DATAMEM_Y;
    wire [31:0]ALU_DATAMEM_Y0 = C_out;
    wire [31:0]ALU_DATAMEM_Y1 = D_out;
    MUX2X32 ALU_DATAMEM(ALU_DATAMEM_Y0, ALU_DATAMEM_Y1, W_Reg2reg,
ALU_DATAMEM_Y);

Endmodule

module CPU_new(Clk, Clrn);
      input Clk,Clrn;
      wire [31:0] Inst, Dread;
      wire [31:0] Iaddr, Daddr, Dwrite;
      wire Wmem;
      CPU2 CPU(Clk, Clrn, Inst, Dread, Iaddr, Daddr, Dwrite, Wmem);

      DATAMEM DATAMEM(Daddr, Dwrite, Clk, Wmem, Dread);

      INSTMEM INSTMEM(Iaddr, Inst);
Endmodule
```

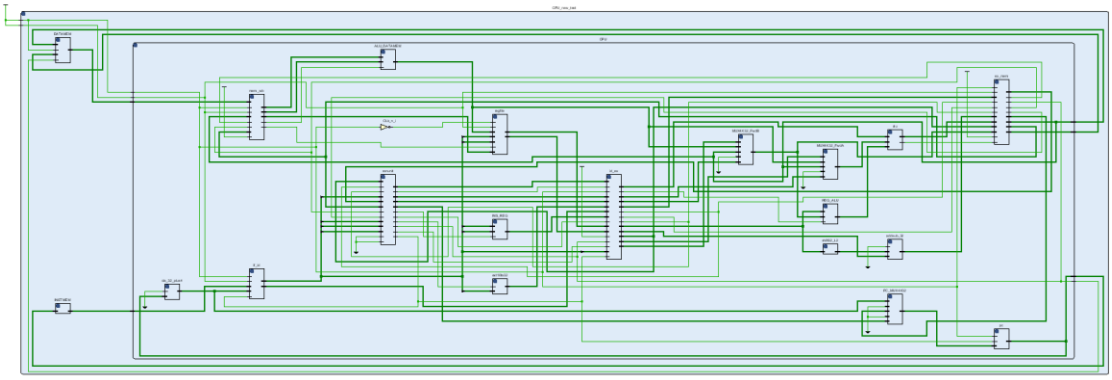
仿真代码：

```
module CPU_new_TEST;
      reg Clk,Clrn;
      CPU_new CPU_new_test(.Clk(Clk),.Clrn(Clrn));
            initial begin;
                  Clrn = 1;
                  Clk = 0;
                  #20;
                  ……
```

```
        Clk =~Clk;
        #20;
        ……
    end
endmodule
```

RTL 图：





仿真图:

新指令：

