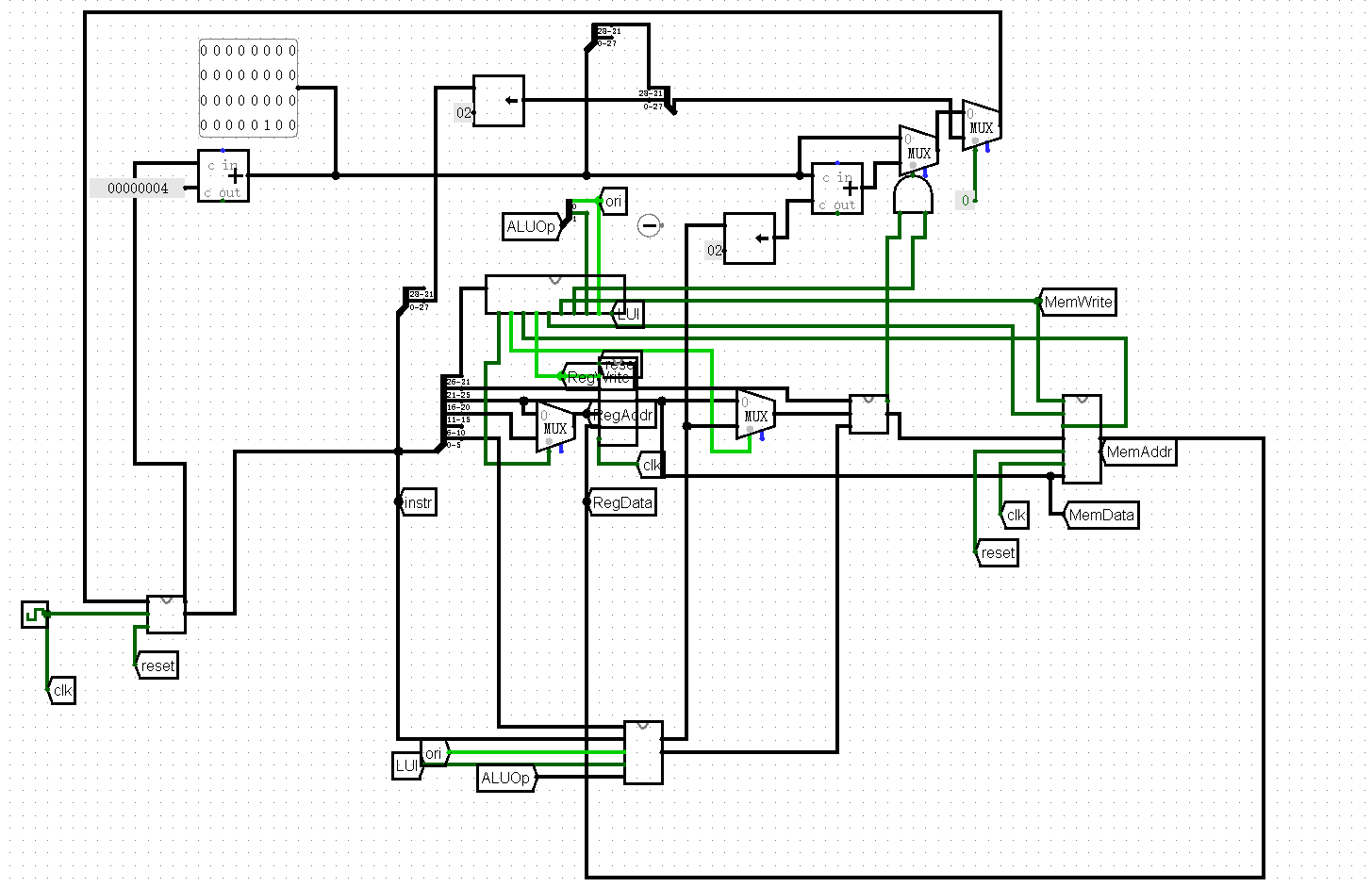
# CPU设计文档

## 模块规格

Verilog cpu设计参考了上一章logisim的设计如图：



1. IFU（取指令单元）：内部包括 PC（程序计数器）、IM(指令存储器)及相关逻辑。

PC 用寄存器实现，具有复位功能。

起始地址：0x00000000。

IM用 ROM 实现，容量为 32bit \* 32。

因 IM 实际地址宽度仅为 5 位，故使用恰当的方法将 PC 中储存的地址同 IM 联系起来。

表1 IFU

|  |  |  |
| --- | --- | --- |
| 序号 | Input/output | 功能描述 |
| 1 | reset | 当复位信号有效时，PC被设置为0x00000000 |
| 2 | clk | 时钟信号 |
| 3 | NEWpc | 下一个pc值，在时钟上升沿来时存入pc |
| 4 | pc | 输出pc信号 |
| 5 | out | 输出IM中取出的指令 |

