北京科技大学 计算机与通信工程学院

实 验 报 告

实验名称：计算机组成原理实验

学生姓名：唐誉源

专　　业：计算机科学与工程

班　　级：计1503

学　　号：41503302

指导教师：张晓彤/刘宏岚 张磊

实验成绩：

实验地点：机电楼304

实验时间：2017 年 5 月 17 日

一、实验目的与实验要求

1、实验目的

本实验要求学生运用EDA技术，采用系统能力培养指定的实践平台（Xilinx Vivado设计工具、 Xilinx EGO1实验平台），自己设计一些基本的计算机组成原理电路单元，进一步加深对组成原理中重要部件原理的理解。学会运用XSIM仿真或下载到板来验证自己的设计。

2、实验要求

1）由于在有限的实验课内学时难以较好完成所有实验内容，因此要求在实验课前进行预习，自主完成部分实验或实验的部分内容（包括代码设计）；

2）完成的每个实验需要在实验课内经指导教师现场检查、查看编写的程序代码以及测试运行情况，回答指导教师提出的问题，以确认实验实际完成的质量；

3）代码应有适当的注释，并在实验报告中体现；仿真实现的设计需要有仿真波形截图。

二、实验设备（环境）及要求

Xilinx Ego1实验平台。

OS：Win7 64位

Software：Vivado15.4开发工具

三、实验内容、步骤与结果分析

1 实验1 32位寄存器和寄存器堆的设计

1.1实验内容

用Vivado软件和VerilogHDL或VHDL语言设计实现32位寄存器堆，要求：

支持32位寄存器地址选择；

支持写入任意32位值；

支持读取寄存器的值。

如图1-1所示，实现圆圈内所示电路。输入输出变量名字尽量与图示保持一致。

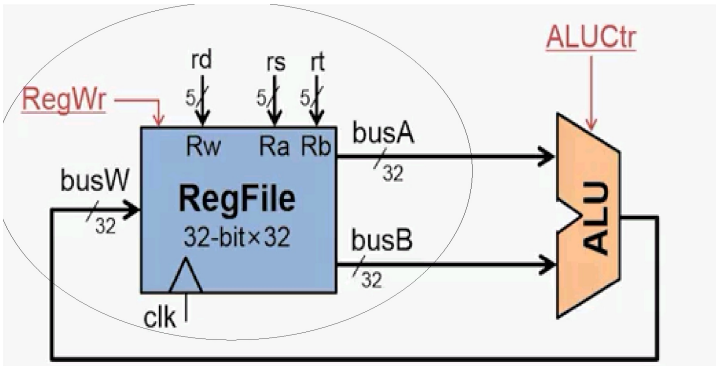


图1-1

1.2主要步骤

1.2.1 分析实现的方法

寄存器堆的结构示意图，内部有 32 个 32 位的寄存器，有 3 组数据接口，其中 busA 和 busB 是两个 32 位的数据输出接口，busW 是 32 位的数据输入接口。对寄存器堆进行读写，需要读写控制信号，首先是 Ra，这是一个 5 位的信号，5 位的信号正好可以选择编号 0-31 的寄存器，因此寄存器堆会根据 Ra 的输入，选择对应编号的寄存器，将其 内容放在 busA 信号上，同样根据 Rb 选中对应编号寄存器，将其内容放到 busB 信号上。这 样外界将两个编号分别为 Ra 和 Rb 信号放在寄存器堆输入端，寄存器堆就会将寄存器的内 容分别放在 busA 和 busB 上。这就完成了同时读取两个寄存器的功能。写寄存器操作稍微复杂一些，首先将要写的寄存器的编号通过 Rw 信号输入，在时钟信号的上升沿如果写使能信号有效(WriteEnable==1)，寄存器堆就会将 busW 的内容存入 Rw 信号指定的寄存器。寄存器堆的读操作不受时钟控制，即在任何时候 Ra 或者 Rb 发生了变化，对应的 busA 或者 busB的信号就会发生变化。