```
//提前半周期写入
assign Rom[5'h00] = 32'b100011_00000_00001_00000_000000_00100;//取出数放在$1=1
assign Rom[5'h01] = 32'b100011_00000_00010_00000_000000_01000;//取出数放在$2=2
assign Rom[5'h02] = 32'b100011_00000_00011_00000_000000_01100;//取出数放在$3=3
```

assign Rom[5'h03] = 32'b100011_00000_00100_00000_000000_10000;//取出数放在$4=4

//内部前推

assign Rom[5'h04] = 32'b001000_00001_00010_0000000000000010;//addi $2,$1,2; $2 = $1 + 2 = 3

assign Rom[5'h05] = 32'b000000_00010_00100_00011_000001_00100;//and $3,$4,$2; $3 = $4 & $2 = 4 & 3 = 0

assign Rom[5'h06] = 32'b100011_00010_01000_0000000000011100;//lw $8, 7($2); $8 = 7

assign Rom[5'h07] = 32'b001101_00010_00110_0000000000000100;//ori $6,$2,20; $6 = $2 | 4 = 3 | 4 = 7

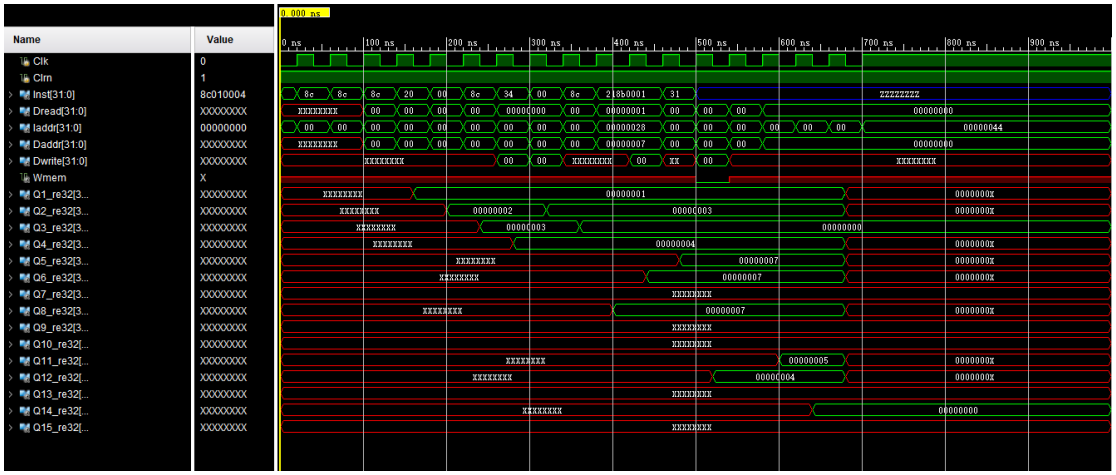
assign Rom[5'h08] = 32'b000000_00100_00010_00101_000001_00000;//add $5, $4, $2; $5 = $2 + $4 = 7

//Lw 指令的数据冒险

assign Rom[5'h09] = 32'b100011_00000_01100_00000_000000_10000;//取出数放在$12=4

assign Rom[5'h0A] = 32'b001000_01100_01011_0000000000000001;//addi $11, $12, 1; $11 = $12 + 1 = 5;

assign Rom[5'h0B] = 32'b001100_01100_01110_0000000000000010;//andi $14, $12 , 2; $14 = $12 & 2 = 0;



3　实验一门电路的实现指导

3.1 相关知识

3.1.1 注释符

Verilog HDL 语言允许插入注释，标明程序代码功能、修改、版本等信息，以增强程序的可阅读性和帮助管理文档。Verilog HDL 有两种注释方式：

- 单行注释：单行注释以“//”开始，Verilog HDL 忽略从此处到行尾的内容；

❖ 多行注释：多行注释以“/*”开始，到“*/”结束，Verilog 忽略其中的注释内容。

3.1.2 标识符和转义符

在 Verilog HDL 中，标识符（Identifier）被用来命令信号名、模块名、参数名等。它可以使任意一组字母、数字、$符号和_符号的组合。应该注意的是，标识符的字符区分大小写，并且第一个字符必须是字母或者下划线。

Verilog HDL 规定了转义标识符（Escaped Identifier）。采用转义字符可以在一条标识符中包含任何可打印的字符。转义标识符以“\”（反斜线）符号开头，以空白符结尾（空白可以是 一个空格、一个制表符或者换行符）。

3.1.3 关键字

Verilog HDL 语言内部已经使用的词称为关键字或保留字，它是 Verilog HDL 语言的内部专用词，是事先定义好的确认符，用来组织语言结构的。需要注意的是，在 Verilog HDL 中，保留字都是小写的。

❖ 与门、或门以及同类门单元

在 Verilog HDL 编程中实例化此类门单元需要用下列关键字中的一个作为实例化的模块名：

and(与)，nand(与非)，nor(或非)，or(或)，xor(异或)，xnor(异或非)

此类门电路的使用如下：

and and1 (tmp_1, x, y，z);

其中，“and1”是门单元“and”的一个实例化接口名称，第一个参数 tmp_1 是输出变量，即输入变量 x，y，z 进行 and 的结果。输出变量只有 1 个，输入变量多于两个。其余此类门电路的使用类似。

❖ 非门

和与/非类门单元相反，非门具有一个输入端口，以及一个或多个输出端口。在在 Verilog HDL 编程中实例化此类门单元需要用下列关键字中的一个作为实例化的模块名：

not(非)

此类门电路的使用如下：

not not1 (tmp_2, x);

