绝密

单时钟CPU设计

自然选择前进四

ZPC 2015
ZHEJIANG UNIVERSITY
浙江大学计算机学院

编号 23333

目录：

单时钟 CPU 实验报告

小组基本信息：

组名：自然选择前进四

项目负责经理：沈旭东

其他组员：钱旭峰、王海容、黄一伦

使用手册：

操控：

可操控的部件为 SW[7:0]，即为开关的右数前八个开关，其中：
SW[4:0]用于查看 32 个寄存器组的指定寄存器号中的数据；
SW[5]用于查看当前 PC 地址，开关开则为显示 PC 地址，关则显示寄存器数据；
SW[6]用于查看当前显示的数据的高 16 位与低 16 位，开关开则为显示数据的高 16 位，关则显示数据的低 16 位；
SW[7]用于单步执行，拨动一次开关则为运行单时钟 CPU 的一个完整周期。当然也设计了 slow1to10 模块来使放慢后的时钟来代替手动拨动为一个完整周期。

显示：

显示的部件为 16 位的 7 段数码管，其中：
实际数据为 32 位，当前显示的数据由 SW[6]控制高低位。

数据导入：

执行代码的导入：
将想要执行的代码变为机器码后整合为后缀为.bin 的二进制文件，然后运行 generate_coe.exe 程序，写入二进制文件的名字（绝对路径），将会得到四个 coe 文件，依次用这四个文件初始化 IP 核 mem1, mem2, mem3, mem4 即可。

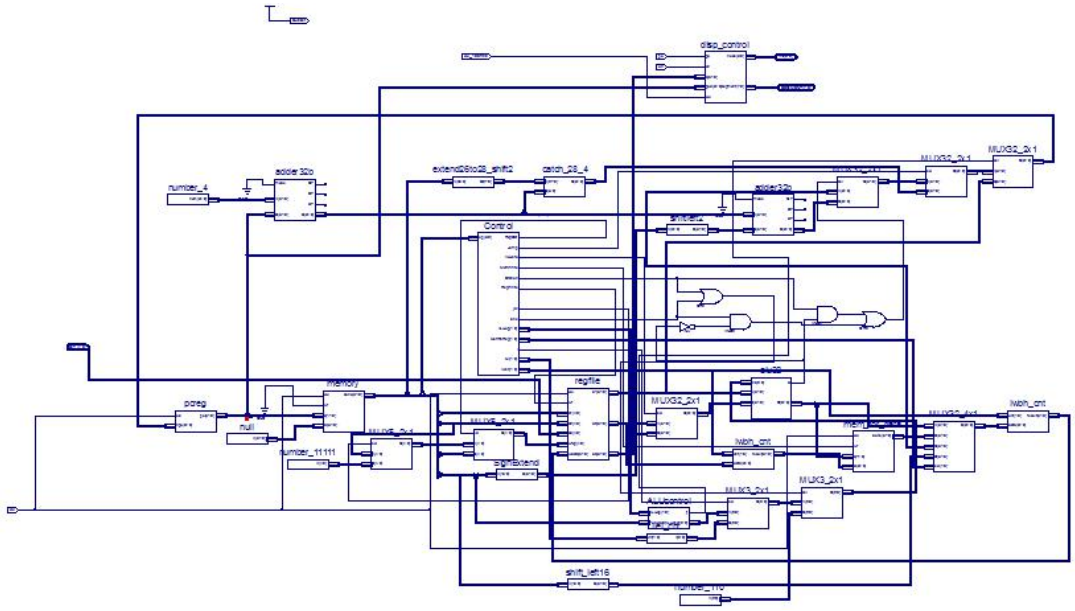
数据存储器数据的导入：
方法同执行代码的导入，初始化四个 IP 核 mem5, mem6, mem7, mem8 就是需要注意数据的导入格式。

单时钟 CPU 基本信息：

语句支持：

支持 and, and, add, sub, or, xor, slt, lw, sw, beq, bne, j, jal, lui, addi, slti, lb, lh, sb, sh, andi, ori, xori, 语句的指令

整体概貌：（具体的在后文介绍）



本单时钟预先内置的测试程序：（为测试所有指令）

```
0x00000000:addi $sp $zero 200      //$sp=200
0x00000004:lui  $t0 2              //$t0=2^17
0x00000008:slti  $t1 $t0 123       //$t1=0
0x0000000c:addi $s0 $t1 1          //
0x00000010:add $t1 $s0 $zero       //$t1=1
0x00000014:addi $t2 $zero 12       //$t2=12
0x00000018:andi $t3 $t2 3          //$t3=0
0x0000001c:ori  $t3 $t2 15         //$t3=15
0x00000020:xori $t3 $t2 15         //$t3=3
0x00000024:sub $t3 $t2 $t1         //$t3=11
0x00000028:and $t1 $t2 $t3         //$t1=1100 & 1011=1000=8
0x0000002c:addi $a0 $t1 1          //$a0=1001=9
```

```
0x00000030:xor $s0 $t1 $a0          //$s0=1
0x00000034:add $t1 $s0 $zero        //$t1=1
0x00000038:add $ra $zero $zero      //$ra=0
0x0000003c:j 18                     //跳转到 18*4 即 0x48 的地址
0x00000040:00000000
0x00000044:00000000
0x00000048:or $s0 $t2 $t1           //$s0=13 address=0x48(为十进制 72)
0x0000004c:lb $t7 0($zero)          //$t7=1
0x00000050:lh $t7 4($zero)          //$t7=2
0x00000054:sb $t7 50($zero)
0x00000058:sh $t7 50($zero)
0x0000005c:add $t2 $s0 $zero        //$t2=13
0x00000060:bne $t1 $t2 2            //指令计数先加一后再加二到 0x6c
0x00000064:00000000
0x00000068:00000000
0x0000006c:addi $s0 $sp -8          //$s0=192 (0xc0)
0x00000070:add $sp $s0 $zero        //$sp=192
0x00000074:sw $ra 4($sp)
0x00000078:sw $a0 0($sp)
0x0000007c:slt $t0 $a0, $t1         //$t0 为 0
0x00000080:beq $t0 $zero 4          //相同时跳转到 0x94 地址
0x00000084:addi $v0 $zero 1         //以上结论只是第一轮适用
0x00000088:addi $s0 $sp 8
0x0000008c:add  $sp $s0 $zero
0x00000090:jr $ra                   //
0x00000094:addi $s0  $a0  -1
0x00000098:add  $a0  $s0 $zero
0x0000009c:jal  27                  //将 0xa0 存入寄存器$ra
0x000000a0:lw $a0 0($sp)
0x000000a4:lw $ra 4($sp)
0x000000a8:addi $s0 $sp, 8
0x000000ac:add  $sp $s0 $zero
0x000000b0:add $s0 $a0 $v0
0x000000b4:add $v0 $s0 $zero        //
0x000000b8:jr $ra
//共 47 行.
```

预先导入的各个 coe 文件（请见附录）

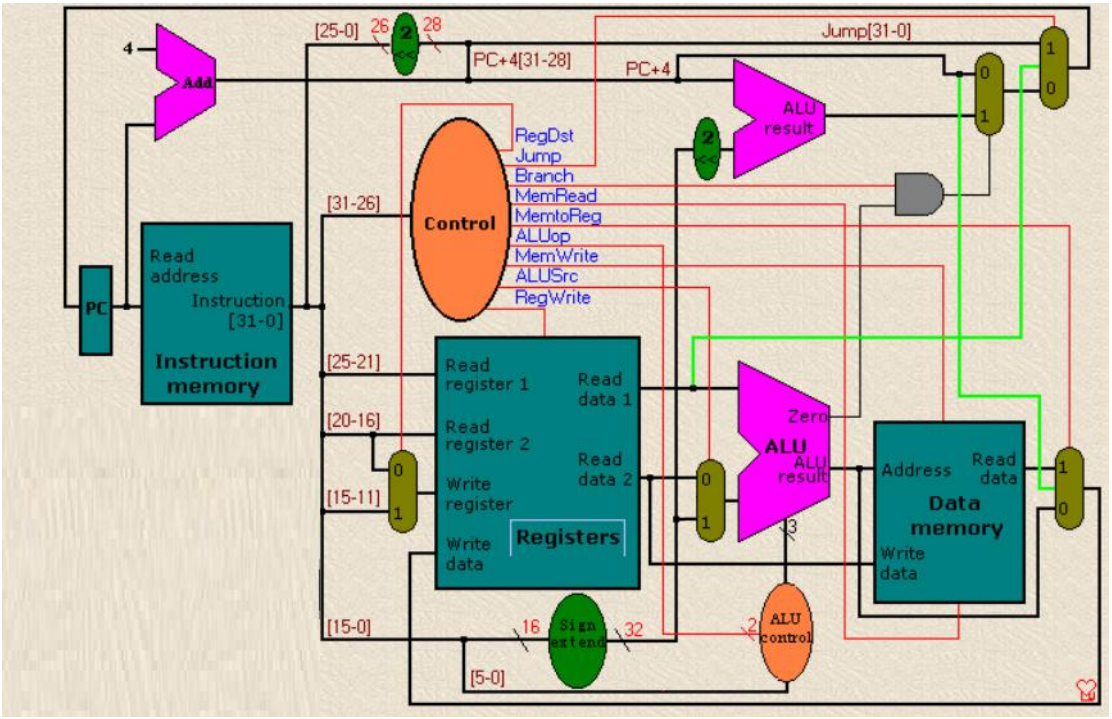
具体的指令转化为二进制文件的步骤请参阅本组的另一个作品，mips 模拟器。同时为本实验提供了理论上软件的模拟。