1.2.2 实现代码和仿真代码

实现代码

```
module Dlatch(d,e,q,_q);
input d,e;
output reg q,_q;
always@(posedge e)
begin
    if(e==1)
    begin
        q=d;_q=~d;
    end
end
endmodule

module Dlatch32(data,e,q);
input e;
input [31:0]data;
output [31:0]q;
wire [31:0]_q;
Dlatch d0(data[0],e,q[0],_q[0]);
Dlatch d1(data[1],e,q[1],_q[1]);
Dlatch d2(data[2],e,q[2],_q[2]);
Dlatch d3(data[3],e,q[3],_q[3]);
Dlatch d4(data[4],e,q[4],_q[4]);
Dlatch d5(data[5],e,q[5],_q[5]);
Dlatch d6(data[6],e,q[6],_q[6]);
Dlatch d7(data[7],e,q[7],_q[7]);
Dlatch d8(data[8],e,q[8],_q[8]);
Dlatch d9(data[9],e,q[9],_q[9]);
Dlatch d10(data[10],e,q[10],_q[10]);
Dlatch d11(data[11],e,q[11],_q[11]);
Dlatch d12(data[12],e,q[12],_q[12]);
```

```
Dlatch d13(data[13],e,q[13],_q[13]);
Dlatch d14(data[14],e,q[14],_q[14]);
Dlatch d15(data[15],e,q[15],_q[15]);
Dlatch d16(data[16],e,q[16],_q[16]);
Dlatch d17(data[17],e,q[17],_q[17]);
Dlatch d18(data[18],e,q[18],_q[18]);
Dlatch d19(data[19],e,q[19],_q[19]);
Dlatch d20(data[20],e,q[20],_q[20]);
Dlatch d21(data[21],e,q[21],_q[21]);
Dlatch d22(data[22],e,q[22],_q[22]);
Dlatch d23(data[23],e,q[23],_q[23]);
Dlatch d24(data[24],e,q[24],_q[24]);
Dlatch d25(data[25],e,q[25],_q[25]);
Dlatch d26(data[26],e,q[26],_q[26]);
Dlatch d27(data[27],e,q[27],_q[27]);
Dlatch d28(data[28],e,q[28],_q[28]);
Dlatch d29(data[29],e,q[29],_q[29]);
Dlatch d30(data[30],e,q[30],_q[30]);
Dlatch d31(data[31],e,q[31],_q[31]);
Dlatch d32(data[32],e,q[32],_q[32]);
endmodule

module Mux32(
      input e1,e2,e3,e4,e5,
      input [31:0]data,
      output [31:0]out
      );
wire e;
and a(e,e1,e2,e3,e4,e5);
and a0(out[0],data[0],e);
and a1(out[1],data[1],e);
and a2(out[2],data[2],e);
and a3(out[3],data[3],e);
and a4(out[4],data[4],e);
and a5(out[5],data[5],e);
and a6(out[6],data[6],e);
and a7(out[7],data[7],e);
and a8(out[8],data[8],e);
and a9(out[9],data[9],e);
and a10(out[10],data[10],e);
and a11(out[11],data[11],e);
and a12(out[12],data[12],e);
and a13(out[13],data[13],e);
and a14(out[14],data[14],e);
and a15(out[15],data[15],e);
and a16(out[16],data[16],e);
and a17(out[17],data[17],e);
and a18(out[18],data[18],e);
and a19(out[19],data[19],e);
and a20(out[20],data[20],e);
and a21(out[21],data[21],e);
and a22(out[22],data[22],e);
and a23(out[23],data[23],e);
and a24(out[24],data[24],e);
and a25(out[25],data[25],e);
and a26(out[26],data[26],e);
and a27(out[27],data[27],e);
and a28(out[28],data[28],e);
and a29(out[29],data[29],e);
and a30(out[30],data[30],e);
and a31(out[31],data[31],e);
endmodule

module Mux(
     i n p u t [31:0]data0,data1,data2,data3,data4,data5,

data6,data7,data8,data9,data10,data11,

data12,data13,data14,data15,data16,

data17,data18,data19,data20,data21,

data22,data23,data24,data25,data26,

data27,data28,data29,data30,data31,
      input [4:0]ra,
      output [31:0]rd
      );
wire [4:0]_ra;
wire [31:0]rd0,rd1,rd2,rd3,rd4,rd5,
           rd6,rd7,rd8,rd9,rd10,rd11,
           rd12,rd13,rd14,rd15,rd16,
           rd17,rd18,rd19,rd20,rd21,
           rd22,rd23,rd24,rd25,rd26,
           rd27,rd28,rd29,rd30,rd31;
not n1(_ra[1],ra[1]);
not n2(_ra[2],ra[2]);
not n3(_ra[3],ra[3]);
not n4(_ra[4],ra[4]);
not n5(_ra[0],ra[0]);
M u x 3 2
m0(_ra[0],_ra[1],_ra[2],_ra[3],_ra[4],data0,r
d0);
M u x 3 2
m1(ra[0],_ra[1],_ra[2],_ra[3],_ra[4],data1,rd
1);
M u x 3 2
m2(_ra[0],ra[1],_ra[2],_ra[3],_ra[4],data2,rd
2);
M u x 3 2
m3(ra[0],ra[1],_ra[2],_ra[3],_ra[4],data3,rd3
);
M u x 3 2
m4(_ra[0],_ra[1],ra[2],_ra[3],_ra[4],data4,rd
4);
```

```
Mux32 m5(ra[0],_ra[1],ra[2],_ra[3],_ra[4],data5,rd5);
Mux32 m6(_ra[0],ra[1],ra[2],_ra[3],_ra[4],data6,rd6);
Mux32 m7(ra[0],ra[1],ra[2],_ra[3],_ra[4],data7,rd7);
Mux32 m8(_ra[0],_ra[1],_ra[2],ra[3],_ra[4],data8,rd8);
Mux32 m9(ra[0],_ra[1],_ra[2],ra[3],_ra[4],data9,rd9);
Mux32 m10(_ra[0],ra[1],_ra[2],ra[3],_ra[4],data10,rd10);
Mux32 m11(ra[0],ra[1],_ra[2],ra[3],_ra[4],data11,rd11);
Mux32 m12(_ra[0],_ra[1],ra[2],ra[3],_ra[4],data12,rd12);
Mux32 m13(ra[0],_ra[1],ra[2],ra[3],_ra[4],data13,rd13);
Mux32 m14(_ra[0],ra[1],ra[2],ra[3],_ra[4],data14,rd14);
Mux32 m15(ra[0],ra[1],ra[2],ra[3],_ra[4],data15,rd15);
Mux32 m16(_ra[0],_ra[1],_ra[2],_ra[3],ra[4],data16,rd16);
Mux32 m17(ra[0],_ra[1],_ra[2],_ra[3],ra[4],data17,rd17);
Mux32 m18(_ra[0],ra[1],_ra[2],_ra[3],ra[4],data18,rd18);
Mux32 m19(ra[0],ra[1],_ra[2],_ra[3],ra[4],data19,rd19);
Mux32 m20(_ra[0],_ra[1],ra[2],_ra[3],ra[4],data20,rd20);
Mux32 m21(ra[0],_ra[1],ra[2],_ra[3],ra[4],data21,rd21);
Mux32 m22(_ra[0],ra[1],ra[2],_ra[3],ra[4],data22,rd22);
Mux32 m23(ra[0],ra[1],ra[2],_ra[3],ra[4],data23,rd23);
Mux32 m24(_ra[0],_ra[1],_ra[2],ra[3],ra[4],data24,rd24);
Mux32 m25(ra[0],_ra[1],_ra[2],ra[3],ra[4],data25,rd25);
Mux32 m26(_ra[0],ra[1],_ra[2],ra[3],ra[4],data26,rd26);
Mux32 m27(ra[0],ra[1],_ra[2],ra[3],ra[4],data27,rd27);
Mux32 m28(_ra[0],_ra[1],ra[2],ra[3],ra[4],data28,rd28);
Mux32 m29(ra[0],_ra[1],ra[2],ra[3],ra[4],data29,rd29);
Mux32 m30(_ra[0],ra[1],ra[2],ra[3],ra[4],data30,rd30);
Mux32 m31(ra[0],ra[1],ra[2],ra[3],ra[4],data31,rd31);
or o0(rd[0],rd0[0],rd1[0],rd2[0],rd3[0],rd4[0],rd5[0],
rd6[0],rd7[0],rd8[0],rd9[0],rd10,rd11[0],rd12[0],
rd13[0],rd14[0],rd15[0],rd16[0],rd17[0],rd18[0],
rd19[0],rd20[0],rd21,rd22[0],rd23[0],rd24[0],rd25[0],
rd26[0],rd27[0],rd28[0],rd29[0],rd30[0],rd31[0]);
or o1(rd[1],rd0[1],rd1[1],rd2[1],rd3[1],rd4[1],rd5[1],
rd6[1],rd7[1],rd8[1],rd9[1],rd10,rd11[1],rd12[1],
rd13[1],rd14[1],rd15[1],rd16[1],rd17[1],rd18[1],
rd19[1],rd20[1],rd21,rd22[1],rd23[1],rd24[1],
```

```
rd25[1],
rd26[1],rd27[1],rd28[1],rd29[1],rd30[1],rd31[
1]);
o r
o2(rd[2],rd0[2],rd1[2],rd2[2],rd3[2],rd4[2],rd
5[2],
rd6[2],rd7[2],rd8[2],rd9[2],rd10,rd11[2],rd12
[2],
rd13[2],rd14[2],rd15[2],rd16[2],rd17[2],rd18[
2],
rd19[2],rd20[2],rd21,rd22[2],rd23[2],rd24[2],
rd25[2],
rd26[2],rd27[2],rd28[2],rd29[2],rd30[2],rd31[
2]);
o r
o3(rd[3],rd0[3],rd1[3],rd2[3],rd3[3],rd4[3],rd
5[3],
rd6[3],rd7[3],rd8[3],rd9[3],rd10,rd11[3],rd12
[3],
rd13[3],rd14[3],rd15[3],rd16[3],rd17[3],rd18[
3],
rd19[3],rd20[3],rd21,rd22[3],rd23[3],rd24[3],
rd25[3],
rd26[3],rd27[3],rd28[3],rd29[3],rd30[3],rd31[
3]);
o r
o4(rd[4],rd0[4],rd1[4],rd2[4],rd3[4],rd4[4],rd
5[4],
rd6[4],rd7[4],rd8[4],rd9[4],rd10,rd11[4],rd12
[4],
rd13[4],rd14[4],rd15[4],rd16[4],rd17[4],rd18[
4],
rd19[4],rd20[4],rd21,rd22[4],rd23[4],rd24[4],
rd25[4],
rd26[4],rd27[4],rd28[4],rd29[4],rd30[4],rd31[
4]);
o r
o5(rd[5],rd0[5],rd1[5],rd2[5],rd3[5],rd4[5],rd
5[5],
rd6[5],rd7[5],rd8[5],rd9[5],rd10,rd11[5],rd12
[5],
rd13[5],rd14[5],rd15[5],rd16[5],rd17[5],rd18[
5],
rd19[5],rd20[5],rd21,rd22[5],rd23[5],rd24[5],
rd25[5],
rd26[5],rd27[5],rd28[5],rd29[5],rd30[5],rd31[
5]);
o r
o6(rd[6],rd0[6],rd1[6],rd2[6],rd3[6],rd4[6],rd
5[6],
rd6[6],rd7[6],rd8[6],rd9[6],rd10,rd11[6],rd12
[6],
rd13[6],rd14[6],rd15[6],rd16[6],rd17[6],rd18[
6],
rd19[6],rd20[6],rd21,rd22[6],rd23[6],rd24[6],
rd25[6],
rd26[6],rd27[6],rd28[6],rd29[6],rd30[6],rd31[
6]);
o r
o7(rd[7],rd0[7],rd1[7],rd2[7],rd3[7],rd4[7],rd
5[7],
rd6[7],rd7[7],rd8[7],rd9[7],rd10,rd11[7],rd12
[7],
rd13[7],rd14[7],rd15[7],rd16[7],rd17[7],rd18[
7],
rd19[7],rd20[7],rd21,rd22[7],rd23[7],rd24[7],
rd25[7],
rd26[7],rd27[7],rd28[7],rd29[7],rd30[7],rd31[
7]);
o r
o8(rd[8],rd0[8],rd1[8],rd2[8],rd3[8],rd4[8],rd
5[8],
rd6[8],rd7[8],rd8[8],rd9[8],rd10,rd11[8],rd12
[8],
rd13[8],rd14[8],rd15[8],rd16[8],rd17[8],rd18[
8],
rd19[8],rd20[8],rd21,rd22[8],rd23[8],rd24[8],
rd25[8],
```

```
rd26[8],rd27[8],rd28[8],rd29[8],rd30[8],rd31[8]);
or o9(rd[9],rd0[9],rd1[9],rd2[9],rd3[9],rd4[9],rd5[9],
rd6[9],rd7[9],rd8[9],rd9[9],rd10,rd11[9],rd12[9],
rd13[9],rd14[9],rd15[9],rd16[9],rd17[9],rd18[9],
rd19[9],rd20[9],rd21,rd22[9],rd23[9],rd24[9],rd25[9],
rd26[9],rd27[9],rd28[9],rd29[9],rd30[9],rd31[9]);
or o10(rd[10],rd0[10],rd1[10],rd2[10],rd3[10],rd4[10],rd5[10],
rd6[10],rd7[10],rd8[10],rd9[10],rd10,rd11[10],rd12[10],
rd13[10],rd14[10],rd15[10],rd16[10],rd17[10],rd18[10],
rd19[10],rd20[10],rd21,rd22[10],rd23[10],rd24[10],rd25[10],
rd26[10],rd27[10],rd28[10],rd29[10],rd30[10],rd31[10]);
or o11(rd[11],rd0[11],rd1[11],rd2[11],rd3[11],rd4[11],rd5[11],
rd6[11],rd7[11],rd8[11],rd9[11],rd10,rd11[11],rd12[11],
rd13[11],rd14[11],rd15[11],rd16[11],rd17[11],rd18[11],
rd19[11],rd20[11],rd21,rd22[11],rd23[11],rd24[11],rd25[11],
rd26[11],rd27[11],rd28[11],rd29[11],rd30[11],rd31[11]);
or o12(rd[12],rd0[12],rd1[12],rd2[12],rd3[12],rd4[12],rd5[12],
rd6[12],rd7[12],rd8[12],rd9[12],rd10,rd11[12],rd12[12],
rd13[12],rd14[12],rd15[12],rd16[12],rd17[12],rd18[12],
rd19[12],rd20[12],rd21,rd22[12],rd23[12],rd24[12],rd25[12],
rd26[12],rd27[12],rd28[12],rd29[12],rd30[12],rd31[12]);
or o13(rd[13],rd0[13],rd1[13],rd2[13],rd3[13],rd4[13],rd5[13],
rd6[13],rd7[13],rd8[13],rd9[13],rd10,rd11[13],rd12[13],
rd13[13],rd14[13],rd15[13],rd16[13],rd17[13],rd18[13],
rd19[13],rd20[13],rd21,rd22[13],rd23[13],rd24[13],rd25[13],
rd26[13],rd27[13],rd28[13],rd29[13],rd30[13],rd31[13]);
or o14(rd[14],rd0[14],rd1[14],rd2[14],rd3[14],rd4[14],rd5[14],
rd6[14],rd7[14],rd8[14],rd9[14],rd10,rd11[14],rd12[14],
rd13[14],rd14[14],rd15[14],rd16[14],rd17[14],rd18[14],
rd19[14],rd20[14],rd21,rd22[14],rd23[14],rd24[14],rd25[14],
rd26[14],rd27[14],rd28[14],rd29[14],rd30[14],rd31[14]);
or o15(rd[15],rd0[15],rd1[15],rd2[15],rd3[15],rd4[15],rd5[15],
rd6[15],rd7[15],rd8[15],rd9[15],rd10,rd11[15],rd12[15],
rd13[15],rd14[15],rd15[15],rd16[15],rd17[15],rd18[15],
rd19[15],rd20[15],rd21,rd22[15],rd23[15],rd24[15],rd25[15],
```

```
rd26[15],rd27[15],rd28[15],rd29[15],rd30[15]
,rd31[15]);
o                                                r
o16(rd[16],rd0[16],rd1[16],rd2[16],rd3[16],rd
4[16],rd5[16],
rd6[16],rd7[16],rd8[16],rd9[16],rd10,rd11[16
],rd12[16],
rd13[16],rd14[16],rd15[16],rd16[16],rd17[16]
,rd18[16],
rd19[16],rd20[16],rd21,rd22[16],rd23[16],rd2
4[16],rd25[16],
rd26[16],rd27[16],rd28[16],rd29[16],rd30[16]
,rd31[16]);
o                                                r
o17(rd[17],rd0[17],rd1[17],rd2[17],rd3[17],rd
4[17],rd5[17],
rd6[17],rd7[17],rd8[17],rd9[17],rd10,rd11[17
],rd12[17],
rd13[17],rd14[17],rd15[17],rd16[17],rd17[17]
,rd18[17],
rd19[17],rd20[17],rd21,rd22[17],rd23[17],rd2
4[17],rd25[17],
rd26[17],rd27[17],rd28[17],rd29[17],rd30[17]
,rd31[17]);
o                                                r
o18(rd[18],rd0[18],rd1[18],rd2[18],rd3[18],rd
4[18],rd5[18],
rd6[18],rd7[18],rd8[18],rd9[18],rd10,rd11[18
],rd12[18],
rd13[18],rd14[18],rd15[18],rd16[18],rd17[18]
,rd18[18],
rd19[18],rd20[18],rd21,rd22[18],rd23[18],rd2
4[18],rd25[18],
rd26[18],rd27[18],rd28[18],rd29[18],rd30[18]
,rd31[18]);
o                                                r
o19(rd[19],rd0[19],rd1[19],rd2[19],rd3[19],rd
4[19],rd5[19],
rd6[19],rd7[19],rd8[19],rd9[19],rd10,rd11[19
],rd12[19],
rd13[19],rd14[19],rd15[19],rd16[19],rd17[19]
,rd18[19],
rd19[19],rd20[19],rd21,rd22[19],rd23[19],rd2
4[19],rd25[19],
rd26[19],rd27[19],rd28[19],rd29[19],rd30[19]
,rd31[19]);
o                                                r
o20(rd[20],rd0[20],rd1[20],rd2[20],rd3[20],rd
4[20],rd5[20],
rd6[20],rd7[20],rd8[20],rd9[20],rd10,rd11[20
],rd12[20],
rd13[20],rd14[20],rd15[20],rd16[20],rd17[20]
,rd18[20],
rd19[20],rd20[20],rd21,rd22[20],rd23[20],rd2
4[20],rd25[20],
rd26[20],rd27[20],rd28[20],rd29[20],rd30[20]
,rd31[20]);
o                                                r
o21(rd[21],rd0[21],rd1[21],rd2[21],rd3[21],rd
4[21],rd5[21],
rd6[21],rd7[21],rd8[21],rd9[21],rd10,rd11[21
],rd12[21],
rd13[21],rd14[21],rd15[21],rd16[21],rd17[21]
,rd18[21],
rd19[21],rd20[21],rd21,rd22[21],rd23[21],rd2
4[21],rd25[21],
rd26[21],rd27[21],rd28[21],rd29[21],rd30[21]
,rd31[21]);
o                                                r
o22(rd[22],rd0[22],rd1[22],rd2[22],rd3[22],rd
4[22],rd5[22],
rd6[22],rd7[22],rd8[22],rd9[22],rd10,rd11[22
],rd12[22],
rd13[22],rd14[22],rd15[22],rd16[22],rd17[22]
,rd18[22],
rd19[22],rd20[22],rd21,rd22[22],rd23[22],rd2
4[22],rd25[22],
rd26[22],rd27[22],rd28[22],rd29[22],rd30[22]
```

```
,rd31[22]);
o                                                r
o23(rd[23],rd0[23],rd1[23],rd2[23],rd3[23],rd
4[23],rd5[23],
rd6[23],rd7[23],rd8[23],rd9[23],rd10,rd11[23
],rd12[23],
rd13[23],rd14[23],rd15[23],rd16[23],rd17[23]
,rd18[23],
rd19[23],rd20[23],rd21,rd22[23],rd23[23],rd2
4[23],rd25[23],
rd26[23],rd27[23],rd28[23],rd29[23],rd30[23]
,rd31[23]);
o                                                r
o24(rd[24],rd0[24],rd1[24],rd2[24],rd3[24],rd
4[24],rd5[24],
rd6[24],rd7[24],rd8[24],rd9[24],rd10,rd11[24
],rd12[24],
rd13[24],rd14[24],rd15[24],rd16[24],rd17[24]
,rd18[24],
rd19[24],rd20[24],rd21,rd22[24],rd23[24],rd2
4[24],rd25[24],
rd26[24],rd27[24],rd28[24],rd29[24],rd30[24]
,rd31[24]);
o                                                r
o25(rd[25],rd0[25],rd1[25],rd2[25],rd3[25],rd
4[25],rd5[25],
rd6[25],rd7[25],rd8[25],rd9[25],rd10,rd11[25
],rd12[25],
rd13[25],rd14[25],rd15[25],rd16[25],rd17[25]
,rd18[25],
rd19[25],rd20[25],rd21,rd22[25],rd23[25],rd2
4[25],rd25[25],
rd26[25],rd27[25],rd28[25],rd29[25],rd30[25]
,rd31[25]);
o                                                r
o26(rd[26],rd0[26],rd1[26],rd2[26],rd3[26],rd
4[26],rd5[26],
rd6[26],rd7[26],rd8[26],rd9[26],rd10,rd11[26
],rd12[26],
rd13[26],rd14[26],rd15[26],rd16[26],rd17[26]
,rd18[26],
rd19[26],rd20[26],rd21,rd22[26],rd23[26],rd2
4[26],rd25[26],
rd26[26],rd27[26],rd28[26],rd29[26],rd30[26]
,rd31[26]);
o                                                r
o27(rd[27],rd0[27],rd1[27],rd2[27],rd3[27],rd
4[27],rd5[27],
rd6[27],rd7[27],rd8[27],rd9[27],rd10,rd11[27
],rd12[27],
rd13[27],rd14[27],rd15[27],rd16[27],rd17[27]
,rd18[27],
rd19[27],rd20[27],rd21,rd22[27],rd23[27],rd2
4[27],rd25[27],
rd26[27],rd27[27],rd28[27],rd29[27],rd30[27]
,rd31[27]);
o                                                r
o28(rd[28],rd0[28],rd1[28],rd2[28],rd3[28],rd
4[28],rd5[28],
rd6[28],rd7[28],rd8[28],rd9[28],rd10,rd11[28
],rd12[28],
rd13[28],rd14[28],rd15[28],rd16[28],rd17[28]
,rd18[28],
rd19[28],rd20[28],rd21,rd22[28],rd23[28],rd2
4[28],rd25[28],
rd26[28],rd27[28],rd28[28],rd29[28],rd30[28]
,rd31[28]);
o                                                r
o29(rd[29],rd0[29],rd1[29],rd2[29],rd3[29],rd
4[29],rd5[29],
rd6[29],rd7[29],rd8[29],rd9[29],rd10,rd11[29
],rd12[29],
rd13[29],rd14[29],rd15[29],rd16[29],rd17[29]
,rd18[29],
rd19[29],rd20[29],rd21,rd22[29],rd23[29],rd2
4[29],rd25[29],
rd26[29],rd27[29],rd28[29],rd29[29],rd30[29]
,rd31[29]);
```

```
o                                                   r
o30(rd[30],rd0[30],rd1[30],rd2[30],rd3[30],rd
4[30],rd5[30],

rd6[30],rd7[30],rd8[30],rd9[30],rd10,rd11[30
],rd12[30],

rd13[30],rd14[30],rd15[30],rd16[30],rd17[30]
,rd18[30],

rd19[30],rd20[30],rd21,rd22[30],rd23[30],rd2
4[30],rd25[30],

rd26[30],rd27[30],rd28[30],rd29[30],rd30[30]
,rd31[30]);
o                                                   r
o31(rd[31],rd0[31],rd1[31],rd2[31],rd3[31],rd
4[31],rd5[31],

rd6[31],rd7[31],rd8[31],rd9[31],rd10,rd11[31
],rd12[31],

rd13[31],rd14[31],rd15[31],rd16[31],rd17[31]
,rd18[31],

rd19[31],rd20[31],rd21,rd22[31],rd23[31],rd2
4[31],rd25[31],

rd26[31],rd27[31],rd28[31],rd29[31],rd30[31]
,rd31[31]);
endmodule



module decoder(
output [31:0]out,
input [4:0]in
    );//Êä³öµÍµçÆ½ÓÐÐ§
wire [4:0]_in;
not n1(_in[0],in[0]);
not n2(_in[1],in[1]);
not n3(_in[2],in[2]);
not n4(_in[3],in[3]);
not n5(_in[4],in[4]);
n           a           n           d
a0(out[0],_in[0],_in[1],_in[2],_in[3],_in[4]);
n           a           n           d
a1(out[1],in[0],_in[1],_in[2],_in[3],_in[4]);
n           a           n           d
a2(out[2],_in[0],in[1],_in[2],_in[3],_in[4]);
n           a           n           d
a3(out[3],in[0],in[1],_in[2],_in[3],_in[4]);
n           a           n           d
a4(out[4],_in[0],_in[1],in[2],_in[3],_in[4]);
n           a           n           d
a5(out[5],in[0],_in[1],in[2],_in[3],_in[4]);
n           a           n           d
a6(out[6],_in[0],in[1],in[2],_in[3],_in[4]);
nand a7(out[7],in[0],in[1],in[2],_in[3],_in[4]);
n           a           n           d
a8(out[8],_in[0],_in[1],_in[2],in[3],_in[4]);
n           a           n           d
a9(out[9],in[0],_in[1],_in[2],in[3],_in[4]);
n           a           n           d
a10(out[10],_in[0],in[1],_in[2],in[3],_in[4]);
n           a           n           d
a11(out[11],in[0],in[1],_in[2],in[3],_in[4]);
n           a           n           d
a12(out[12],_in[0],_in[1],in[2],in[3],_in[4]);
n           a           n           d
a13(out[13],in[0],_in[1],in[2],in[3],_in[4]);
n           a           n           d
a14(out[14],_in[0],in[1],in[2],in[3],_in[4]);
n           a           n           d
a15(out[15],in[0],in[1],in[2],in[3],_in[4]);
n           a           n           d
a16(out[16],_in[0],_in[1],_in[2],_in[3],in[4]);
n           a           n           d
a17(out[17],in[0],_in[1],_in[2],_in[3],in[4]);
n           a           n           d
a18(out[18],_in[0],in[1],_in[2],_in[3],in[4]);
n           a           n           d
a19(out[19],in[0],in[1],_in[2],_in[3],in[4]);
n           a           n           d
a20(out[20],_in[0],_in[1],in[2],_in[3],in[4]);
n           a           n           d
a21(out[21],in[0],_in[1],in[2],_in[3],in[4]);
n           a           n           d
a22(out[22],_in[0],in[1],in[2],_in[3],in[4]);
n           a           n           d
a23(out[23],in[0],in[1],in[2],_in[3],in[4]);
n           a           n           d
a24(out[24],_in[0],_in[1],_in[2],in[3],in[4]);
n           a           n           d
a25(out[25],in[0],_in[1],_in[2],in[3],in[4]);
n           a           n           d
a26(out[26],_in[0],in[1],_in[2],in[3],in[4]);
n           a           n           d
a27(out[27],in[0],in[1],_in[2],in[3],in[4]);
n           a           n           d
a28(out[28],_in[0],_in[1],in[2],in[3],in[4]);
n           a           n           d
a29(out[29],in[0],_in[1],in[2],in[3],in[4]);
n           a           n           d
a30(out[30],_in[0],in[1],in[2],in[3],in[4]);
nand a31(out[31],in[0],in[1],in[2],in[3],in[4]);
```