其中，“not1”是门单元“not”的一个实例化接口名称，第一个参数 tmp_2 输出变量，即输入变量 x 进行 not 的结果。

3.1.4 数值

Verilog HDL 有四种基本的逻辑数值状态，用数字或字符表达数字电路中传送的逻辑状态和存储信息。Verilog HDL 逻辑数值中，x 和 z 都不区分大小写。也就是说 ，0x1z 和值 0X1Z 是等同的。

Verilog HDL 中的四值电平逻辑如下表：

状态	含义
0	低电平、逻辑 0、“假”
1	高电平，逻辑 1 或“真”
X 或 x	不确定或未知的逻辑状态
Z 或 Z	高阻态

3.1.5 仿真文件中的变量类型

reg：reg 定义的是寄存器类型，通过赋值语句可以改变寄存器储存的值，reg 型变量常用来表示用于“always”模块内的指定信号，在“always”块内被赋值的每一个信号都必须定义成 reg 型。

定义 reg 变量的格式为”reg ([n – 1:0]) 变量名 1,变量名 2,…,变量名 I;”其中当宽度为 1 时，括号中的内容可以去掉，n 代表是一个几位的变量，比如说声明一个 32 位的寄存器类型变量，格式为 reg [31:0] a;声明一个一位的寄存器变量为 reg b;

wire：wire 型变量通常是用来表示单个门驱动或连续赋值语句驱动的网络型数据，是用于连接器件单元，不能直接进行赋值，而必须使用 wire 或者 reg 来赋值，定义的方式和 reg 相同。比如说 wire a; a = 1 这种做法是错误的，而必须使用 “reg a = 1; wire b = a;wire c = b; ”。

3.1.6 阻塞赋值

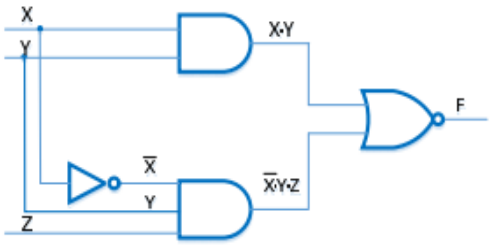
阻塞是指在同一个 always 块中，其后面的赋值语句从概念上是 在前一条赋值语句结束后开始赋值的。非阻塞语句的执行过程是：首先计算语句块内部所有右边表达式的值，然后完成对左边寄存器变量的赋值操作。

阻塞赋值： begin B=A; C=B+1; end 上述代码先将 A 的值赋值给 B，C 的值是 A+1；	非阻塞赋值： begin B<=A; C<=B+1; end 上述代码的最终结果是：将 A 赋值给了 B，但是 C 的值是 B 原来的值+1。因为最先计是的是右边的表达式。

3.2 实验演示

门电路如右图所示，则其设计文件如下：

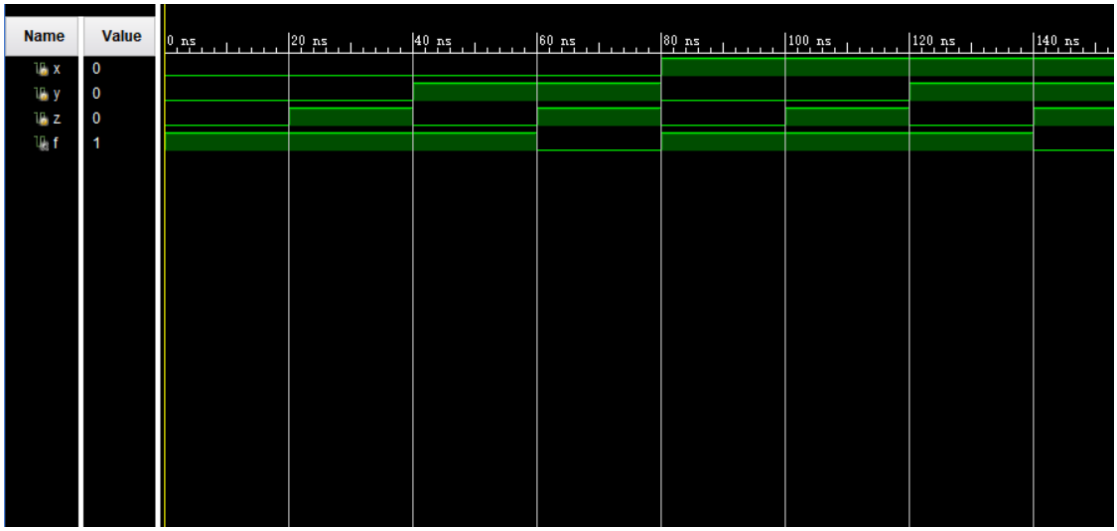
```
module door(x, y, z, f);
    input x, y, z;
    output f;
    and and1 (tmp_1, z, y);
    not not1 (tmp_2, x);
    and and2 (tmp_3, tmp_2, y, z);
    nor nor1 (f, tmp_1, tmp_3);
endmodule
```



仿真文件：

```
module door_TEST;
    // Inputs
    reg x,y,z;     //  定义三个输入变量，为模块的三个输入，定义成 reg 赋值
    // Outputs
    wire f;        //  定义输出变量，使用 wire 作为变量类型
    //wire S0, S1,S2;       //  定义三个中间变量
     // Instantiate the Unit Under Test (UUT)
    door uut (.x(x),.y(y),.z(z),.f(f));          //应用设计模块
    initial begin
        // Initialize Inputs
        x = 0;            //把 x，y，z 的初始值设置为 0
        y = 0;
        z = 0;
        // Wait 20 ns for global reset to finish
        #20;            //等待 20ns 改变 x,z,y 参数的值
        x = 0;
        y = 0;
        z = 1;
        #20;
        x = 0;
        y = 1;

      z = 0;
        ......
        ......
        #20;
        x = 1;
        y = 1;
        z = 1;
    end
```

仿真图如下：


Name
Value
x
0
y
0
z
0
f
1
0 ns
20 ns
40 ns
60 ns
80 ns
100 ns
120 ns
140 ns