任务分工：

钱旭峰：提供指令以及数据的二进制文件，即测试案例
王海容：负责 32 寄存器组与 PC 寄存器，指令数据存储器
黄一伦：从运算器中提取了符号加法模块，并包装为 alu32 模块
沈旭东：设计添加指令，控制器的设计，整体线路设计与最终调试改错

语句扩展原理图：（在已有个人 CPU 上补充）

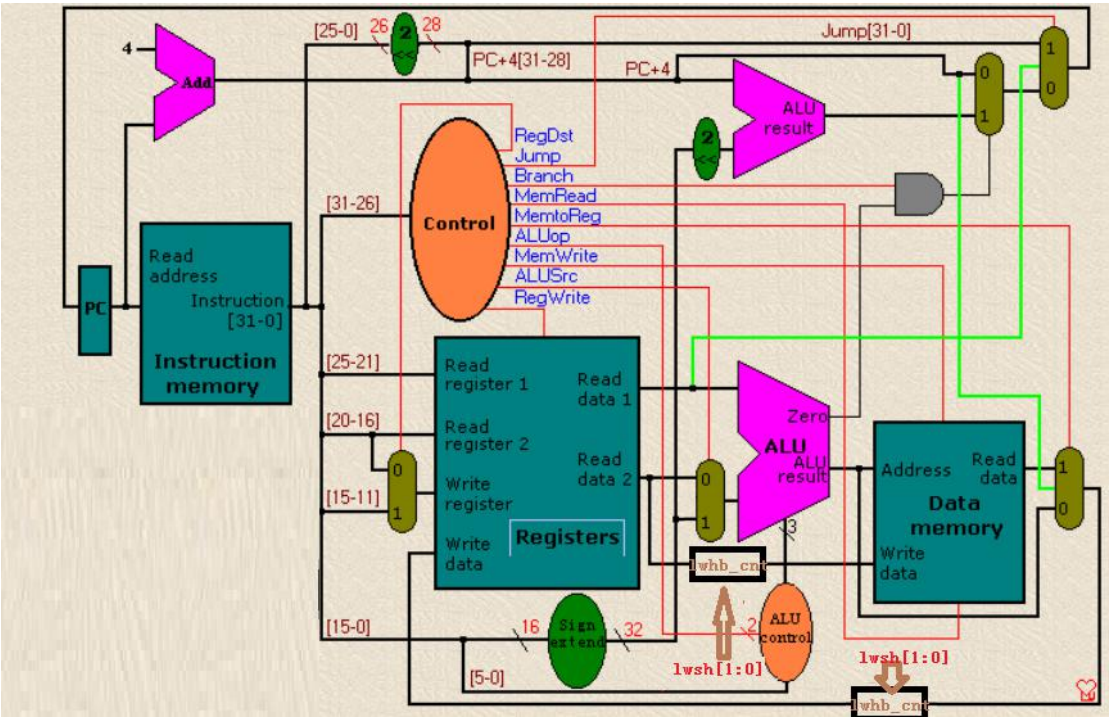
下图为个人单时钟 CPU 的原理图(包括 R-语句，lw，sw，beq，j）



其中控制信号为：

	R-	lw	sw	beq	j
RegDst	1	0	x	x	x
ALUSrc	0	1	1	0	x
MemtoReg	0	1	x	x	0
RegWrite	1	1	0	0	0
MemRead	0	1	0	0	0
MemWrite	0	0	1	0	0
Branch	0	0	0	1	0
ALUop	10	00	00	01	00
Jump	0	0	0	0	1

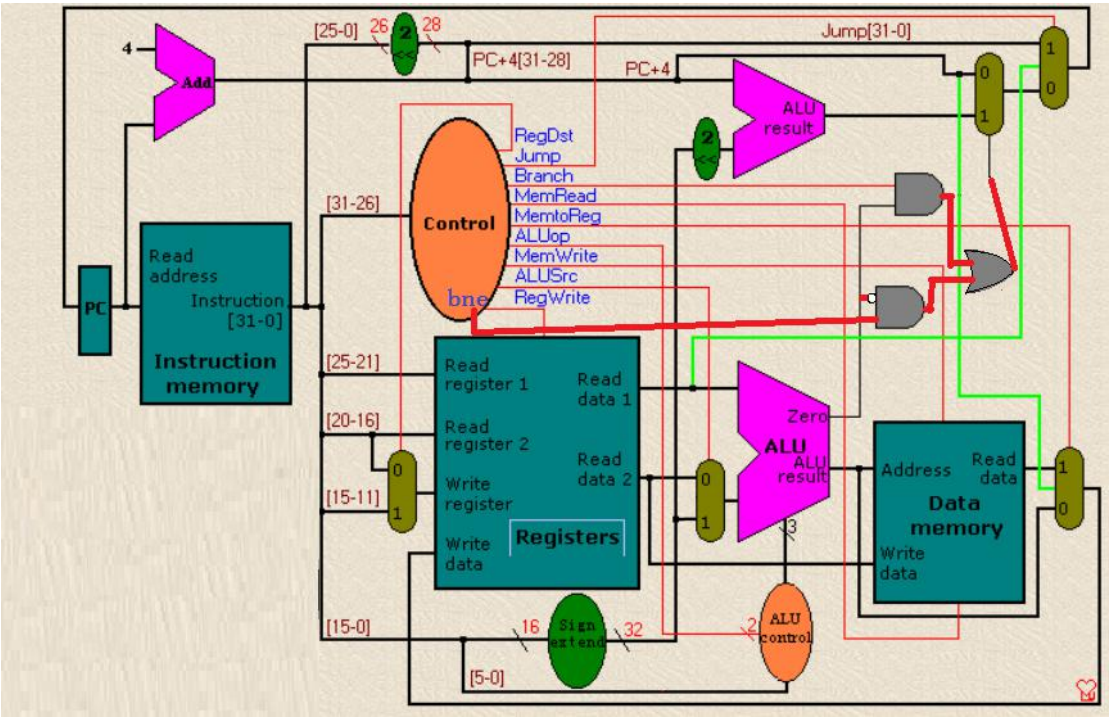
扩展 sh， lh， sb， lb 语句：



增添 lwhb_cnt 模块，功能为处理输入的 32 位数据，根据信号截取 32 位数据为 16 位，8 位，或者不变的数据并且输出。控制信号为：

	lw	sw	lh	sh	lb	sb
RegDst	0	x	0	x	0	x
ALUSrc	1	1	1	1	1	1
MemtoReg	1	x	1	x	1	x
RegWrite	1	0	1	0	1	0
MemRead	1	0	1	0	1	0
MemWrite	0	1	0	1	0	1
Branch	0	0	0	0	0	0
ALUop	00	00	00	00	00	00
Jump	0	0	0	0	0	0
Lwsh(新增)	00	00	10	10	01	01

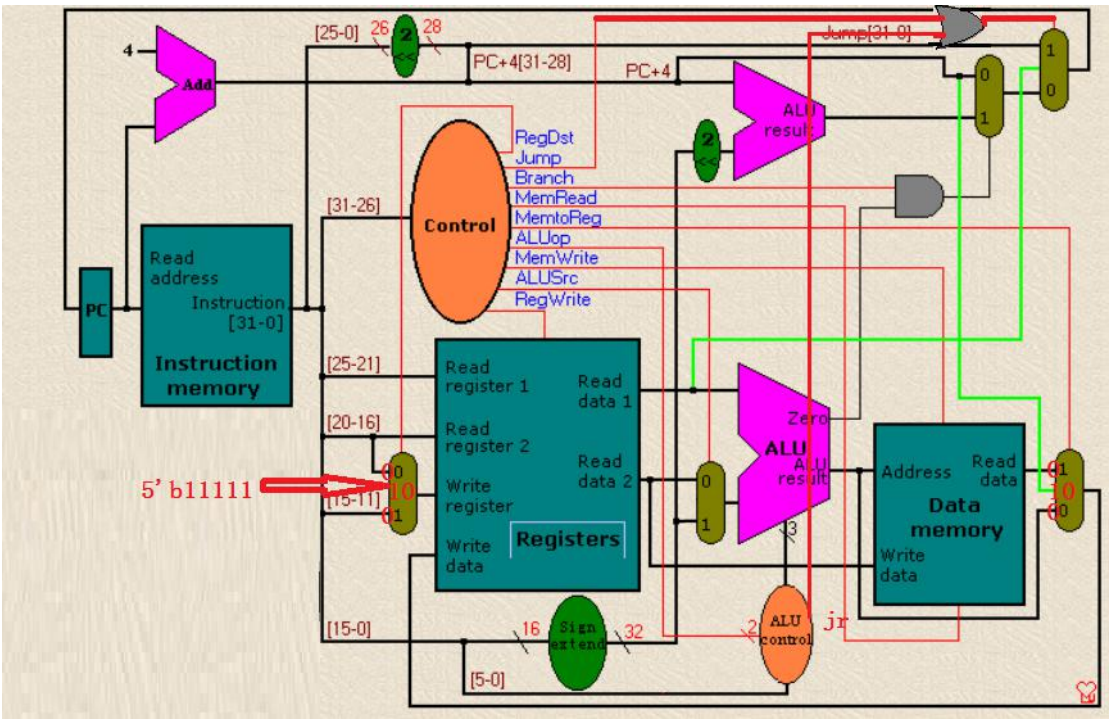
扩充 bne 语句：



增添 control 模块控制信号，控制在 op 为 bne 的时候检测 ALU 的零判断是否为真，如果为假则实行与 beq 相同的地址操作。控制信号为：

	beq	beq
RegDst	x	x
ALUSrc	0	0
MemtoReg	x	x
RegWrite	0	0
MemRead	0	0
MemWrite	0	0
Branch	1	1
ALUop	01	01
Jump	0	0
bne(新增)	0	1