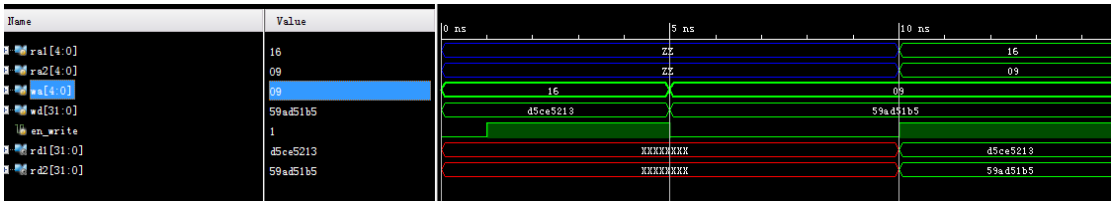
```
endmodule



module write(
    input       en_write,//µÍµçÆ½ÓÐÐ§£¬¿ªÊ¼½«data´ÈëÄ´æÆ÷
    input [5:1]reg_num,
    output [31:0]w2
    );
wire [31:0]w1;
decoder d0(w1,reg_num);
nor nr1(w2[0],en_write,w1[0]);
nor nr2(w2[1],en_write,w1[1]);
nor nr3(w2[2],en_write,w1[2]);
nor nr4(w2[3],en_write,w1[3]);
nor nr5(w2[4],en_write,w1[4]);
nor nr6(w2[5],en_write,w1[5]);
nor nr7(w2[6],en_write,w1[6]);
nor nr8(w2[7],en_write,w1[7]);
nor nr9(w2[8],en_write,w1[8]);
nor nr10(w2[9],en_write,w1[9]);
nor nr11(w2[10],en_write,w1[10]);
nor nr12(w2[11],en_write,w1[11]);
nor nr13(w2[12],en_write,w1[12]);
nor nr14(w2[13],en_write,w1[13]);
nor nr15(w2[14],en_write,w1[14]);
nor nr16(w2[15],en_write,w1[15]);
nor nr17(w2[16],en_write,w1[16]);
nor nr18(w2[17],en_write,w1[17]);
nor nr19(w2[18],en_write,w1[18]);
nor nr20(w2[19],en_write,w1[19]);
nor nr21(w2[20],en_write,w1[20]);
nor nr22(w2[21],en_write,w1[21]);
nor nr23(w2[22],en_write,w1[22]);
nor nr24(w2[23],en_write,w1[23]);
nor nr25(w2[24],en_write,w1[24]);
nor nr26(w2[25],en_write,w1[25]);
nor nr27(w2[26],en_write,w1[26]);
nor nr28(w2[27],en_write,w1[27]);
nor nr29(w2[28],en_write,w1[28]);
nor nr30(w2[29],en_write,w1[29]);
nor nr31(w2[30],en_write,w1[30]);
endmodule

module read(
    input [4:0]ra1,ra2,
    input [31:0]data0,
    input [31:0]data1,
    input [31:0]data2,
    input [31:0]data3,
    input [31:0]data4,
    input [31:0]data5,
    input [31:0]data6,
    input [31:0]data7,
    input [31:0]data8,
    input [31:0]data9,
    input [31:0]data10,
    input [31:0]data11,
    input [31:0]data12,
    input [31:0]data13,
    input [31:0]data14,
    input [31:0]data15,
    input [31:0]data16,
    input [31:0]data17,
    input [31:0]data18,
    input [31:0]data19,
    input [31:0]data20,
    input [31:0]data21,
    input [31:0]data22,
    input [31:0]data23,
    input [31:0]data24,
    input [31:0]data25,
    input [31:0]data26,
    input [31:0]data27,
    input [31:0]data28,
    input [31:0]data29,
    input [31:0]data30,
    input [31:0]data31,
    output [31:0]rd1,rd2
    );
Mux m1(data0,data1,data2,data3,data4,
    data5,data6,data7,data8,data9,data10,
    data11,data12,data13,data14,data15,
    data16,data17,data18,data19,data20,
    data21,data22,data23,data24,data25,
    data26,data27,data28,data29,data30,
    data31,ra1,rd1);
Mux m2(data0,data1,data2,data3,data4,
    data5,data6,data7,data8,data9,data10,
    data11,data12,data13,data14,data15,
    data16,data17,data18,data19,data20,
    data21,data22,data23,data24,data25,
    data26,data27,data28,data29,data30,
    data31,ra2,rd2);
endmodule

module RegisterFile(
    input [4:0]ra1,ra2,wa,
    input [31:0]wd,
    input en_write,//µÍµçÆ½ÓÐÐ§
    output [31:0]rd1,rd2
    );
wire [31:0]w2;
wire [31:0]data0,data1,data2,data3,data4,
    data5,data6,data7,data8,data9,data10,
```

```
    data11,data12,data13,data14,data15,
    data16,data17,data18,data19,data20,
    data21,data22,data23,data24,data25,
    data26,data27,data28,data29,data30,
    data31;
write w1(en_write,wa,w2);
r e a d
r0(ra1,ra2,data0,data1,data2,data3,data4,
    data5,data6,data7,data8,data9,data10,
    data11,data12,data13,data14,data15,
    data16,data17,data18,data19,data20,
    data21,data22,data23,data24,data25,
    data26,data27,data28,data29,data30,
    data31,rd1,rd2);
Dlatch32 d0(wd,w2[0],data0);
Dlatch32 d1(wd,w2[1],data1);
Dlatch32 d2(wd,w2[2],data2);
Dlatch32 d3(wd,w2[3],data3);
Dlatch32 d4(wd,w2[4],data4);
Dlatch32 d5(wd,w2[5],data5);
Dlatch32 d6(wd,w2[6],data6);
Dlatch32 d7(wd,w2[7],data7);
Dlatch32 d8(wd,w2[8],data8);
Dlatch32 d9(wd,w2[9],data9);
Dlatch32 d10(wd,w2[10],data10);
Dlatch32 d11(wd,w2[11],data11);
Dlatch32 d12(wd,w2[12],data12);
Dlatch32 d13(wd,w2[13],data13);
Dlatch32 d14(wd,w2[14],data14);
Dlatch32 d15(wd,w2[15],data15);
Dlatch32 d16(wd,w2[16],data16);
Dlatch32 d17(wd,w2[17],data17);
Dlatch32 d18(wd,w2[18],data18);
Dlatch32 d19(wd,w2[19],data19);
Dlatch32 d20(wd,w2[20],data20);
Dlatch32 d21(wd,w2[21],data21);
Dlatch32 d22(wd,w2[22],data22);
Dlatch32 d23(wd,w2[23],data23);
Dlatch32 d24(wd,w2[24],data24);
Dlatch32 d25(wd,w2[25],data25);
Dlatch32 d26(wd,w2[26],data26);
Dlatch32 d27(wd,w2[27],data27);
Dlatch32 d28(wd,w2[28],data28);
Dlatch32 d29(wd,w2[29],data29);
Dlatch32 d30(wd,w2[30],data30);
Dlatch32 d31(wd,w2[31],data31);
endmodule
```

仿真代码：

```
module RegisterFile_tb;
reg [4:0]ra1,ra2,wa;
reg [31:0]wd;
reg en_write;
wire [31:0]rd1,rd2;
RegisterFile rf(ra1,ra2,wa,wd,en_write,rd1,rd2);
initial begin
    en_write=0;
    wd=32'b11010101110011100101001000010011;
    wa=5'b10110;
    ra1=5'bz;
    ra2=5'bz;
    #1en_write=1;
    #4 wa=5'b01001;en_write=0;
    wd=32'b01011001101011010101000110110101;
    #5 en_write=1;
    ra1=5'b10110;
    ra2=5'b01001;
    #5 $finish;
end
endmodule
```