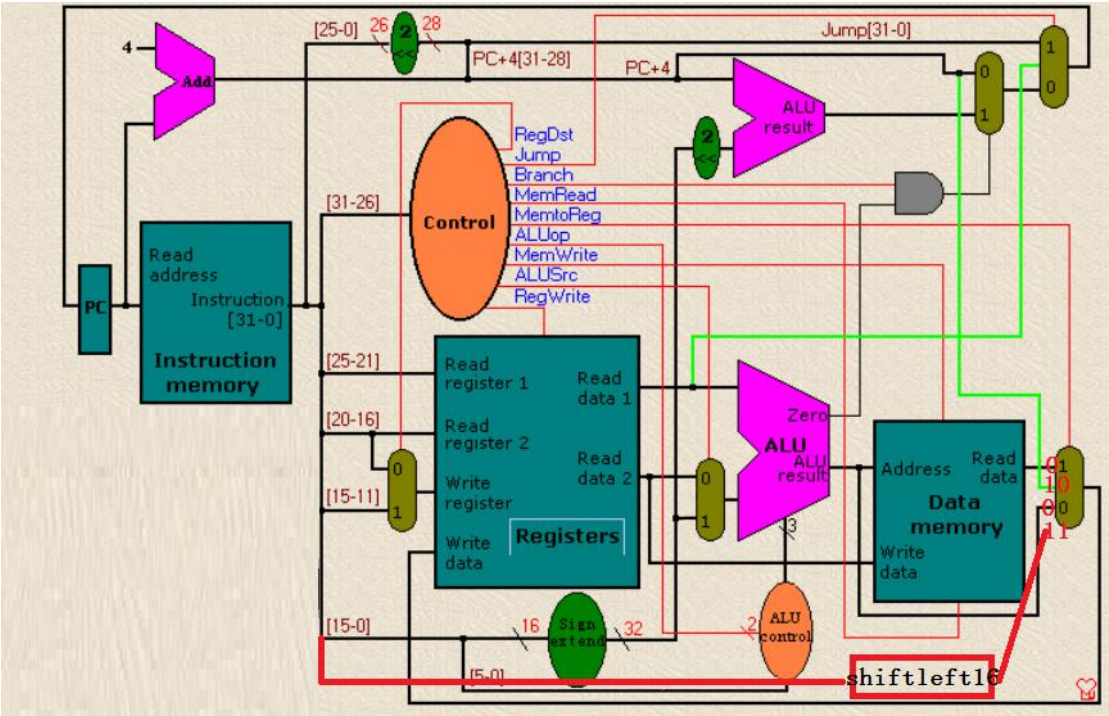
扩充 jal 与 jr 语句：



在原先拥有 j 语句的基础上加以实现，jr 的判别在 ALUcontrol 中实现，因为它也是 R-类语句，所以可以在 R-类语句基础上加以一定的修改。而 jal 语句则是在 j 的基础上添加了将该语句的后一条语句的地址写如$ra 的功能，可以改成如图的方式，下表为对应的控制信号，而实际制作过程中由于两处地方要使用到这个信号所以就不把它与 RegDst 混在一起了。

	j	jal	jr
RegDst	x	10	x
ALUSrc	x	x	x
MemtoReg(改)	00	10	00
RegWrite	0	0	0
MemRead	0	0	0
MemWrite	0	0	0
Branch	0	0	0
ALUop	00	00	00
Jump	1	1	1
jr(新增)	0	0	1

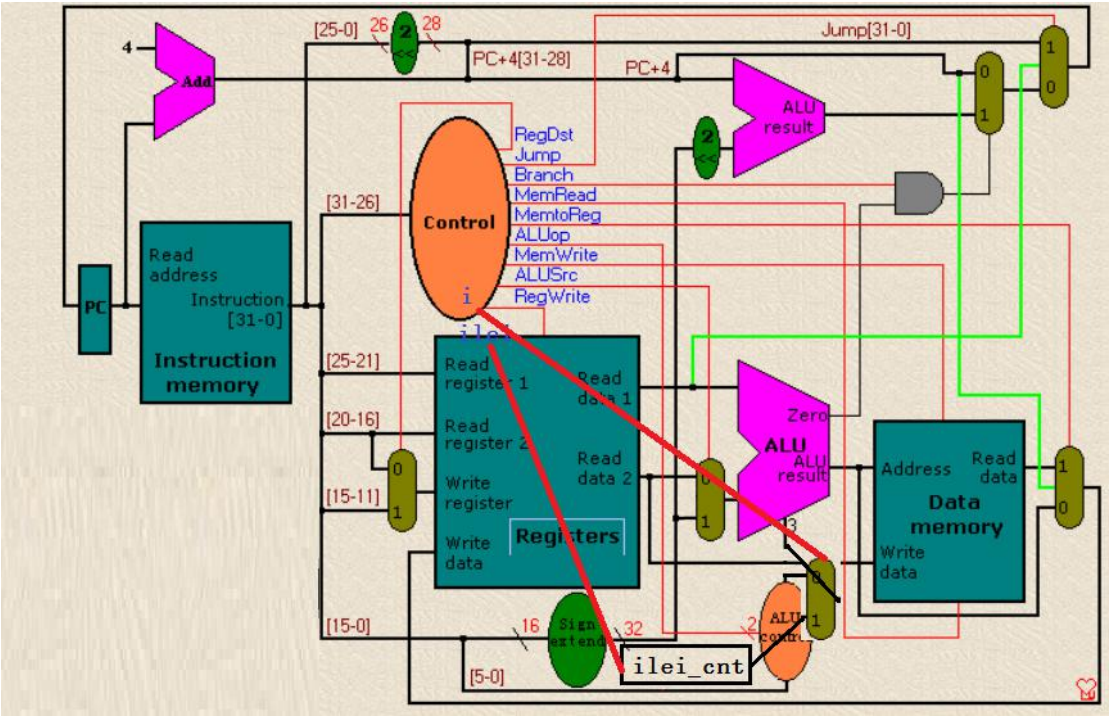
扩充 lui 语句：



新添 shiftleft16 模块，将一个 16 位立即数左移 16 位后输出一个 32 位数据，并载入到目标寄存器当中去，其控制信号是：

	lui
RegDst	0
ALUSrc	1
MemtoReg(改)	11
RegWrite	1
MemRead	1
MemWrite	0
Branch	0
ALUop	00
Jump	0
Lwsh(新增)	00

扩充 i 类其他语句（addi，ori，xori，andi，slti）：



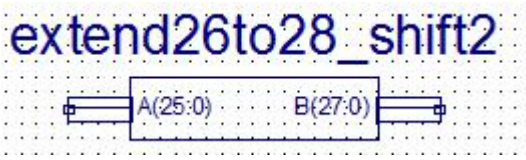
新添一个 ilei_cnt 的模块来处理 ori, xori, andi, slti 所需要的 ALU 控制指令，由于 addi 指令与 add 指令可以都通过原有的 ALUcontrol 传递，所以无需加入此处的控制，新添一个控制信号 i 用于传输立即数的指令信号，新添控制信号 ilei，用于判别此为哪一条 i 类信号从而控制 ilei_cnt 生成正确 ALU 操作信号。控制信号如下：

	addi	slti	andi	ori	xori
RegDst	1	1	1	1	1
ALUSrc(改)	1	1	1	1	1
MemtoReg	0	0	0	0	0
RegWrite	1	1	1	1	1
MemRead	0	0	0	0	0
MemWrite	0	0	0	0	0
Branch	0	0	0	0	0
ALUop	10	10	10	10	10
Jump	0	0	0	0	0
I(新增)	0	1	1	1	1
Ilei(新增)	x	00	01	10	11