1.2.3 仿真波形



1.3 结果分析

如图所示，顺序在两个不同的地址写入两个不同的值，输出时可以成功获得两个输入的值。完成了实验所需的要求。

2 实验2 32位数据存储器设计

2.1实验内容

用Vivado软件和VerilogHDL或VHDL语言设计实现32位数据存储器，要求：

支持写入存储器32位数据；

支持从存储器读出32位数据；

该存储器实为CPU内的高速缓存，假定容量为32*32bit。建议位数用参数形式来定义。

如图2-1所示，实现圆圈内所示电路。输入输出变量名字尽量与图示保持一致。

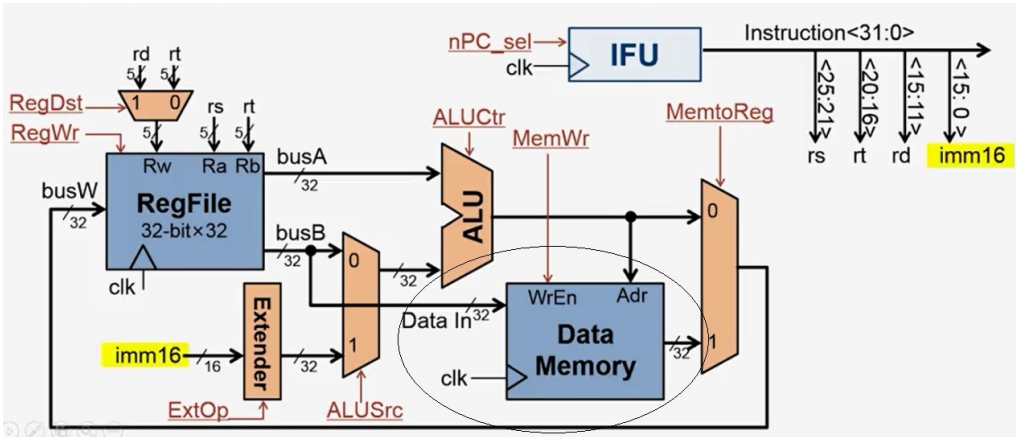


图2-1

2.2主要步骤

2.2.1 分析实现的方法

存储器有两组数据接口信号，有 32 位的数据输入和 32 位的数据输出。存储器的读写和寄存器堆有些类似，只要给出一个地址，存储器就会将对应地址单元中的数送到数据输出信号上，而与寄存器堆不同在于只需给出一组输入信号，而不是寄存器堆的两组数据输入，从这个角度来说，存储器是一个一读一写的存储器。另一个方面，我们给出的地址信号是 32 位的，所以理论上存储器可以达到 2 的 32 次方那么大，当然这只是理想化的情况，对于存储器的写操作，也供了一个写使能信号，在时钟上升沿到来时，如果写使能为 1，存储器就会将数据输入

信号的内容存入地址信号所指定的存储单元中。同样需要注意，存储器的写操作是在时钟上升沿发生的，而存储器的读操作不受时钟信号的控制。 只要输入的地址信号发生变化，需要很短的时间输出的数据信号随之发生变化。

2.2.2 实现代码和仿真代码

实现代码：

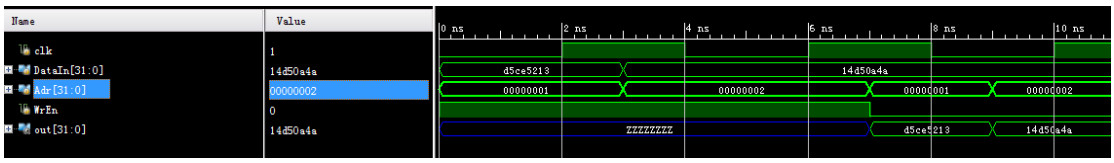
```
module DataMemory(
      input clk,
      input [31:0]DataIn,
      input WrEn,
      input [31:0]Adr,
      output [31:0]Out
      );
reg [31:0]DM[5:0];
assign Out=WrEn?32'hzzzzzzzz:DM[Adr];
integer i;
initial begin
  for(i=0;i<32;i=i+1)
    DM[i]<=0;
end

always @(posedge clk) begin
      if(WrEn)
            DM[Adr]<=DataIn;
end
endmodule
```

仿真代码：

```
module DataMemory_tb;
reg clk;
reg [31:0]DataIn,Adr;
reg WrEn;
wire [31:0]out;
DataMemory t1(clk,DataIn,WrEn,Adr,out);
initial begin
      WrEn=1;
      DataIn=32'b11010101110011100101001000010011;
      Adr=1;
      clk=0;
      #3 Adr=2;
      DataIn=32'b01010011010101000001010010010010;
      #4 WrEn=0; Adr=1;
      #2 Adr=2;
      #2 $finish;
end
always  begin
```

```
    #2 clk=~clk;
end
endmodule
```

2.2.3 仿真波形



2.3 结果分析

由于无法设计成理想中状态中2的32次方那么大，最后只设计成6*32位存储器。由图可以看出，顺序输入存入两个不同的值，在输出时，顺序改变地址，输出端也能得到输入的两个值。能满足任务要求。

3、实验3 Cache控制器设计（加分）

3.1 实验内容

用Vivado软件和VerilogHDL或VHDL语言设计实现32位Cache控制器，要求：

Cache控制器内区号块号和块内大小自行设计，图3-1仅供参考；

实现直接相联映像电路。

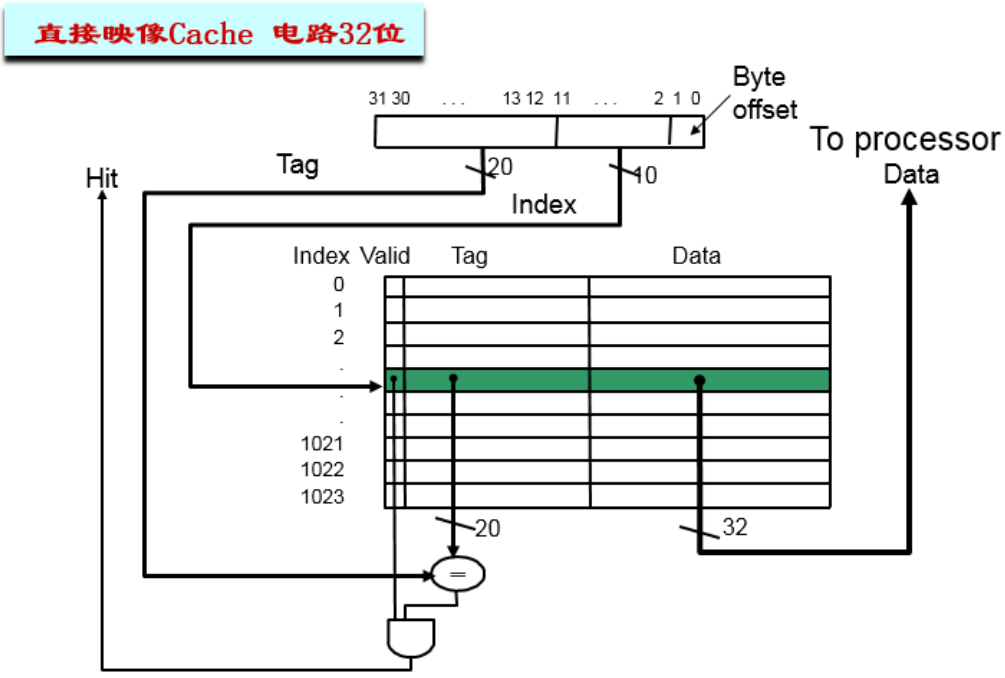


图3-1

3.2 主要步骤

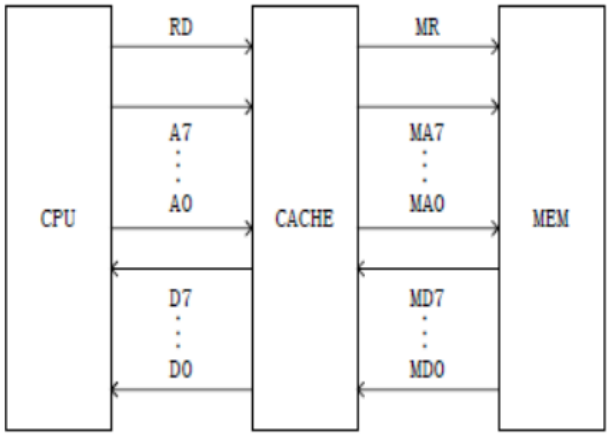
3.2.1 分析实现的方法

(1)使用D触发器作为存储一个二进制代码的存储单元，设计一个8位的存储单元模块使用这个8位的存储单元构成一个32*8位的Cache存储体。

(2)实现一个4位的存储单元模块；然后使用这个 4 位存储单元构成一个8*4位的区表存储器，用来存放区号和有效位。由图2可知，还需要实现一个区号比较器，如果主存地址的区号 E 和区表中相应单元中的区号相等，且有效位为 1 ，则Cache命中，否则Cache失效，标志位为M，M为0时表示Cache失效。

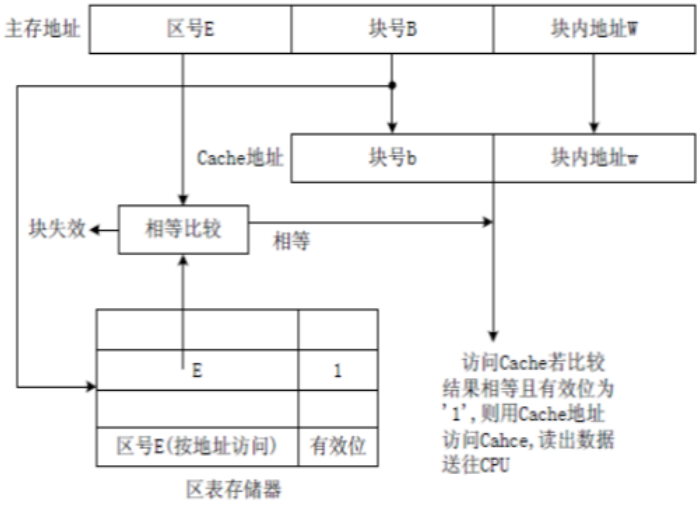
(3)当Cache命中时，就将Cache存储体中相应单元的数据送往CPU；当Cache失效时，需要将主存中相应块中的数据读出写入Cache中，这样Cache控制器就要产生访问主存储器的地址和主存储器的读信号，由于每块占4个单元，所以需要连续访问4次主存，这就需要一个低地址发生器。用一个循环语句实现四次访问

(4)将产生的低2位地址和CPU给出的高6位地址组合起来，形成访问主存储器的地址，M可以作为主存的读信号。这样在时钟T2的控制下，就可以将主存中相应的块写入Cache的相应块中，最后再修改区表。



CPU 读内存时 和 CACHE MEM之间的连接信号

本实验直接映像Cache 电路



3.2.2 实现代码和仿真代码

区表存储模块

```
module AreaMem(dataout,datain,address,CLR,RW);
  output [3:0] dataout;
  input [3:0] datain;
  input [2:0] address;
  input CLR;
  input RW;

  reg [3:0] data1,data2,data3,data4,
          data5,data6,data7,data8;
  reg [2:0] addr;
  reg [3:0] dataout;

  always @(*)
  begin
    if(CLR==1)
      begin
        data1 = 0;
        data2 = 0;
        data3 = 0;
        data4 = 0;
        data5 = 0;
        data6 = 0;
        data7 = 0;
        data8 = 0;
        dataout = 0;
      end
    else if(RW==0)
      case (addr)
        3'b000:data1=datain;
        3'b001:data2=datain;
        3'b010:data3=datain;
        3'b011:data4=datain;
        3'b100:data5=datain;
        3'b101:data6=datain;
        3'b110:data7=datain;
        3'b111:data8=datain;
        default:;
      endcase
    else if(RW==1)
      case (addr)
        3'b000:dataout=data1;
        3'b001:dataout=data2;
        3'b010:dataout=data3;
        3'b011:dataout=data4;
        3'b100:dataout=data5;
        3'b101:dataout=data6;
        3'b110:dataout=data7;
        3'b111:dataout=data8;
        default:;
```

```
    endcase
  end
```

Cache储存器

```
m o d u l e CacheMem(dataout,datain,address,CLR,RW);
  output [7:0] dataout;
  input [7:0] datain;
  input [4:0] address;
  input CLR;
  input RW;     //read1,write0
                r e g [ 7 : 0 ] data0,data1,data2,data3,data4,data5,data6,data7,

data8,data9,data10,data11,data12,data13,data14,data15,

data16,data17,data18,data19,data20,data21,data22,data23,

data24,data25,data26,data27,data28,data29,data30,data31;
  reg [4:0] addr;
  reg [7:0] dataout;

  always @(*)
  begin
    if(CLR==1)
    begin
      data0 = 0;
      data1 = 0;
      data2 = 0;
      data3 = 0;
      data4 = 0;
      data5 = 0;
      data6 = 0;
      data7 = 0;
      data8 = 0;
      data9 = 0;
      data10 = 0;
      data11 = 0;
      data12 = 0;
      data13 = 0;
      data14 = 0;
      data15 = 0;
      data16 = 0;
      data17 = 0;
      data18 = 0;
      data19 = 0;
      data20 = 0;
      data21 = 0;
      data22 = 0;
      data23 = 0;
      data24 = 0;
      data25 = 0;
      data26 = 0;
      data27 = 0;
      data28 = 0;
      data29 = 0;
      data30 = 0;
      data31 = 0;
    end
    else if(RW==0)
    begin
      addr=address;
      case (addr)
        5'b00000:data0=datain;
        5'b00001:data1=datain;
        5'b00010:data2=datain;
        5'b00011:data3=datain;
        5'b00100:data4=datain;
        5'b00101:data5=datain;
        5'b00110:data6=datain;
        5'b00111:data7=datain;
        5'b01000:data8=datain;
        5'b01001:data9=datain;
        5'b01010:data10=datain;
        5'b01011:data11=datain;
        5'b01100:data12=datain;
        5'b01101:data13=datain;
        5'b01110:data14=datain;
        5'b01111:data15=datain;
endmodule
```

```
        5'b10000:data16=datain;
        5'b10001:data17=datain;
        5'b10010:data18=datain;
        5'b10011:data19=datain;
        5'b10100:data20=datain;
        5'b10101:data21=datain;
        5'b10110:data22=datain;
        5'b10111:data23=datain;
        5'b11000:data24=datain;
        5'b11001:data25=datain;
        5'b11010:data26=datain;
        5'b11011:data27=datain;
        5'b11100:data28=datain;
        5'b11101:data29=datain;
        5'b11110:data30=datain;
        5'b11111:data31=datain;
        default:;
      endcase
    end
    else if(RW==1)
    begin
      addr=address;
      case (addr)
        5'b00000:dataout=data0;
        5'b00001:dataout=data1;
        5'b00010:dataout=data2;
        5'b00011:dataout=data3;
        5'b00100:dataout=data4;
        5'b00101:dataout=data5;
        5'b00110:dataout=data6;
        5'b00111:dataout=data7;
        5'b01000:dataout=data8;
        5'b01001:dataout=data9;
        5'b01010:dataout=data10;
        5'b01011:dataout=data11;
        5'b01100:dataout=data12;
        5'b01101:dataout=data13;
        5'b01110:dataout=data14;
        5'b01111:dataout=data15;
        5'b10000:dataout=data16;
        5'b10001:dataout=data17;
        5'b10010:dataout=data18;
        5'b10011:dataout=data19;
        5'b10100:dataout=data20;
        5'b10101:dataout=data21;
        5'b10110:dataout=data22;
        5'b10111:dataout=data23;
        5'b11000:dataout=data24;
        5'b11001:dataout=data25;
        5'b11010:dataout=data26;
        5'b11011:dataout=data27;
        5'b11100:dataout=data28;
        5'b11101:dataout=data29;
        5'b11110:dataout=data30;
        5'b11111:dataout=data31;
        default:;
      endcase
    end
  end
endmodule
```