各个模块介绍：（各个模块的代码将附在附录中）

extend26to28_shift2：

模块功能：将 26 位数据左移 2 位，输出 28 位数据
用途：用于 J 语句的地址处理
图样：



shiftleft2：

模块功能：将 32 位数据左移 2 位，输出 32 位数据
用途：用于 beq 语句的地址处理
图样：

SignExtend：

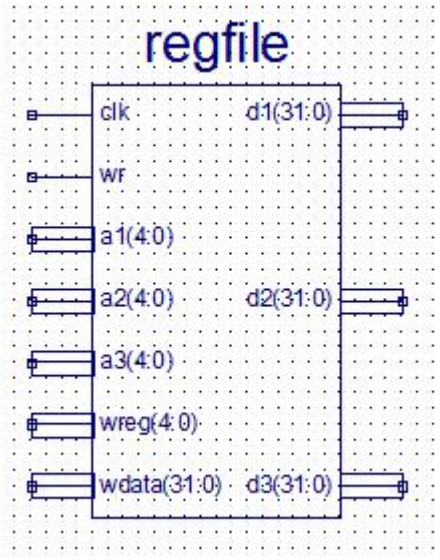
模块功能：将 16 位有符号数据扩展为 32 位有符号数据并输出
用途：用于立即数（i 类）的处理
图样：

Regfile：

模块功能：即 32 个 32 位寄存器组，a1,a2 输入读取数据的寄存器号，对应 d1,d2 输出该寄存器的数据，wr 信号为写入信号，wreg 输入写入数据的寄存器号，wdata 则是写入的数据，而 a3 则是留下的后门，可输入要查看的寄存器号，从 d3 输出

用途：CPU 处理寄存器组

图样：

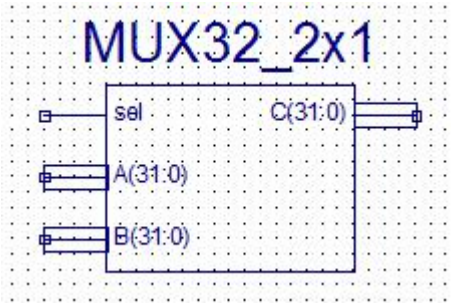


MUX32_2x1：

模块功能：根据 sel（选择）信号来选择输出数据，0 则输出 A，1 则为 B

用途：选择数据，同时也可以被四路选择调用

图样：

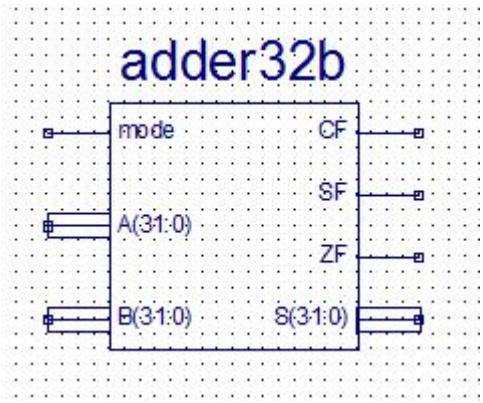


adder32b：

模块功能：根据 mode 信号来对输入数据（32 位）进行加减法并且输出结果，以及对结果的判断（后面主要用到的是 0 值判断）

用途：PC 地址加 4，还有 jal 语句对地址的处理，同时也可以被 alu32 模块调用

图样：

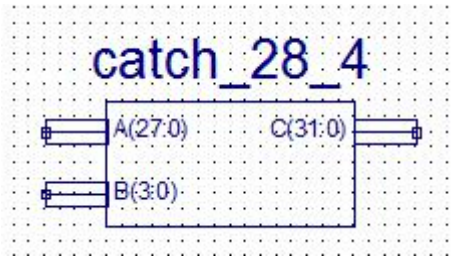


catch_28_4：

模块功能：连接 28 位（作为低位）和 4 位（作为高位）数据，输出连接后的 32 位数据

用途：用于 J 语句的地址处理

图样：

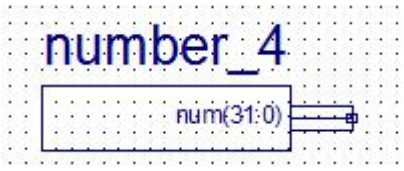


number_4:

模块功能：提供数据 4

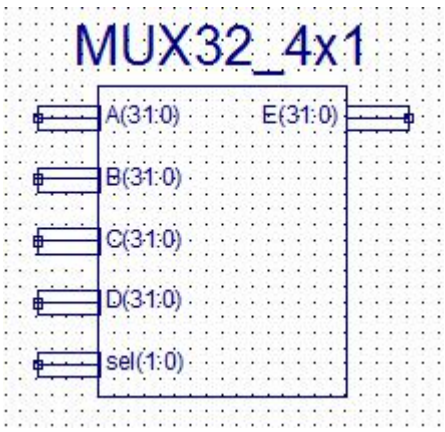
用途：为地址加 4 提供数据

图样：



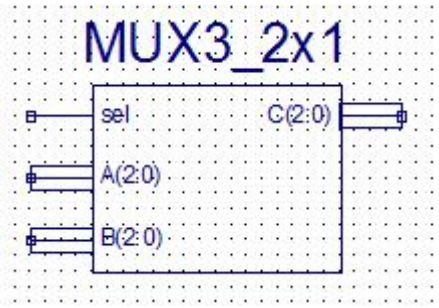
MUX32_4x1：

模块功能：根据 sel（选择）信号来选择输出的数据，00 则输出 A，01 则输出 B，10 则输出 C，11 则输出 D
用途：选择数据
图样：



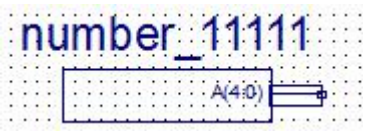
MUX3_2x1：

模块功能：根据 sel（选择）信号来选择输出的数据，0 则输出 A，1 则为 B
用途：在 i 类和 r 类语句调用 alu32 模块的时候有不同选择
图样：



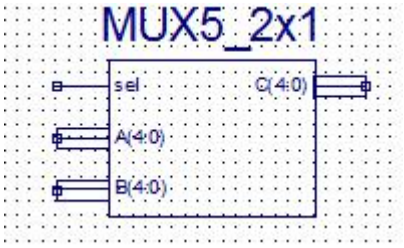
number_11111：

模块功能：提供数据 5’b11111
用途：用于 jal 语句时存入下一条指令地址选择$ra 时提供寄存器号
图样：



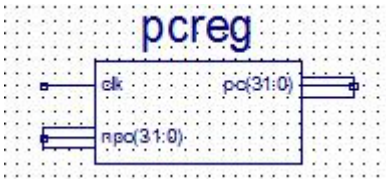
MUX5_2x1：

模块功能：根据 sel（选择）信号来选择输出的数据，0 则输出 A，1 则为 B
用途：在确定 r 类和 i 类第二个寄存器为写入还是读出信号时用
图样：



pcreg:

模块功能：存放当前 PC 值，并且在下一个大周期上升沿读取下一个地址
用途：PC 地址寄存器
图样：

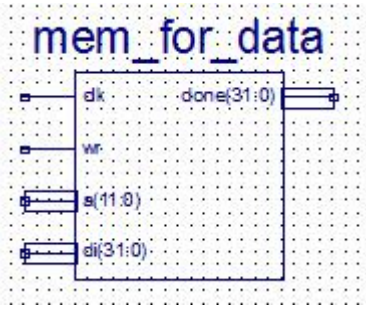


shift_left16：

模块功能：将 16 位数据左移 16 位，输出 32 位数据
用途：用于 lui 语句的读入数据的处理
图样：

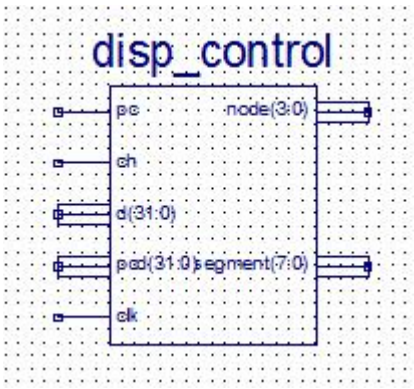
mem_for_data：

模块功能：调用四个 ip 核来完成数据的存储
用途：存放数据
图样：



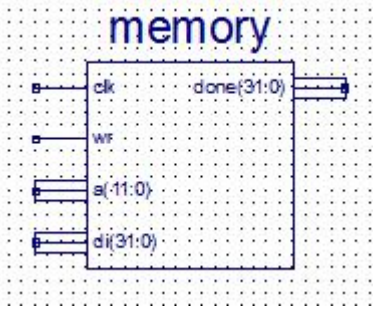
disp_control：

模块功能：显示数据，当 ch 为 1 时，显示高 16 位数据，否则为低 16 位，当 pc 为 1 时显示 pc 地址，否则为从 regfile 的 d3 端口读出的数据
用途：显示模块
图样：