比较模块

```
module Compare(
    input               CLR,
    input       [2:0]   Compare_addr,
    input       [2:0]   Compare_in,
    input               Area_RW,
    output              M
    );

    wire    [3:0]   Area_out;
    reg     [2:0]   Area_addr;
    wire            V;

    assign V = Area_out[0];
    assign M = ~(Area_out[3:1] == Compare_addr[2:0]);

    AreaMem U3 (Area_out,Compare_in,Compare_addr,CLR, Area_RW);  //read1,write0

endmodule
```

主控模块

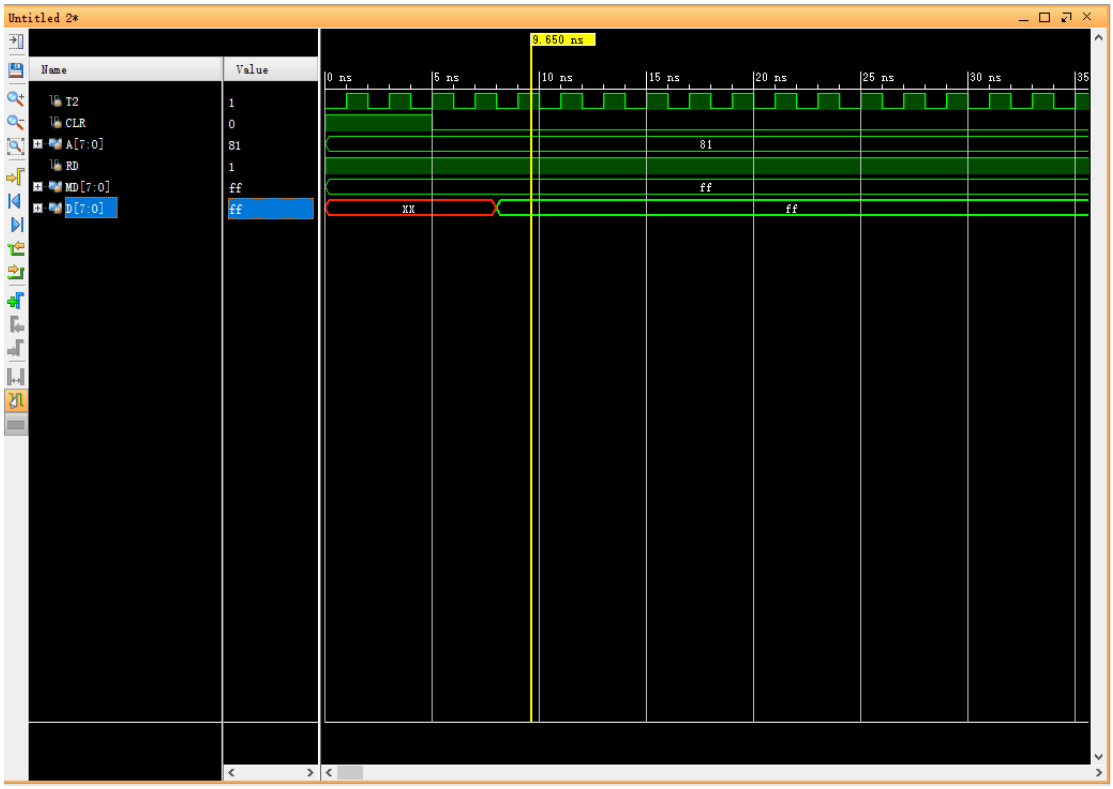
```
module    Cache(
    // 控制信号
    input               T2,                 // 时钟信号
    input               CLR,                // 系统清零信号
    // 与主存通信信号
    input     [7:0] MD,                     // 读数据
    // 与cpu通信信号
    input     [7:0] A,                      // cpu给出的主存地址
    input               RD,                 // cpu访问主存读信号
    output reg  [7:0] D                     // Cache送cpu数据
    );

    parameter           X1 = 0;
    parameter           X2 = 1;

    reg                 Y;                  //状态机
    wire                M;                  //Cache失效信号
    wire      [7:0] Cache_out;
    reg         [7:0] Cache_in;
    reg         [4:0] Cache_addr;           //Cache地址，即CA
    wire                Cache_RW;           //Cache读写
    wire                Area_RW;            //Area读写
    reg         [1:0] LA;                   //块内地址
    wire      [2:0] Compare_addr;
    reg         [2:0] Compare_in;


    always@(posedge T2)
    begin
        if(!CLR)
        begin
            if(!M)                          //从Cache送入cpu
            begin
                Cache_addr = A[4:0];
                D = Cache_out;
            end
            else if(RD)                     // 从主存取数据放入
            begin
                Compare_in <= MD[7:5];
                for(LA = 0; LA < 4 ; LA = LA + 1)
                begin
                    Cache_addr = {A[4:0],LA[1:0]};
                    Cache_in = MD;
                end
            end

        end
    end


    always@(posedge T2)                     //状态机
    begin
```

```
        if(CLR)
            Y    <=   X1;
        else
            case(Y)
                X1:  if(M)
                         Y    <=   X2;
                     else
                         Y    <=   X1;
                X2:  if(!M)
                         Y    <=   X1;
                     else
                         Y    <=   X2;
                default:  Y   <=   X1;
            endcase
    end

    assign Compare_addr = A[4:2];          //输入AreaMem的地址一直是cpu请求内存的地址
    assign Cache_RW = ~(Y == X2);
    assign Area_RW = ~(Y == X2);           //从主存写入AreaMem
    assign RD =      Y == X2 ;

    Compare      U1 (CLR,Compare_addr,Compare_in,Area_RW,M);
    CacheMem     U2 (Cache_out,Cache_in,Cache_addr,CLR,Area_RW);

endmodule
```

测试激励：

```
module Cache_tb();
    reg                 T2;
    reg                 CLR;
    reg   [7:0] A;
    reg                 RD;
    wire  [7:0] D1;
    reg         [7:0] MD;
    reg     [7:0]   D;
    initial
    begin
        T2=0;
        forever #1 T2=~T2;
    end
    initial
    begin
        CLR=1;
        A = 8'b1000_0001;
        RD = 1;
        MD = 8'b1111_1111;
        #5 CLR=0;
    end
    Cache   U3(T2, CLR, MD, A ,RD, D1);
endmodule
```

3.2.3 仿真波形



3.3 结果分析

由图可看出， Cache初始化之后全部置零，首先会比较区块号，然后判断不匹配，从测试模块读取模拟主存给Cache的数据MD：ffH，读入数据并重置区块

4 实验4 IFU设计

4.1实验内容

用Vivado软件和VerilogHDL或VHDL语言设计实现32位取指令部件，要求：

支持指令的连续读取；

支持任意给定目标地址指令的读取。

如图4-1所示，实现方框内所示电路。输入输出变量名字尽量与图示保持一致。

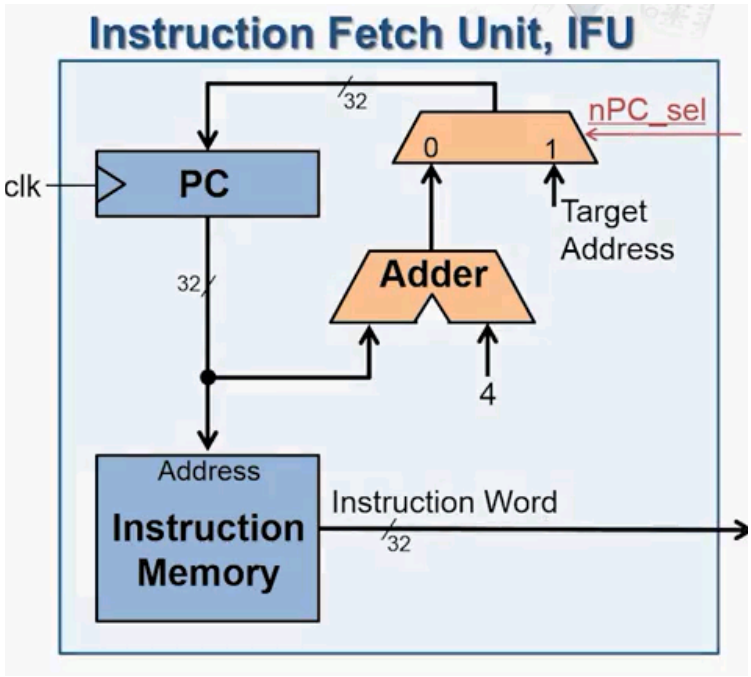


图4-1

4.2主要步骤

4.2.1 分析实现的方法

指令大多顺序执行，PC 只需要加上当前这条指令的长度就可以得到下一条指令的地址，即进行 PC 加 4 的运算，增加一个加法器，PC 的输出连接到加法器的一个输入端，另一端为常数 4， PC 的输出一方面送到指令寄存器输入端，另一方面送到加法器计算出 PC 加 4 的值，在下一个时钟上升沿到来时，PC 寄存器中就会将 PC 加 4 的值存入其中，然后再将这个更新后 的内容同时送到指令存储器和加法器，如此周而复始就完成了每个时钟上升沿更新 PC 寄存 器的内容，然后指令存储器就会送出新一条指令的二进制编码。 如果遇到分支情况，下一条指令地址就不是 PC 加 4，而是由分支指令进行指定。增加一个二选一的多选器，在顺序执行时，选择 0 输入端输入，在分支执行时，选择 1 输入端的输入。在下一个时钟上升沿到来时，PC寄存器就会采样多选器的输出，并将其保存起来，这样的结构，就完成了不断取回指令的功能。

4.2.2 实现代码和仿真代码

实现代码：

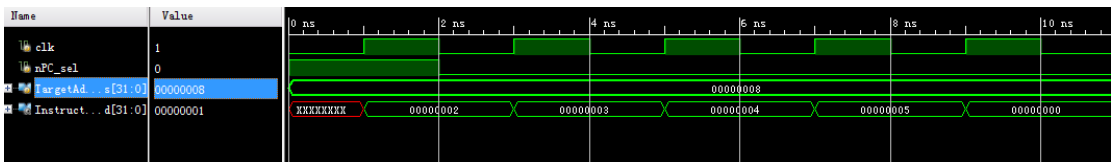
```
module IFU(
      input clk,
      input nPC_sel,
      input [31:0]TargetAddress,
      output [31:0]InstructionWord
      );
wire [31:0]pc;
reg [31:0]pc_clk;
```

```
wire [31:0]adderout;
reg [31:0]InstructionMem[31:0];

integer i;
initial begin
  for(i=0;i<32;i=i+1)
    InstructionMem[i]<=i;
end
assign pc = nPC_sel?TargetAddress:adderout;

always @(posedge clk) begin
      pc_clk=pc;
      if(pc_clk>=24) pc_clk=pc_clk%4;
end

assign adderout = pc_clk+4;
assign InstructionWord = InstructionMem[pc_clk/4];
endmodule
```

仿真代码：

```
module IFU_tb;
reg clk,nPC_sel;
reg [31:0]TargetAddress;
wire [31:0]InstructionWord;

IFU u1(clk,nPC_sel,TargetAddress,InstructionWord);

initial begin
      clk=0;
      nPC_sel=1;
      TargetAddress=1;
      #2 nPC_sel=0;
      #20 $finish;
end

always  begin
      #1 clk=~clk;
end

endmodule
```

4.2.3 仿真波形



4.3 结果分析

如图所示，在nPC_sel有效时，进行置数，由于非理想情况，所以指令寄存器的大小设置成只能存放六条指令，当置数为8的时候，实际置入的是8mod6=2.所以从2开始增加指令，增加到5之后变成0进行循环。满足题目要求。

5 实验5 ALU设计

5.1实验内容

用Vivado软件和VerilogHDL或VHDL语言设计实现32位ALU部件，要求：

支持无符号加法、减法、逻辑或运算等；

加法部分采用超前进位方法实现。

5.2主要步骤

5.2.1 分析实现的方法

输入了一个两位选择运算类型的输入端，两个32位操作数的输入端，一个32位结果输出端。并行加法器之间亦为并行相连。

5.2.2 实现代码或用LPM的实现步骤和仿真代码（如用代码仿真增加此项）

实验代码：

```
module ALU(
    input [1:0]ALUCtr,
    input [31:0]busA,busB,
    output [31:0]busW
    );
wire [31:0]b;
wire  [31:0]subb,s,s2;
assign subb=(~busB)-1;
assign b = ALUCtr[0]?subb:busB;
assign cin=0;
carrylookahead32 adder(busA,b,cin,cout,s);
assign s2 = busA||busB;
assign busW = ALUCtr[1]?s2:s;//ALUCtr=00Î³¼Ó£¬=01Î³¼õ£¬=10Î³Âß¼»òÔËã¡£
endmodule

module carrylookahead4(
    input [3:0]a,b,
    input cin,
    output [3:0]s,
    output Pm,Gm
    );
wire [3:0]G,P,C;
assign G = a&b;
assign P = a^b;
```