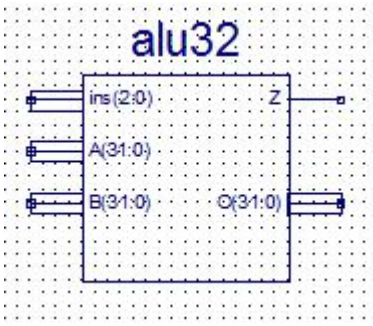
memory：

模块功能：调用四个 ip 核来存放各个指令，根据 pcreg 的地址来读取指令并输出
用途：存放指令
图样：



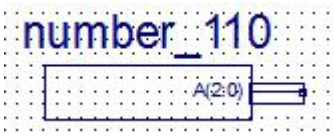
alu32：

模块功能：根据 ALUcontrol 的信号来调用 adder32b 模块，并提供输出结果
用途：处理各种数据
图样：



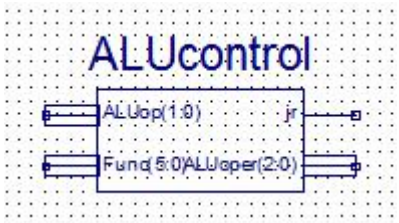
number_110：

模块功能：提供数据 2’b110
用途：在 beq 和 bne 语句中来额外操控 alu32，判断是否为 0
图样：



ALUcontrol：

模块功能：根据 Control 模块提供的 ALUop，以及指令的最后六位来向 alu32 模块提供操控信号，同时也是判断 jr 语句的标志
用途：控制 alu32 模块
图样：

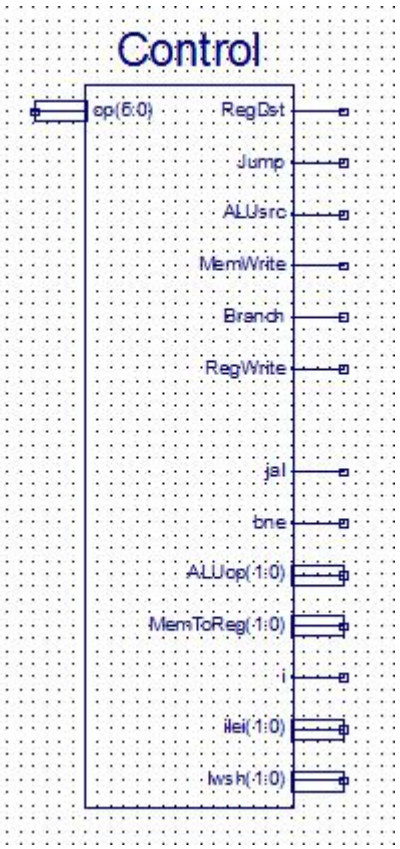


Control：

模块功能：根据指令的前缀码来完成各种类型指令的判断以及对各个信号的控制。RegDst 信号控制写入寄存器号的来源；Jump 信号控制 J 语句的地址信号来源；ALUsrc 信号控制放入运算器的数据来源；MemWrite 信号控制数据存放的写入；Branch 信号控制 Beq 语句的地址选择；RegWrite 信号控制 32 寄存器组的数据写入；jal 信号控制 jal 语句的相关操作；bne 信号控制 bne 语句的相关处理；ALUop 信号控制运算器是为 r 类服务还是为地址跳转服务；MemToReg 信号则是选择写入数据的数据来源；i 信号则是选择 i 类语句特有的 ALU 控制信号源；ilei 信号则是为 i 类语句提供特有的 ALU 控制的信号；lwsh 信号则是为了 lw，lh，lb，sw，sh，sb 语句提供相对应的数据的截取。

用途：控制各处的信号

图样：



ilei_cnt：

模块功能：为 i 类的语句提供相应的 alu32 控制信号

用途：用于 i 类语句的控制

图样：

lwbh_cnt：

模块功能：通过对指令的判断来决定存入/读出的数据的类型（字节，半字，字）

用途：用于 lw, sw, lh, sh, lb, sb 语句的数据处理

图样：

附录：

各模块的代码：

extend26to28_shift2：

```
module extend26to28_shift2(input wire [25:0]A,
                  output wire [27:0]B
    );
    assign B[27:2]=A[25:0];
    assign B[1:0]=0;
endmodule
```

shiftleft2：

```
module shiftleft2(input wire [31:0]A,
                  output wire [31:0]B
    );
assign B[31:2]=A[29:0];
assign B[1:0]=2'b0;

endmodule
```

SignExtend：

```
module SignExtend(input wire [15:0]A,
                     output wire [31:0]B
    );
assign B[15:0]=A[15:0];
assign
```

```
B[31:16]={A[15],A[15],A[15],A[15],A[15],A[15],A[15],A[15],A[15],A[15]
,A[15],A[15],A[15],A[15],A[15],A[15]};

endmodule
```

regfile:

```
module regfile(clk,a1,a2,a3,d1,d2,d3,wr,wreg,wdata);
    input clk;
    input [4:0]a1;
    input [4:0]a2;
    input [4:0]a3;
    output [31:0]d1;
    output [31:0]d2;
    output [31:0]d3;
    input wr;
    input [4:0]wreg;
    input [31:0]wdata;
    reg [31:0]_d1,_d2,_d3;
    reg [31:0]m[0:31];
    always @(*) begin
        if (a1 == 5'd0)
            _d1 = 32'd0;
        else if ((a1 == wreg) && wr)
            _d1 = wdata;
        else
            _d1 = m[a1][31:0];
    end
    always @(*) begin
        if (a2 == 5'd0)
            _d2 = 32'd0;
        else if ((a2 == wreg) && wr)
            _d2 = wdata;
        else
            _d2 = m[a2][31:0];
    end
    always @(*) begin
        if (a3 == 5'd0)
            _d3 = 32'd0;
        else if ((a3 == wreg) && wr)
            _d3 = wdata;
        else
            _d3 = m[a3][31:0];
    end
```

```
    assign d1 = _d1;
    assign d2 = _d2;
    assign d3 = _d3;

    always @(posedge clk) begin
        if (wr && wreg != 5'd0) begin
            m[wreg] <= wdata;
        end
    end
endmodule
```

MUX32_2x1：

```
module MUX32_2x1(input wire [31:0]A,
                 input wire [31:0]B,
                 input wire sel,
                 output wire [31:0]C
    );

assign C[31]=(A[31]&~sel)|(B[31]&sel);
assign C[30]=(A[30]&~sel)|(B[30]&sel);
assign C[29]=(A[29]&~sel)|(B[29]&sel);
assign C[28]=(A[28]&~sel)|(B[28]&sel);
assign C[27]=(A[27]&~sel)|(B[27]&sel);
assign C[26]=(A[26]&~sel)|(B[26]&sel);
assign C[25]=(A[25]&~sel)|(B[25]&sel);
assign C[24]=(A[24]&~sel)|(B[24]&sel);
assign C[23]=(A[23]&~sel)|(B[23]&sel);
assign C[22]=(A[22]&~sel)|(B[22]&sel);
assign C[21]=(A[21]&~sel)|(B[21]&sel);
assign C[20]=(A[20]&~sel)|(B[20]&sel);
assign C[19]=(A[19]&~sel)|(B[19]&sel);
assign C[18]=(A[18]&~sel)|(B[18]&sel);
assign C[17]=(A[17]&~sel)|(B[17]&sel);
assign C[16]=(A[16]&~sel)|(B[16]&sel);
assign C[15]=(A[15]&~sel)|(B[15]&sel);
assign C[14]=(A[14]&~sel)|(B[14]&sel);
assign C[13]=(A[13]&~sel)|(B[13]&sel);
assign C[12]=(A[12]&~sel)|(B[12]&sel);
assign C[11]=(A[11]&~sel)|(B[11]&sel);
assign C[10]=(A[10]&~sel)|(B[10]&sel);
assign C[9]=(A[9]&~sel)|(B[9]&sel);
```

```
assign C[8]=(A[8]&~sel)|(B[8]&sel);
assign C[7]=(A[7]&~sel)|(B[7]&sel);
assign C[6]=(A[6]&~sel)|(B[6]&sel);
assign C[5]=(A[5]&~sel)|(B[5]&sel);
assign C[4]=(A[4]&~sel)|(B[4]&sel);
assign C[3]=(A[3]&~sel)|(B[3]&sel);
assign C[2]=(A[2]&~sel)|(B[2]&sel);
assign C[1]=(A[1]&~sel)|(B[1]&sel);
assign C[0]=(A[0]&~sel)|(B[0]&sel);
endmodule
```

catch_28_4:

```
module catch_28_4(input wire [27:0]A,
                  input wire [3:0]B,
                  output wire [31:0]C
   );
assign C[31:28]=B[3:0];
assign C[27:0]=A[27:0];

endmodule
```

MUX32_4x1:

```
module MUX32_4x1(input wire [31:0]A,
                 input wire [31:0]B,
                 input wire [31:0]C,
                 input wire [31:0]D,
                 input wire [1:0]sel,
                 output wire [31:0]E
   );
wire [31:0]temp0;
wire [31:0]temp1;

MUX32_2x1 m0(A[31:0],B[31:0],sel[0],temp0[31:0]);
MUX32_2x1 m1(C[31:0],D[31:0],sel[0],temp1[31:0]);
MUX32_2x1 m2(temp0[31:0],temp1[31:0],sel[1],E[31:0]);

endmodule
```

MUX3_2x1:

```
module MUX3_2x1(input wire [2:0]A,
```

```
                input wire [2:0]B,
                input wire sel,
                output wire [2:0]C
    );
assign C[2]=(A[2]&~sel)|(B[2]&sel);
assign C[1]=(A[1]&~sel)|(B[1]&sel);
assign C[0]=(A[0]&~sel)|(B[0]&sel);

endmodule
```