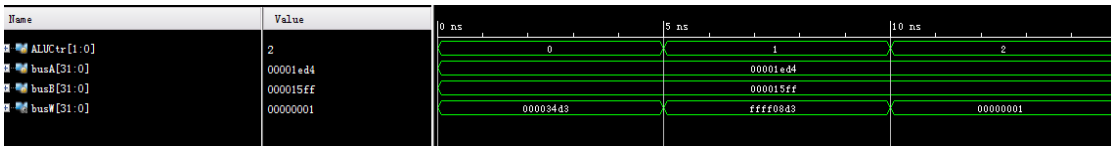
```
assign C[0]=cin;
assign C[1] = G[0]|P[0]&C[0];
assign C[2] = G[1]|P[1]&G[0]|P[1]&P[0]&C[0];
assign C[3] = G[2]|P[2]&G[1]|P[2]&P[1]&G[0]|P[2]&P[1]&P[0]&C[0];
assign s = P^C;
assign Pm = &P;
assign Gm = G[3]|P[3]&G[2]|P[3]&P[2]&G[1]|P[3]&P[2]&P[1]&G[0];
endmodule

module carrylookahead32(
    input [31:0]a,b,
    input cin,
    output cout,
    output [31:0]s
    );
wire [7:0]C;
wire [8:1]Pm,Gm;
wire Pmx,Gmx;
assign C[0]=cin;
assign C[1]=Gm[1]|Pm[1]&C[0];
assign C[2]=Gm[2]|Pm[2]&Gm[1]|Pm[2]&Pm[1]&C[0];
assign C[3]=Gm[3]|Pm[3]&Gm[2]|Pm[3]&Pm[2]&Gm[1]|Pm[3]&Pm[2]&Pm[1]&C[0];
carrylookahead4 c1(a[3:0],b[3:0],cin,s[3:0],Pm[1],Gm[1]);
carrylookahead4 c2(a[7:4],b[7:4],C[1],s[7:4],Pm[2],Gm[2]);
carrylookahead4 c3(a[11:8],b[11:8],C[2],s[11:8],Pm[3],Gm[3]);
carrylookahead4 c4(a[15:12],b[15:12],C[3],s[15:12],Pm[4],Gm[4]);
assign Gmx=Gm[4]|Pm[4]&Gm[3]|Pm[4]&Pm[3]&Gm[2]|Pm[4]&Pm[3]&Pm[2]&Gm[1];
assign Pmx=Pm[4]&Pm[3]&Pm[2]&Pm[1];
assign C[4]=Gmx+Pmx&C[0];
assign C[5]=Gm[5]|Pm[5]&C[4];
assign C[6]=Gm[6]|Pm[6]&Gm[5]|Pm[6]&Pm[5]&C[4];
assign C[7]=Gm[7]|Pm[7]&Gm[6]|Pm[7]&Pm[6]&Gm[5]|Pm[7]&Pm[6]&Pm[5]&C[4];
carrylookahead4 c5(a[19:16],b[19:16],C[4],s[19:16],Pm[5],Gm[5]);
carrylookahead4 c6(a[23:20],b[23:20],C[5],s[23:20],Pm[6],Gm[6]);
carrylookahead4 c7(a[27:24],b[27:24],C[6],s[27:24],Pm[7],Gm[7]);
carrylookahead4 c8(a[31:28],b[31:28],C[7],s[31:28],Pm[8],Gm[8]);
assign cout=Gm[8]|Pm[8]&Gm[7]|Pm[8]&Pm[7]&Gm[6]|Pm[8]&Pm[7]&Pm[6]&Gm[5]|Pm[8]&
Pm[7]&Pm[6]&Pm[5]&C[4];
endmodule
```

仿真代码：

```
module ALU_tb;
reg [1:0]ALUCtr;
reg [31:0]busA,busB;
wire [31:0]busW;
initial begin
    busA=7892;
```

```
        busB=5631;
        ALUCtr=00;
        #5 ALUCtr=01;
        #5 ALUCtr=10;
        #5 $finish;
end
ALU u0(ALUCtr,busA,busB,busW);
endmodule
```

5.2.3 仿真波形



5.3 结果分析

输入两个操作数，分别实现了加法、减法和或运算，由图，运算结果均正确，实现了程序的基本要求。

6 实验6硬布线控制器设计

6.1实验内容

用Vivado软件和VerilogHDL或VHDL语言设计实现硬布线控制器部件，要求：

输入信号和输出信号如图6-1所示；

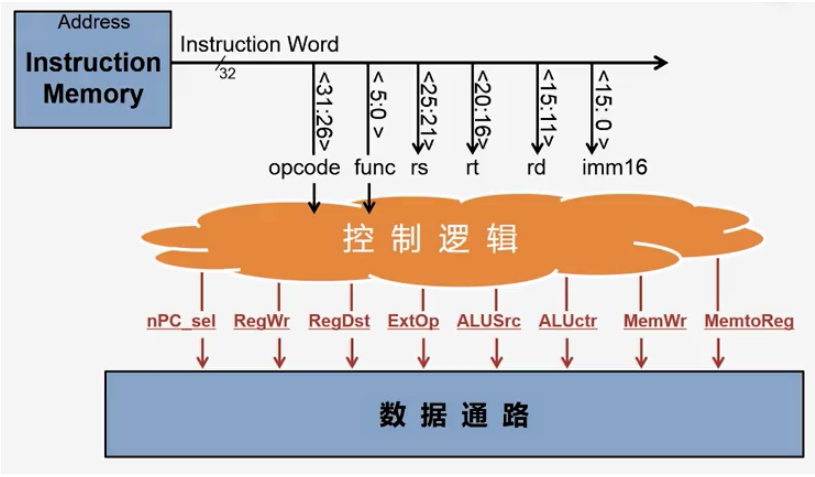


图6-1

6.2主要步骤

6.2.1 分析实现的方法

func	100000	100010	/			
opcode(op)	000000	000000	001101	100011	101011	000100
	add	sub	ori	lw	sw	beq
RegDst	1	1	0	0	x	x
ALUSrc	0	0	1	1	1	0
MemtoReg	0	0	0	1	x	x
RegWr	1	1	1	1	0	0
MemWr	0	0	0	0	1	0
nPC_sel	0	0	0	0	0	1
ExtOp	x	x	0	1	1	x
ALUctr<1:0>	00(ADD)	01(SUB)	10(OR)	00(ADD)	00(ADD)	01(SUB)

图 3-27 六条指令的控制信号

```
RegDst    = add + sub
ALUSrc    = ori + lw + sw
MemtoReg  = lw
RegWr     = add + sub + ori + lw
MemWr     = sw
nPC_sel   = beq
ExtOp     = lw + sw
ALUctr[0] = sub + beq
ALUctr[1] = or

add   = rtype·func5·~func4·~func3·~func2·~func1·~func0
sub   = rtype·func5·~func4·~func3·~func2· func1·~func0
rtype = ~op5·~op4·~op3·~op2·~op1·~op0,
ori   = ~op5·~op4· op3· op2·~op1· op0
lw    =  op5·~op4·~op3·~op2· op1· op0
sw    =  op5·~op4· op3·~op2· op1· op0
beq   = ~op5·~op4·~op3· op2·~op1·~op0
```

图 3-28 控制信号的逻辑表达式

如图所示，通过分析简化指令系统中所有的指令和控制信号，并集成这些控制信号，形成完整的控制逻辑。

6.2.2 实现代码和仿真代码

实现代码：

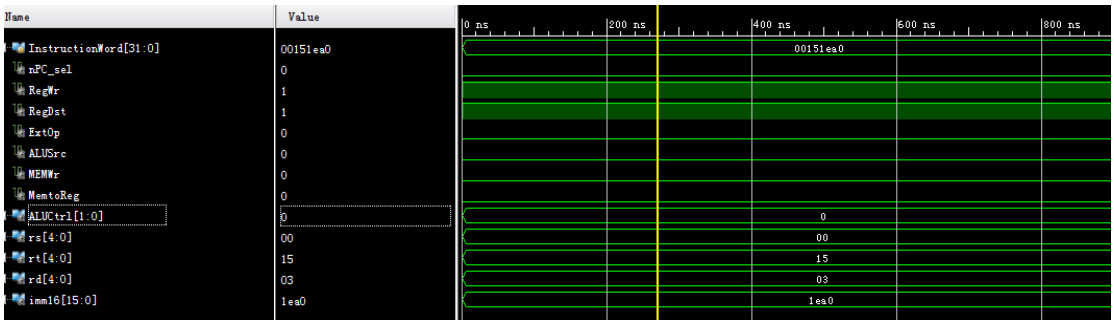
```
module Ctrl(
    input [31:0]InstructionWord,
    output nPC_sel,RegWr,RegDst,ExtOp,ALUSrc,MEMWr,MemtoReg,
    output [1:0]ALUCtrl,
    output [4:0]rs,rt,rd,
```

```
    output [15:0]imm16
    );
parameter ADD = 00;
parameter SUB = 01;
parameter OR = 10;
parameter ZERO = 0;//½øÐÐÁãÀ©Õ¹
parameter SIGN = 1;//½øÐÐÎ»À©Õ¹

wire [5:0]op;
wire [5:0]func;

assign op = InstructionWord[31:26];
assign func = InstructionWord[5:0];
assign rs = InstructionWord[25:21];
assign rt = InstructionWord[20:16];
assign rd = InstructionWord[15:11];
assign imm16 = InstructionWord[15:0];

wire rtype;
assign rtype = ~op[5]&~op[4]&~op[3]&~op[2]&~op[1]&~op[0];

assign add = rtype&func[5]&~func[4]&~func[3]&~func[2]&~func[1]&~func[0];
assign sub = rtype&func[5]&~func[4]&~func[3]&~func[2]&func[1]&~func[0];
assign ori = ~op[5]&~op[4]&op[3]&op[2]&~op[1]&op[0];
assign lw = op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];
assign sw = op[5]&~op[4]&op[3]&~op[2]&op[1]&op[0];
assign beq = ~op[5]&~op[4]&~op[3]&op[2]&~op[1]&~op[0];

assign RegDst = add||sub;
assign ALUSrc = ori||lw||sw;
assign MemtoReg = lw;
assign RegWr = !(sw||beq);
assign MEMWr = sw;
assign nPC_sel = beq;
assign ExtOp = lw||sw;
assign ALUCtrl= (add||lw||sw)?00:((sub||beq)?01:10);

endmodule
```

仿真代码：

```
module Ctrl_tb;
reg [31:0]InstructionWord;
wire nPC_sel,RegWr,RegDst,ExtOp,ALUSrc,MEMWr,MemtoReg;
wire [1:0]ALUCtrl;
wire [4:0]rs,rt,rd;
wire [15:0]imm16;
Ctrl u0(InstructionWord,nPC_sel,RegWr,RegDst,ExtOp,ALUSrc,MEMWr,MemtoReg,ALUCtrl,rs,rt,rd,imm16);
initial begin
InstructionWord=32'b0000000000000000_10101_00011_11010_100000;
end
endmodule
```

6.2.3 仿真波形



6.3 结果分析

由图所示，输入一个32位指令，可得到16位立即数，3个5位地址码，1个2位ALU控制码，以及其他各部件（nPC_sel,RegWr,RegDst,ExtOp,ALUSrc,MEMWr,MemtoReg,

）的控制码。处理正确，满足题目要求。

7 实验7 单周期MIPS处理器设计（不要求，可以放在小学期）

7.1实验内容

用Vivado软件和VerilogHDL或VHDL语言设计实现含有至少6条Mips指令的CPU，要求：

支持无符号加法和减法各一条： addu rd, rs; subu rd, rs, rt；

支持一个操作数为立即数的逻辑运算指令：ori rt, rs, imm16；

支持访问存储器的指令： lw rt, imm16(rs) ； sw rt, imm16(rs)；

支持分支指令： beq rs, rt, imm16；

并仿真验证指令的正确性。

7.2主要步骤

7.2.1 分析实现的方法

这个实验中设计的16位单周期MIPS处理器，是一个RISC（精简指令集）设计，指令设计如下表所示

Name	Fields					Comments
Field size	3 bits	3 bits	3 bits	3 bits	4 bits	All MIPS-L instructions 16 bits
R-format	op	rs	rt	rd	funct	Arithmetic instruction format
I-format	op	rs	rt	Address/immediate		Transfer, branch, immediate format
J-format	op	target address				Jump instruction format

指令设计

Name	Format	Example					Comments
		3 bits	3 bits	3 bits	3 bits	4 bits	
add	R	0	2	3	1	0	add $1,$2,$3
sub	R	0	2	3	1	1	sub $1,$2,$3
and	R	0	2	3	1	2	and $1,$2,$3
or	R	0	2	3	1	3	or $1,$2,$3
slt	R	0	2	3	1	4	slt $1,$2,$3
jr	R	0	7	0	0	8	jr $7
lw	I	4	2	1	7		lw $1, 7 ($2)
sw	I	5	2	1	7		sw $1, 7 ($2)
beq	I	6	1	2	7		beq $1,$2, 7
addi	I	7	2	1	7		addi $1,$2, 7
j	J	2	500				j 1000
jal	J	3	500				jal 1000
slti	I	1	2	1	7		slti $1,$2, 7