MUX5_2x1:

```
module MUX5_2x1(input wire [4:0]A,
                input wire [4:0]B,
                input wire sel,
                output wire [4:0]C
    );
assign C[4]=(A[4]&~sel)|(B[4]&sel);
assign C[3]=(A[3]&~sel)|(B[3]&sel);
assign C[2]=(A[2]&~sel)|(B[2]&sel);
assign C[1]=(A[1]&~sel)|(B[1]&sel);
assign C[0]=(A[0]&~sel)|(B[0]&sel);

endmodule
```

pcreg:

```
module pcreg(clk,pc,npc);
    input wire clk;
    input wire [31:0]npc;
    output reg [31:0]pc;
    initial pc=32'b0;
    always @(posedge clk) begin
        pc=npc;
    end
endmodule
```

shift_left16:

```
module shift_left16(input wire [15:0]A,
                output wire [31:0]B
    );
assign B[31:16]=A[15:0];
```

```
assign B[15:0]=16'b0;

endmodule
```

mem_for_data:

```
module mem_for_data(clk, a, di, wr, done);
    parameter size=12;
    input clk;
    input [size-1:0]a;
    input [31:0]di;
    input wr;
    wire[7:0] do1,do2,do3,do4;
    wire [size-3:0]a1,a2,a3,a4,na;
    assign na=a[size-1:2];
    assign a1=na+(a[0]|a[1]);
    assign a2=na+(a[0]&a[1])+((~a[0])&a[1]);
    assign a3=na+(a[0]&a[1]);
    assign a4=na;
    output [31:0]done;
    assign done={do1,do2,do3,do4};
    mem5 m5(a1,di[31:24],clk,wr,do1);
    mem6 m6(a2,di[23:16],clk,wr,do2);
    mem7 m7(a3,di[15:8],clk,wr,do3);
    mem8 m8(a4,di[7:0],clk,wr,do4);
endmodule
```

disp_control:

```
module disp_control(clk,pc, ch, d, node, segment, pcd);
    input ch,pc,clk;
    input [31:0]d, pcd;
    output reg[3:0]node;
    output reg[7:0]segment;
    wire [15:0] digit;
    reg [15:0] _digit;
    reg [3:0]  code  =  4'b0;
    reg [15:0] count = 15'b0;
    always @(posedge clk) begin
        if (ch&!pc) begin
            _digit<=d[31:16];
        end
        else if(!ch&!pc) begin
            _digit<=d[15:0];
```

```
        end
        else if(ch&pc) begin
            _digit<=pcd[31:16];
        end
        else begin
            _digit<=pcd[15:0];
        end
    end
    assign digit=_digit;
    always @(posedge clk) begin
        case (count[15:14])

            2'b00 : begin
                node <= 4'b1110;
                code <= digit[3:0];
                end
            2'b01 : begin
                node <= 4'b1101;
                code <= digit[7:4];
                end
            2'b10 : begin
                node <= 4'b1011;
                code <= digit[11:8];
                end
            2'b11 : begin
                node <= 4'b0111;
                code <= digit[15:12];
                end
        endcase
        case (code)
            4'b0000: segment <= 8'b11000000;
            4'b0001: segment <= 8'b11111001;
            4'b0010: segment <= 8'b10100100;
            4'b0011: segment <= 8'b10110000;
            4'b0100: segment <= 8'b10011001;
            4'b0101: segment <= 8'b10010010;
            4'b0110: segment <= 8'b10000010;
            4'b0111: segment <= 8'b11111000;
            4'b1000: segment <= 8'b10000000;
            4'b1001: segment <= 8'b10010000;
            4'b1010: segment <= 8'b10001000;
            4'b1011: segment <= 8'b10000011;
            4'b1100: segment <= 8'b11000110;
            4'b1101: segment <= 8'b10100001;
```

```
            4'b1110: segment <= 8'b10000110;
            4'b1111: segment <= 8'b10001110;
            default: segment <= 8'b00000000;
        endcase
        count <= count + 1;
    end

endmodule
```

Memory:

```
module memory(clk, a, di, wr, done);
    parameter size=12;
    input clk;
    input [size-1:0]a;
    input [31:0]di;
    input wr;
    wire[7:0] do1,do2,do3,do4;
    wire [size-3:0]a1,a2,a3,a4,na;
    assign na=a[size-1:2];
    assign a1=na+(a[0]|a[1]);
    assign a2=na+(a[0]&a[1])+((~a[0])&a[1]);
    assign a3=na+(a[0]&a[1]);
    assign a4=na;
    output [31:0]done;
    assign done={do1,do2,do3,do4};
    mem1 m1(a1,di[31:24],clk,wr,do1);
    mem2 m2(a2,di[23:16],clk,wr,do2);
    mem3 m3(a3,di[15:8],clk,wr,do3);
    mem4 m4(a4,di[7:0],clk,wr,do4);
endmodule
```

alu32:

```
module alu32(ins,
             A,
             B,
             O,
             Z);
input  [2:0]ins;
input  [31:0]A;
input  [31:0]B;
output reg[31:0]O;
```

```
output reg Z;
reg mode;
wire [31:0]S0;
wire [31:0]S1;
wire [31:0]S2;
wire [31:0]S3;
adder32b m1(.A(A[31:0]),
          .B(B[31:0]),
          .mode(mode),
          .CF(),
          .S(S0[31:0]),
          .SF(),
          .ZF());
and32  m2(.A(A[31:0]),
           .B(B[31:0]),
           .O(S1[31:0])
          );
or32   m3(.A(A[31:0]),
           .B(B[31:0]),
           .O(S2[31:0])
          );
always@*begin
case(ins[2:0])
   3'b010:begin//加法
          mode<=0;
          O<=S0;
          end
   3'b110:begin//减法
          mode<=1;
          O<=S0;

          end
   3'b000:begin//and
          O<=S1;
          end
   3'b001:begin//or
          O<=S2;
          end
   3'b011:begin
          O<=A[31:0]^B[31:0];
          end
   3'b111:begin//slt
          O<=(A[31:0]<B[31:0])?1:0;
          end
```

```
endcase
          Z<=(O==0)?1:0;
end
endmodule
```

ALUcontrol:

```
module ALUcontrol(ALUop, Func, ALUoper,jr
   );
input [1:0] ALUop;
input [5:0] Func;
output [2:0] ALUoper;
output jr;

or (t1, Func[0], Func[3]);
and(t2, Func[1], ALUop[1]);
or (ALUoper[2], t2, t3);
and(t3, ALUop[0], ~ALUop[1]);
or (ALUoper[1], ~Func[2], ~ALUop[1]);
and(ALUoper[0], t1, ALUop[1]);
and(jr,~Func[5],~Func[4],Func[3],~Func[2],~Func[1],~Func[0],ALUop[1])
;
endmodule
```

Control:

```
module Control(input wire [5:0]op,
               output wire RegDst,
               output wire Jump,
               output wire ALUsrc,
               output wire [1:0]ALUop,
               output wire [1:0]MemToReg,
               output wire MemWrite,
               output wire Branch,
               output wire RegWrite,
               output wire [1:0]ilei,
               output wire jal,
               output wire bne,
               output wire [1:0]lwsh,
               output wire i
   );
wire r,lw,sw,beq,j,addi,lui,andi,ori,xori,slti,lb,lh,sb,sh,l,s;
```

```
and (r, ~op[5], ~op[4], ~op[3], ~op[2], ~op[1], ~op[0]);
and (lw, op[5], ~op[4], ~op[3], ~op[2], op[1], op[0]);
and (sw, op[5], ~op[4], op[3], ~op[2], op[1], op[0]);
and (beq, ~op[5], ~op[4], ~op[3], op[2], ~op[1], ~op[0]);
and (j, ~op[5], ~op[4], ~op[3], ~op[2], op[1], ~op[0]);
and (addi, ~op[5], ~op[4], op[3], ~op[2], ~op[1], ~op[0]);
and (slti, ~op[5], ~op[4], op[3], ~op[2], op[1], ~op[0]);
and (jal, ~op[5], ~op[4], ~op[3], ~op[2], op[1], op[0]);
and (bne, ~op[5], ~op[4], ~op[3], op[2], ~op[1], op[0]);
and (lui, ~op[5], ~op[4], op[3], op[2], op[1], op[0]);
and (andi, ~op[5], ~op[4], op[3], op[2], ~op[1], ~op[0]);
and (ori, ~op[5], ~op[4], op[3], op[2], ~op[1], op[0]);
and (xori, ~op[5], ~op[4], op[3], op[2], op[1], ~op[0]);
and (lb, op[5], ~op[4], ~op[3], ~op[2], ~op[1], ~op[0]);
and (lh, op[5], ~op[4], ~op[3], ~op[2], ~op[1], op[0]);
and (sb, op[5], ~op[4], op[3], ~op[2], ~op[1], ~op[0]);
and (sh, op[5], ~op[4], op[3], ~op[2], ~op[1], op[0]);

or(l,lw,lb,lh);
or(s,sw,sb,sh);
assign RegDst = r;
or (Jump, j, jal);
or (ALUsrc, l, s, addi, slti, andi, ori, xori);
assign ALUop[1]=r;
assign ALUop[0]=beq;
or (MemToReg[0],l,lui);
or (MemToReg[1],jal,lui);
assign MemWrite=sw;
assign Branch=beq;
or (RegWrite,r,l,addi,slti,jal,lui,ori,xori,andi,xori);
or (ilei[1],ori,xori);
or (ilei[0],andi,xori);
or (i,andi,ori,xori,slti);
or (lwsh[1],lh,sh);
or (lwsh[0],lb,sb);

endmodule
```

ilei_cnt:

```
module ilei_cnt(input wire[1:0]cnt,
                output wire[2:0]A
```

```
    );
assign A=(cnt==0)?3'b111:3'bz;
assign A=(cnt==2'b1)?3'b0:3'bz;
assign A=(cnt==2'b10)?3'b1:3'bz;
assign A=(cnt==2'b11)?3'b11:3'bz;

endmodule
```

lwbh_cnt:

```
module lwbh_cnt(input wire[1:0]cnt,
                input wire[31:0]data,
                output wire[31:0]result
    );
assign result[31:0]=(cnt==0)?data[31:0]:32'bz;
assign result[31:0]=(cnt==2'b1)?{24'b0,data[7:0]}:32'bz;
assign result[31:0]=(cnt==2'b10)?{16'b0,data[15:0]}:32'bz;

endmodule
```

adder32b:

```
`timescale  100 ps / 10 ps
module AND8_HXILINX_adder32 (O, I0, I1, I2, I3, I4, I5, I6, I7);
   output O;
   input I0;
   input I1;
   input I2;
   input I3;
   input I4;
   input I5;
   input I6;
   input I7;
assign O = I0 && I1 && I2 && I3 && I4 && I5 && I6 && I7;

endmodule

module add1_MUSER_add4(A0,
                   B0,
                   C0,
                   C1,
                   S0);
```

```
    input A0;
    input B0;
    input C0;
   output C1;
   output S0;

   wire XLXN_1;
   wire XLXN_14;
   wire XLXN_15;
   wire XLXN_16;

   XOR2  XLXI_1 (.I0(B0),
                 .I1(A0),
                 .O(XLXN_1));
   XOR2  XLXI_2 (.I0(C0),
                 .I1(XLXN_1),
                 .O(S0));
   AND2  XLXI_3 (.I0(B0),
                 .I1(A0),
                 .O(XLXN_14));
   AND2  XLXI_4 (.I0(C0),
                 .I1(A0),
                 .O(XLXN_15));
   AND2  XLXI_5 (.I0(C0),
                 .I1(B0),
                 .O(XLXN_16));
   OR3  XLXI_6 (.I0(XLXN_16),
                .I1(XLXN_15),
                .I2(XLXN_14),
                .O(C1));
endmodule

module add4b(A,
             B,
             CIN,
             COUT,
             S);

    input [3:0] A;
    input [3:0] B;
    input CIN;
   output COUT;
   output [3:0] S;
```

```
  wire XLXN_42;
  wire XLXN_46;
  wire XLXN_47;
  wire XLXN_48;
  wire XLXN_72;
  wire XLXN_76;
  wire XLXN_77;
  wire XLXN_78;
  wire XLXN_113;
  wire XLXN_114;
  wire XLXN_116;
  wire XLXN_117;
  wire XLXN_126;
  wire XLXN_140;
  wire XLXN_141;
  wire XLXN_142;
  wire XLXN_149;
  wire XLXN_150;
  wire XLXN_151;
  wire XLXN_152;
  wire XLXN_153;

  add1_MUSER_add4  XLXI_1 (.A0(A[1]),
                           .B0(B[1]),
                           .C0(XLXN_48),
                           .C1(),
                           .S0(S[1]));
  add1_MUSER_add4  XLXI_2 (.A0(A[3]),
                           .B0(B[3]),
                           .C0(XLXN_117),
                           .C1(),
                           .S0(S[3]));
  add1_MUSER_add4  XLXI_3 (.A0(A[2]),
                           .B0(B[2]),
                           .C0(XLXN_78),
                           .C1(),
                           .S0(S[2]));
  add1_MUSER_add4  XLXI_4 (.A0(A[0]),
                           .B0(B[0]),
                           .C0(CIN),
                           .C1(),
                           .S0(S[0]));
  AND2  XLXI_5 (.I0(B[0]),
                .I1(A[0]),
```

```
           .O(XLXN_47));
OR2  XLXI_6 (.I0(B[0]),
           .I1(A[0]),
           .O(XLXN_42));
AND2  XLXI_11 (.I0(XLXN_42),
            .I1(CIN),
            .O(XLXN_46));
OR2  XLXI_12 (.I0(XLXN_47),
           .I1(XLXN_46),
           .O(XLXN_48));
AND2  XLXI_13 (.I0(B[1]),
            .I1(A[1]),
            .O(XLXN_76));
OR2  XLXI_14 (.I0(B[1]),
           .I1(A[1]),
           .O(XLXN_141));
AND2  XLXI_15 (.I0(XLXN_47),
            .I1(XLXN_141),
            .O(XLXN_72));
AND3  XLXI_16 (.I0(CIN),
            .I1(XLXN_42),
            .I2(XLXN_141),
            .O(XLXN_77));
OR3  XLXI_17 (.I0(XLXN_77),
           .I1(XLXN_76),
           .I2(XLXN_72),
           .O(XLXN_78));
OR2  XLXI_18 (.I0(B[2]),
           .I1(A[2]),
           .O(XLXN_142));
AND2  XLXI_19 (.I0(B[2]),
            .I1(A[2]),
            .O(XLXN_126));
AND2  XLXI_20 (.I0(XLXN_76),
            .I1(XLXN_142),
            .O(XLXN_114));
AND3  XLXI_21 (.I0(XLXN_47),
            .I1(XLXN_141),
            .I2(XLXN_142),
            .O(XLXN_116));
AND4  XLXI_22 (.I0(XLXN_142),
            .I1(XLXN_141),
            .I2(XLXN_42),
            .I3(CIN),
```

```
            .O(XLXN_113));
  OR4  XLXI_23 (.I0(XLXN_116),
           .I1(XLXN_126),
           .I2(XLXN_114),
           .I3(XLXN_113),
           .O(XLXN_117));
  OR2  XLXI_24 (.I0(B[3]),
           .I1(A[3]),
           .O(XLXN_140));
  AND2  XLXI_25 (.I0(B[3]),
           .I1(A[3]),
           .O(XLXN_151));
  AND2  XLXI_26 (.I0(XLXN_140),
           .I1(XLXN_126),
           .O(XLXN_150));
  AND3  XLXI_27 (.I0(XLXN_76),
           .I1(XLXN_142),
           .I2(XLXN_140),
           .O(XLXN_152));
  AND4  XLXI_28 (.I0(XLXN_140),
           .I1(XLXN_141),
           .I2(XLXN_142),
           .I3(XLXN_47),
           .O(XLXN_149));
  AND5  XLXI_29 (.I0(XLXN_140),
           .I1(XLXN_142),
           .I2(XLXN_141),
           .I3(XLXN_42),
           .I4(CIN),
           .O(XLXN_153));
  OR5  XLXI_30 (.I0(XLXN_153),
           .I1(XLXN_152),
           .I2(XLXN_151),
           .I3(XLXN_150),
           .I4(XLXN_149),
           .O(COUT));
endmodule

module zero_4_MUSER_adder32(A,
                      z);

   input [3:0] A;
  output z;
```

```
   wire XLXN_6;

   OR4  XLXI_1 (.I0(A[0]),
               .I1(A[1]),
               .I2(A[2]),
               .I3(A[3]),
               .O(XLXN_6));
   INV  XLXI_2 (.I(XLXN_6),
               .O(z));
endmodule

module xor_4bit_MUSER_adder32(A,
                              mode,
                              B);

    input [3:0] A;
    input mode;
   output [3:0] B;


   XOR2  XLXI_1 (.I0(mode),
                .I1(A[0]),
                .O(B[0]));
   XOR2  XLXI_2 (.I0(mode),
                .I1(A[1]),
                .O(B[1]));
   XOR2  XLXI_3 (.I0(mode),
                .I1(A[2]),
                .O(B[2]));
   XOR2  XLXI_4 (.I0(mode),
                .I1(A[3]),
                .O(B[3]));
endmodule

module adder32b(A,
               B,
               mode,
               CF,
               S,
               SF,
               ZF);

    input [31:0] A;
    input [31:0] B;
```

```
 input mode;
output CF;
output [31:0] S;
output SF;
output ZF;

wire temp1;
wire [3:0] XLXN_20;
wire [3:0] XLXN_21;
wire [3:0] XLXN_22;
wire [3:0] XLXN_23;
wire [3:0] XLXN_24;
wire [3:0] XLXN_25;
wire [3:0] XLXN_26;
wire [3:0] XLXN_27;
wire XLXN_28;
wire XLXN_29;
wire XLXN_30;
wire XLXN_31;
wire XLXN_34;
wire XLXN_35;
wire XLXN_83;
wire z0;
wire z1;
wire z2;
wire z3;
wire z4;
wire z5;
wire z6;
wire z7;
wire [31:0] S_DUMMY;

assign S[31:0] = S_DUMMY[31:0];
xor_4bit_MUSER_adder32  XLXI_17 (.A(B[3:0]),
                                 .mode(mode),
                                 .B(XLXN_20[3:0]));
xor_4bit_MUSER_adder32  XLXI_18 (.A(B[7:4]),
                                 .mode(mode),
                                 .B(XLXN_21[3:0]));
xor_4bit_MUSER_adder32  XLXI_19 (.A(B[11:8]),
                                 .mode(mode),
                                 .B(XLXN_22[3:0]));
xor_4bit_MUSER_adder32  XLXI_20 (.A(B[15:12]),
                                 .mode(mode),
```

```
                           .B(XLXN_23[3:0]));
xor_4bit_MUSER_adder32  XLXI_21 (.A(B[31:28]),
                           .mode(mode),
                           .B(XLXN_24[3:0]));
xor_4bit_MUSER_adder32  XLXI_22 (.A(B[27:24]),
                           .mode(mode),
                           .B(XLXN_25[3:0]));
xor_4bit_MUSER_adder32  XLXI_23 (.A(B[23:20]),
                           .mode(mode),
                           .B(XLXN_26[3:0]));
xor_4bit_MUSER_adder32  XLXI_24 (.A(B[19:16]),
                           .mode(mode),
                           .B(XLXN_27[3:0]));
XOR2  XLXI_25 (.I0(XLXN_83),
           .I1(mode),
           .O(CF));
BUF  XLXI_26 (.I(S_DUMMY[31]),
          .O(SF));
zero_4_MUSER_adder32  XLXI_27 (.A(S_DUMMY[11:8]),
                           .z(z2));
zero_4_MUSER_adder32  XLXI_28 (.A(S_DUMMY[7:4]),
                           .z(z1));
zero_4_MUSER_adder32  XLXI_29 (.A(S_DUMMY[3:0]),
                           .z(z0));
zero_4_MUSER_adder32  XLXI_30 (.A(S_DUMMY[23:20]),
                           .z(z5));
zero_4_MUSER_adder32  XLXI_31 (.A(S_DUMMY[19:16]),
                           .z(z4));
zero_4_MUSER_adder32  XLXI_32 (.A(S_DUMMY[15:12]),
                           .z(z3));
zero_4_MUSER_adder32  XLXI_33 (.A(S_DUMMY[31:28]),
                           .z(z7));
zero_4_MUSER_adder32  XLXI_34 (.A(S_DUMMY[27:24]),
                           .z(z6));
(* HU_SET = "XLXI_35_0" *)
AND8_HXILINX_adder32  XLXI_35 (.I0(z0),
                           .I1(z1),
                           .I2(z2),
                           .I3(z3),
                           .I4(z4),
                           .I5(z5),
                           .I6(z6),
                           .I7(z7),
                           .O(ZF));
```

```
   add4b  XLXI_36 (.A(A[3:0]),
                        .B(XLXN_20[3:0]),
                        .CIN(mode),
                        .COUT(XLXN_28),
                        .S(S_DUMMY[3:0]));
   add4b  XLXI_37 (.A(A[7:4]),
                        .B(XLXN_21[3:0]),
                        .CIN(XLXN_28),
                        .COUT(XLXN_29),
                        .S(S_DUMMY[7:4]));
   add4b  XLXI_38 (.A(A[11:8]),
                        .B(XLXN_22[3:0]),
                        .CIN(XLXN_29),
                        .COUT(XLXN_30),
                        .S(S_DUMMY[11:8]));
   add4b  XLXI_39 (.A(A[15:12]),
                        .B(XLXN_23[3:0]),
                        .CIN(XLXN_30),
                        .COUT(temp1),
                        .S(S_DUMMY[15:12]));
   add4b  XLXI_40 (.A(A[31:28]),
                        .B(XLXN_24[3:0]),
                        .CIN(XLXN_31),
                        .COUT(XLXN_83),
                        .S(S_DUMMY[31:28]));
   add4b  XLXI_41 (.A(A[27:24]),
                        .B(XLXN_25[3:0]),
                        .CIN(XLXN_34),
                        .COUT(XLXN_31),
                        .S(S_DUMMY[27:24]));
   add4b  XLXI_42 (.A(A[23:20]),
                        .B(XLXN_26[3:0]),
                        .CIN(XLXN_35),
                        .COUT(XLXN_34),
                        .S(S_DUMMY[23:20]));
   add4b  XLXI_43 (.A(A[19:16]),
                        .B(XLXN_27[3:0]),
                        .CIN(temp1),
                        .COUT(XLXN_35),
                        .S(S_DUMMY[19:16]));
endmodule
```