1. **Add : R[rd] = R[rs] + R[rt]**
2. **Subtract : R[rd] = R[rs] - R[rt]**
3. **And: R[rd] = R[rs] & R[rt]**
4. **Or : R[rd] = R[rs] | R[rt]**
5. **SLT: R[rd] = 1 if R[rs] < R[rt] else 0**
6. **Jr: PC=R[rs]**
7. **Lw: R[rt] = M[R[rs]+SignExtImm]**
8. **Sw : M[R[rs]+SignExtImm] = R[rt]**
9. **Beq : if(R[rs]==R[rt]) PC=PC+1+BranchAddr**
10. **Addi: R[rt] = R[rs] + SignExtImm**
11. **J : PC=JumpAddr**
12. **Jal : R[7]=PC+2;PC=JumpAddr**
13. **SLTI: R[rt] = 1 if R[rs] < imm else 0**

SignExtImm = { 9{immediate[6]}, imm

JumpAddr = { (PC+1)[15:13], address}

BranchAddr = { 7{immediate[6]}, immediate, 1'b0 }

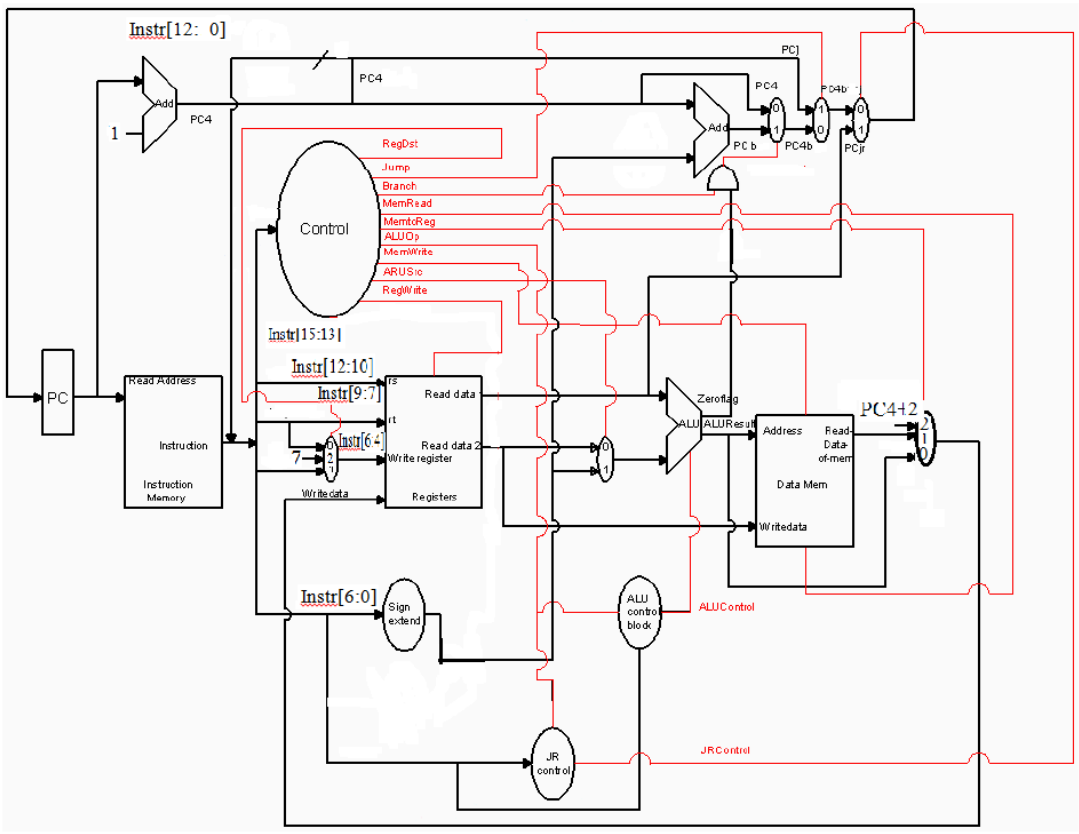
指令详细构成与示例编码

Control signals									
Instruction	Reg Dst	ALU Src	Memto Reg	Reg Write	MemRead	Mem Write	Branch	ALUOp	Jump
R-type	1	0	0	1	0	0	0	00	0
LW	0	1	1	1	1	0	0	11	0
SW	0	1	0	0	0	1	0	11	0
addi	0	1	0	1	0	0	0	11	0
beq	0	0	0	0	0	0	1	01	0
j	0	0	0	0	0	0	0	00	1
jal	2	0	2	1	0	0	0	00	1
slti	0	1	0	1	0	0	0	10	0

控制器微指令控制信号

ALU Control				
ALU op	Function	ALUcnt	ALU Operation	Instruction
11	xxxx	000	ADD	Addi,lw,sw
01	xxxx	001	SUB	BEQ
00	00	000	ADD	R-type: ADD
00	01	001	SUB	R-type: sub
00	02	010	AND	R-type: AND
00	03	011	OR	R-type: OR
00	04	100	slt	R-type: slt
10	xxxxxx	100	slt	i-type: slti

运算器操作指令



MIPS处理器中的数据通路与控制单元示意图

7.2.2 实现代码和仿真代码顶层模块

```
1 `timescale 1ns / 1ps
2 module mips_16(
3               input clk,reset,
4    output[15:0] pc_out,alu_result,reg3,reg4
5    );
6 reg  [15:0] pc_current;
7 wire signed[15:0] pc_next,pc2;
8 wire [15:0] instr;
9 wire [1:0] reg_dst,mem_to_reg,alu_op;
10 wire jump,branch,mem_read,mem_write,alu_src,reg_write     ;
11 wire [2:0]  reg_write_dest;
12 wire [15:0] reg_write_data;
13 wire [2:0]  reg_read_addr_1;
14 wire [15:0] reg_read_data_1;
15 wire [2:0]  reg_read_addr_2;
16 wire [15:0] reg_read_data_2;
17 wire [15:0] sign_ext_im,read_data2,zero_ext_im,imm_ext;
18 wire JRControl;
19 wire [2:0] ALU_Control;
20 wire [15:0] ALU_out;
```

```
21 wire zero_flag;
22 wire signed[15:0] im_shift_1, PC_j, PC_beq, PC_4beq,PC_4beqj,PC_jr;
23 wire beq_control;
24 wire [14:0] jump_shift_1;
25 wire [15:0]mem_read_data;
26 wire [15:0] no_sign_ext;
27 wire sign_or_zero;
28 always @(posedge clk or posedge reset)
29 begin
30     if(reset)
31         pc_current <= 16'd0;
32     else
33         pc_current <= pc_next;
34 end
35 assign pc2 = pc_current + 16'd2;
36 instr_mem instrucion_memory(.pc(pc_current),.instruction(instr));
37 assign jump_shift_1 = {instr[13:0],1'b0};
38 control control_unit(.reset(reset),.opcode(instr[15:13]),.reg_dst(reg_dst)
39             ,.mem_to_reg(mem_to_reg),.alu_op(alu_op),.jump(jump),.branch(branch),.mem_read(mem_read),
40             .mem_write(mem_write),.alu_src(alu_src),.reg_write(reg_write),.sign_or_zero(sign_or_zero));
41 assign reg_write_dest = (reg_dst==2'b10) ? 3'b111: ((reg_dst==2'b01) ? instr[6:4] :instr[9:7]);
42 assign reg_read_addr_1 = instr[12:10];
43 assign reg_read_addr_2 = instr[9:7];
44 register_file reg_file(.clk(clk),.rst(reset),.reg_write_en(reg_write),
45 .reg_write_dest(reg_write_dest),
46 .reg_write_data(reg_write_data),
47 .reg_read_addr_1(reg_read_addr_1),
48 .reg_read_data_1(reg_read_data_1),
49 .reg_read_addr_2(reg_read_addr_2),
50 .reg_read_data_2(reg_read_data_2),
51 .reg3(reg3),
52 .reg4(reg4));
53 assign sign_ext_im = {{9{instr[6]}},instr[6:0]};
54 assign zero_ext_im = {{9{1'b0}},instr[6:0]};
55 assign imm_ext = (sign_or_zero==1'b1) ? sign_ext_im : zero_ext_im;
56 JR_Control JRControl_unit(.alu_op(alu_op),.funct(instr[3:0]),.JRControl(JRControl));
57 ALUControl ALU_Control_unit(.ALUOp(alu_op),.Function(instr[3:0]),.ALU_Control(ALU_Control));
58 assign read_data2 = (alu_src==1'b1) ? imm_ext : reg_read_data_2;
59 alu
alu_unit(.a(reg_read_data_1),.b(read_data2),.alu_control(ALU_Control),.result(ALU_out),.zero(zero_flag));
60 assign im_shift_1 = {imm_ext[14:0],1'b0};
61 assign no_sign_ext = ~(im_shift_1) + 1'b1;
62 assign PC_beq = (im_shift_1[15] == 1'b1) ? (pc2 - no_sign_ext): (pc2 +im_shift_1);
63 assign beq_control = branch & zero_flag;
64 assign PC_4beq = (beq_control==1'b1) ? PC_beq : pc2;
65 assign PC_j = {pc2[15],jump_shift_1};
66 assign PC_4beqj = (jump == 1'b1) ? PC_j : PC_4beq;
67 assign PC_jr = reg_read_data_1;
68 assign pc_next = (JRControl==1'b1) ? PC_jr : PC_4beqj;
69 data_memory datamem(.clk(clk),.mem_access_addr(ALU_out),
70 .mem_write_data(reg_read_data_2),.mem_write_en(mem_write),.mem_read(mem_read),
71 .mem_read_data(mem_read_data));
72 assign reg_write_data = (mem_to_reg == 2'b10) ? pc2:((mem_to_reg == 2'b01)? mem_read_data: ALU_out);
73 assign pc_out = pc_current;
74 assign alu_result = ALU_out;
75 endmodule
```

控制器模块

```
`timescale 1ns / 1ps
module control(
    input[2:0] opcode,
    input reset,
    output reg[1:0] reg_dst,mem_to_reg,alu_op,
    output reg jump,branch,mem_read,mem_write,alu_src,reg_write,sign_or_zero
    );
always @(*)
begin
    if(reset == 1'b1) begin
            reg_dst = 2'b00;
            mem_to_reg = 2'b00;
            alu_op = 2'b00;
            jump = 1'b0;
            branch = 1'b0;
            mem_read = 1'b0;
            mem_write = 1'b0;
            alu_src = 1'b0;
            reg_write = 1'b0;
            sign_or_zero = 1'b1;
    end
    else begin
    case(opcode)
    3'b000: begin // add
            reg_dst = 2'b01;
            mem_to_reg = 2'b00;
            alu_op = 2'b00;
            jump = 1'b0;
            branch = 1'b0;
            mem_read = 1'b0;
            mem_write = 1'b0;
            alu_src = 1'b0;
            reg_write = 1'b1;
            sign_or_zero = 1'b1;
            end
    3'b001: begin // sli
            reg_dst = 2'b00;
            mem_to_reg = 2'b00;
            alu_op = 2'b10;
            jump = 1'b0;
            branch = 1'b0;
            mem_read = 1'b0;
            mem_write = 1'b0;
            alu_src = 1'b1;
            reg_write = 1'b1;
            sign_or_zero = 1'b0;
            end
    3'b010: begin // j
            reg_dst = 2'b00;
            mem_to_reg = 2'b00;
            alu_op = 2'b00;
            jump = 1'b1;
            branch = 1'b0;
```

```
54              mem_read = 1'b0;
55              mem_write = 1'b0;
56              alu_src = 1'b0;
57              reg_write = 1'b0;
58              sign_or_zero = 1'b1;
59              end
60      3'b011: begin // jal
61              reg_dst = 2'b10;
62              mem_to_reg = 2'b10;
63              alu_op = 2'b00;
64              jump = 1'b1;
65              branch = 1'b0;
66              mem_read = 1'b0;
67              mem_write = 1'b0;
68              alu_src = 1'b0;
69              reg_write = 1'b1;
70              sign_or_zero = 1'b1;
71              end
72      3'b100: begin // lw
73              reg_dst = 2'b00;
74              mem_to_reg = 2'b01;
75              alu_op = 2'b11;
76              jump = 1'b0;
77              branch = 1'b0;
78              mem_read = 1'b1;
79              mem_write = 1'b0;
80              alu_src = 1'b1;
81              reg_write = 1'b1;
82              sign_or_zero = 1'b1;
83              end
84      3'b101: begin // sw
85              reg_dst = 2'b00;
86              mem_to_reg = 2'b00;
87              alu_op = 2'b11;
88              jump = 1'b0;
89              branch = 1'b0;
90              mem_read = 1'b0;
91              mem_write = 1'b1;
92              alu_src = 1'b1;
93              reg_write = 1'b0;
94              sign_or_zero = 1'b1;
95              end
96      3'b110: begin // beq
97              reg_dst = 2'b00;
98              mem_to_reg = 2'b00;
99              alu_op = 2'b01;
100             jump = 1'b0;
101             branch = 1'b1;
102             mem_read = 1'b0;
103             mem_write = 1'b0;
104             alu_src = 1'b0;
105             reg_write = 1'b0;
106             sign_or_zero = 1'b1;
107             end
108     3'b111: begin // addi
109             reg_dst = 2'b00;
```

```
            mem_to_reg = 2'b00;
            alu_op = 2'b11;
            jump = 1'b0;
            branch = 1'b1;
            mem_read = 1'b0;
            mem_write = 1'b0;
            alu_src = 1'b1;
            reg_write = 1'b1;
            sign_or_zero = 1'b1;
            end
    default: begin
            reg_dst = 2'b01;
            mem_to_reg = 2'b00;
            alu_op = 2'b00;
            jump = 1'b0;
            branch = 1'b0;
            mem_read = 1'b0;
            mem_write = 1'b0;
            alu_src = 1'b0;
            reg_write = 1'b1;
            sign_or_zero = 1'b1;
            end
    endcase
    end
end
endmodule
```

寄存器模块

```
`timescale 1ns / 1ps
module register_file
(
    input               clk,
    input               rst,
    input               reg_write_en,
    input       [2:0]     reg_write_dest,
    input       [15:0]    reg_write_data,
    input       [2:0]     reg_read_addr_1,
    output      [15:0]    reg_read_data_1,
    input       [2:0]     reg_read_addr_2,
    output      [15:0]    reg_read_data_2
);
    reg    [15:0]    reg_array [7:0];
            always @ (posedge clk or posedge rst) begin
        if(rst) begin
            reg_array[0] <= 15'b0;
            reg_array[1] <= 15'b0;
            reg_array[2] <= 15'b0;
            reg_array[3] <= 15'b0;
            reg_array[4] <= 15'b0;
            reg_array[5] <= 15'b0;
            reg_array[6] <= 15'b0;
            reg_array[7] <= 15'b0;
        end
        else begin
            if(reg_write_en) begin
```

```
28                  reg_array[reg_write_dest] <= reg_write_data;
29              end
30          end
31      end
32      assign reg_read_data_1 = ( reg_read_addr_1 == 0)? 15'b0 : reg_array[reg_read_addr_1];
33      assign reg_read_data_2 = ( reg_read_addr_2 == 0)? 15'b0 : reg_array[reg_read_addr_2];
34  endmodule
```

```
1 module JR_Control(
2               input[1:0] alu_op,
3               input [3:0] funct,
4               output JRControl
5 );
6 assign JRControl = ({alu_op,funct}==6'b001000) ? 1'b1 : 1'b0;
7
8 endmodule
```

运算器控制模块

```
1 `timescale 1ns / 1ps
 2  module ALUControl( ALU_Control, ALUOp, Function);
 3  output reg[2:0] ALU_Control;
 4  input [1:0] ALUOp;
 5  input [3:0] Function;
 6  wire [5:0] ALUControlIn;
 7  assign ALUControlIn = {ALUOp,Function};
 8  always @(ALUControlIn)
 9  casex (ALUControlIn)
10   6'b11xxxx: ALU_Control=3'b000;
11   6'b10xxxx: ALU_Control=3'b100;
12   6'b01xxxx: ALU_Control=3'b001;
13   6'b000000: ALU_Control=3'b000;
14   6'b000001: ALU_Control=3'b001;
15   6'b000010: ALU_Control=3'b010;
16   6'b000011: ALU_Control=3'b011;
17   6'b000100: ALU_Control=3'b100;
18   default: ALU_Control=3'b000;
19   endcase
20  endmodule
```

运算器模块

```
1 module alu(
 2      input         [15:0]     a,          //src1
 3      input         [15:0]     b,          //src2
 4      input         [2:0]      alu_control,     //function sel
 5      output     reg     [15:0]     result,          //result
 6      output zero
 7    );
 8  always @(*)
 9  begin
10      case(alu_control)
11      3'b000: result = a + b; // add
12      3'b001: result = a - b; // sub
13      3'b010: result = a & b; // and
14      3'b011: result = a | b; // or
15      3'b100: begin if (a<b) result = 16'd1;
16                  else result = 16'd0;
```

```
                end
    default:result = a + b; // add
    endcase
end
assign zero = (result==16'd0) ? 1'b1: 1'b0;
endmodule
```

```
`timescale 1ns / 1ps
module data_memory
(
    input                   clk,
    input    [15:0]           mem_access_addr,
    input    [15:0]           mem_write_data,
    input                   mem_write_en,
    input                   mem_read,
    output   [15:0]           mem_read_data
);
    integer i;
    reg [15:0] ram [255:0];
    wire [7 : 0] ram_addr = mem_access_addr[9 : 2];
    initial begin
        for(i=0;i<256;i=i+1)
            ram[i] <= 16'd0;
    end
    always @(posedge clk) begin
        if (mem_write_en)
            ram[ram_addr] <= mem_write_data;
    end
    assign mem_read_data = (mem_read==1'b1) ? ram[ram_addr]: 16'd0;
endmodule
```

指令预设模块

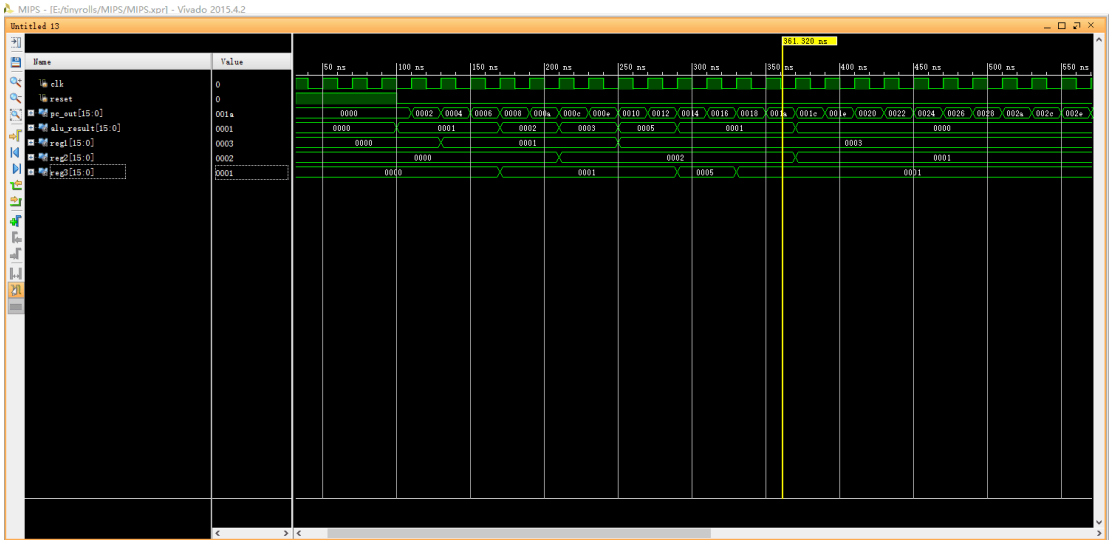
```
`timescale 1ns / 1ps
module instr_mem(
 input          [15:0]    pc,
 output          [15:0]    instruction
);
    wire [3 : 0] rom_addr = pc[5 : 2];
    reg [15:0] rom[15:0];
    initial
    begin
                                                rom[0] = 16'b001_010_001_000_0111;//reg1 = 1;
            rom[1] = 16'b000_010_001_011_0000;//reg3 = reg1 + reg2, reg3=1
            rom[2] = 16'b000_011_001_010_0000;//reg2 = reg1 + reg3, reg2=2
            rom[3] = 16'b000_010_011_001_0000;//reg1 = reg2 + reg3, reg1=3
            rom[4] = 16'b000_001_010_011_0000;//reg3 = reg1 + reg2, reg3=5
            rom[5] = 16'b000_001_010_011_0001;// reg3 = reg1 - reg2, reg3=1
            rom[6] = 16'b000_011_001_010_0010;//reg2 = reg1 & reg3   reg2=3&1=1
            rom[7] = 16'b0000000000000000;
            rom[8] = 16'b0000000000000000;
            rom[9] = 16'b0000000000000000;
            rom[10] = 16'b0000000000000000;
            rom[11] = 16'b0000000000000000;
            rom[12] = 16'b0000000000000000;
            rom[13] = 16'b0000000000000000;
```

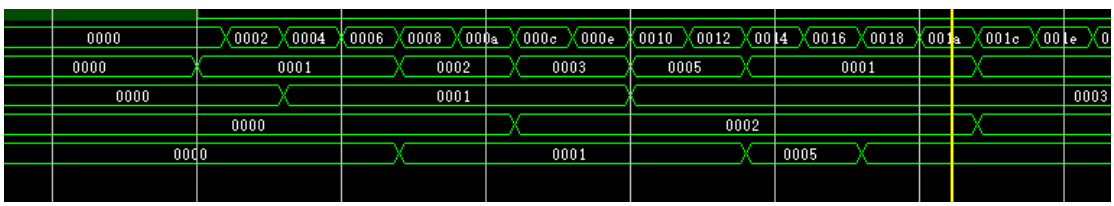
```
24              rom[14] = 16'b0000000000000000;
25              rom[15] = 16'b0000000000000000;
26      end
27      assign instruction = (pc[15:0] < 64 )? rom[rom_addr[3:0]]: 16'd0;
28  endmodule
```

测试激励文件

```
1  `timescale 1ns / 1ps
2  module tb_mips16;
3      reg clk;
4      reg reset;
5          wire [15:0] pc_out;
6      wire [15:0] alu_result,reg3,reg4;
7          mips_16 uut (
8          .clk(clk),
9          .reset(reset),
10         .pc_out(pc_out),
11         .alu_result(alu_result),
12         .reg3(reg3),
13         .reg4(reg4)
14     );
15     initial begin
16         clk = 0;
17         forever #10 clk = ~clk;
18     end
19     initial begin
20         reset = 1;
21                 #100;
22     reset = 0;
23             end
24  endmodule
```

7.2.3 仿真波形





7.3 结果分析

```
rom[0] = 16'b001_010_001_000_0111;//reg1 = 1;
rom[1] = 16'b000_010_001_011_0000;//reg3 = reg1 + reg2, reg3=1
rom[2] = 16'b000_011_001_010_0000;//reg2 = reg1 + reg3, reg2=2
rom[3] = 16'b000_010_011_001_0000;//reg1 = reg2 + reg3, reg1=3
rom[4] = 16'b000_001_010_011_0000;//reg3 = reg1 + reg2, reg3=5
rom[5] = 16'b000_001_010_011_0001;// reg3 = reg1 - reg2, reg3=1
rom[6] = 16'b000_011_001_010_0010;//reg2 = reg1 & reg3   reg2=3 &1 = 1
rom[7] = 16'b0000000000000000;
rom[8] = 16'b0000000000000000;
```

上图是预设的指令及其注释，将波形图中的倒数三行是寄存器监控波形，程序计数器保持+2，运算器结果输出也是每次运算的正确结果，3个寄存器的值的变化也如指令所给的一样正确存取。

四：结论（讨论）

1、结论

1.1 实验1

为理解32位寄存器堆的主要原理，实验一除触发器外全部由门级电路构成。加深了对多路选通器和译码器的理解，以及利用小位数芯片级联扩展成多位数的芯片的应用思路。同时从具体到抽象理解了寄存器堆内部的数据通路。

1.2 实验2

存储器是CPU内的高速缓存，其与寄存器堆的原理大致相同，支持写入和读出32位数据，不同的是容量有2的32次方bit，在我们编写时，难以达到理想的2的32次方bit容量，于是简化了模型。对于存储器的写操作，在时钟上升沿发生，而读操作不受时钟信号的控制。

1.3 实验3

实现了8位直接相联方式的单向读出数据的Cache部件，将主存中的块放到Cache中，在程序访问的局部性条件下，使得主存的平均读出时间接近Cache的读出时间，大大提高了CPU的访存效率。编写Cache的过程，不仅让我们学会了信号传递的时序逻辑和控制逻辑，以及各模块间的组合调用控制，更加深了我们能对程序设计的空间局部性和时间局部性的理解，对计算机的底层硬件优化有了更深层次的认识。

1.4 实验4

实现了32位取指部件，该部件可以支持指令的连续读取以及任意给定目标地址的读取。可

完成了不断取回指令的功能。对IFU内部功能实现有了认识，也对取指过程有了更深层次的理解，将理论与实际结合起来。

1.5 实验5

实现了32位ALU部件，可以支持无符号加法、减法、逻辑或运算，加法部分采用超前进位方法实现。并行加法器之间亦为并行相连。通过实验5，我熟悉了并行加法器进位链表达式，片间并行的表达式，以及其具体连接方式。

1.6 实验6

实现了硬布线控制器部件。主要是为了把控制信号集成起来，让这个处理器自动的工作。分析完简化指令系统中所有的指令和控制信号集成这些控制信号，形成完整的控制逻辑。通过实验6，我了解了分析控制信号，并声称逻辑电路的基本流程。

1.7 实验7

实现了16位单周期MIPS处理器部件，一共预设了13条指令，参考了计组指导书的部件现实代码，结合国外大学的数据通路的实现思路，初步搭建了一个较为简易的处理器框架，整体设计难点在控制信号的安排与传递，微指令的安排设计等， 我们反复分析反复试错，运用自顶向下的设计思想，先把整体框架搭建完成，再去具体实现各个模块，而且前面的实验已经实现了处理器的大部分模块。通过整个实验过程，我们对处理器整个系统有了较为充分的了解，对各模块间的数据通路的衔接也有了明确认识，更重要的是学会了在实现较大工程时的思路和想法的构建以及良好的心态。

2、讨论

可改进：实验一中的多路选通器可以用三态门替代。多路选通器完全由门级搭建所需代码过于繁杂，而使用三态门替代可精简代码。但二者的功能和作用类似，在实际应用上差别不大。

在本实验我们运用EDA技术，采用系统能力培养指定的实践平台（Xilinx Vivado设计工具、 Xilinx EGO1实验平台），自己设计一些基本的计算机组成原理电路单元，进一步加深对组成原理中重要部件原理的理解。

五、教师评审

教师评语	实验成绩
签名: 日期:	