UCF 文件：

```
NET "clk_100mhz"     LOC = AC18     | IOSTANDARD = LVCMOS18 ;
NET "SW[0]"             LOC = AA10  | IOSTANDARD = LVCMOS15 ;
NET "SW[1]"             LOC = AB10  | IOSTANDARD = LVCMOS15 ;
NET "SW[2]"             LOC = AA13  | IOSTANDARD = LVCMOS15 ;
NET "SW[3]"             LOC = AA12  | IOSTANDARD = LVCMOS15 ;
NET "SW[4]"             LOC = Y13   | IOSTANDARD = LVCMOS15 ;
NET "pc"             LOC = Y12   | IOSTANDARD = LVCMOS15 ;
NET "ch"             LOC = AD11  | IOSTANDARD = LVCMOS15 ;
NET "clk"            LOC = AD10  | IOSTANDARD = LVCMOS15 ;
NET "clk" CLOCK_DEDICATED_ROUTE = FALSE;

NET "Buzzer"         LOC = AF24  | IOSTANDARD = LVCMOS33 ;
NET "SEGMENT[0]"     LOC = AB22  | IOSTANDARD = LVCMOS33 ;#a
NET "SEGMENT[1]"     LOC = AD24 | IOSTANDARD = LVCMOS33 ;#b
NET "SEGMENT[2]"     LOC = AD23 | IOSTANDARD = LVCMOS33 ;
NET "SEGMENT[3]"     LOC = Y21  | IOSTANDARD = LVCMOS33 ;
NET "SEGMENT[4]"     LOC = W20  | IOSTANDARD = LVCMOS33 ;
NET "SEGMENT[5]"     LOC = AC24 | IOSTANDARD = LVCMOS33 ;
NET "SEGMENT[6]"     LOC = AC23 | IOSTANDARD = LVCMOS33 ;#g
NET "SEGMENT[7]"     LOC = AA22 | IOSTANDARD = LVCMOS33 ;#point

NET "AN[0]"          LOC = AD21  | IOSTANDARD = LVCMOS33 ;
NET "AN[1]"          LOC = AC21  | IOSTANDARD = LVCMOS33 ;
NET "AN[2]"          LOC = AB21  | IOSTANDARD = LVCMOS33 ;
NET "AN[3]"          LOC = AC22  | IOSTANDARD = LVCMOS33 ;
```

各个预先导入的 coe 文件

mem1.coe

```
memory_initialization_radix = 16;
memory_initialization_vector =
20,3c,29,21,2,20,31,35,39,1,1,21,1,2,0,8,0,0,1,80,84,a0,a4,2,15,0,0,2
3,2,af,af,0,11,20,23,2,3,20,2,c,8f,8f,23,2,0,2,3;
```

mem2.coe

```
memory_initialization_radix = 16;
memory_initialization_vector =
20,3c,29,21,2,20,31,35,39,1,1,21,1,2,0,8,0,0,1,80,84,a0,a4,2,15,0,0,2
3,2,af,af,0,11,20,23,2,3,20,2,c,8f,8f,23,2,0,2,3;
```

mem3.coe

```
memory_initialization_radix = 16;
memory_initialization_vector =
0,0,0,0,48,0,0,0,0,58,48,0,80,48,f8,0,0,0,80,0,0,0,0,50,0,0,0,ff,e8,0
,0,40,0,0,0,e8,0,ff,20,0,0,0,0,e8,80,10,0;
```

mem4.coe

```
memory_initialization_radix = 16;
memory_initialization_vector =
c8,2,7b,1,20,c,3,f,f,22,24,1,28,20,20,12,0,0,25,0,4,32,32,20,2,0,0,f8
,20,4,0,2a,4,1,8,20,8,ff,20,1b,0,4,8,20,20,20,8;
```

mem5.coe

```
memory_initialization_radix = 16;
memory_initialization_vector =
0,0,0,0,0,0,0,0,0,0;
```

mem6.coe

```
memory_initialization_radix = 16;
memory_initialization_vector =
0,0,0,0,0,0,0,0,0,0;
```

mem7.coe

```
memory_initialization_radix = 16;
memory_initialization_vector =
0,0,0,0,0,0,0,0,0,0;
```

mem8.coe

```
memory_initialization_radix = 16;
memory_initialization_vector =
1,2,3,4,5,6,7,8,9,a;
```