设计：

module IFU(

input [31:0] NEWpc,

input clk,

input reset,

output [31:0] pc,

output [31:0] out

);

reg[31:0] IM[1023:0];

reg[31:0] pc\_reg;

wire[21:0] n0;

wire[9:0] add\_IM;

assign {n0,add\_IM}=pc\_reg/4;

assign pc=pc\_reg;

assign out=IM[add\_IM];

initial begin

pc\_reg=32'h00003000;

$readmemh("code.txt",IM);

end

always @(posedge clk)begin

if(reset==1) pc\_reg=32'h00003000;

else pc\_reg=NEWpc;

end

endmodule

2. GRF（通用寄存器组，也称为寄存器文件、寄存器堆）

用具有写使能的寄存器实现，寄存器总数为 32 个。

0 号寄存器的值始终保持为 0,通过使其clear端始终为1实现。其他寄存器初始值均为 0，无需专门设置。

表2 GRF

|  |  |  |
| --- | --- | --- |
| 序号 | Input/output | 功能描述 |
| 1 | reset | 当复位信号有效时，PC被设置为0x00000000 |
| 2 | clk | 时钟信号 |
| 3 | A1 | 读地址信号 |
| 4 | A2 | 读地址信号 |
| 5 | A3 | 写地址信号 |
| 6 | WD | 写使能信号 |
| 7 | RD1 | 数据输出 |
| 8 | RD2 | 数据输出 |

设计：

module grf(

input reset,

input we,

input [4:0] a1,

input [4:0] a2,

input [4:0] a3,

input [31:0] WD,

input clk,

input [31:0] wpc,

output [31:0] RD1,

output [31:0] RD2,

input jal

);

wire [4:0]add3;

wire [31:0] WD1;

reg[31:0] grf[31:0];

reg[5:0] yj0;

assign RD1=grf[a1];

assign RD2=grf[a2];

assign add3=(jal)?31:a3;

assign WD1=(jal)?(wpc+4):WD;

initial begin

for(yj0=0;yj0<32;yj0=yj0+1)

grf[yj0]=0;

end

always @(posedge clk) begin

grf[0]=0;

end

always @(posedge clk)begin

if(reset==1) begin

for(yj0=0;yj0<32;yj0=yj0+1)

grf[yj0]=0;

end

else if(we==1)begin

grf[add3]=WD1;

if({1'b0,add3}!=0)$display("@%h: $%d <= %h",wpc,{1'b0,add3},WD1);

end

end

endmodule

3. ALU（算术逻辑单元）

提供 32 位加、减、或运算及大小比较功能。

可以不支持溢出（不检测溢出）。

表3 ALU

|  |  |  |
| --- | --- | --- |
| 序号 | Input/output | 功能描述 |
| 1 | In1 | 输入数据1 |
| 2 | In2 | 输入数据2 |
| 3 | ALUcontrol | 选择输出，0000选与运算，0001选或运算，0010选加法运算，0110选减法运算 |
| 4 | Out | 输出计算结果 |
| 5 | Zero | 若两输入数据相等则输出1，否则输出0 |

设计：

module ALU(

input lui,

input [3:0] ALUcontrol,

input [31:0] in1,

input [31:0] in2,

output reg [31:0] out,

output zero

);

assign zero=(in1==in2);

wire cin;

always @(\*)begin

if(lui==1) out=in2;

else begin

case(ALUcontrol)

4'b0000:out=in1&in2;

4'b0001:out=in1|in2;

4'b0010:out=in1+in2;

4'b0110:out=in1-in2;

endcase

end

end

endmodule

4. DM（数据存储器）

使用 RAM 实现，容量为 32bit \* 32。

起始地址：0x00000000。

RAM 使用双端口模式，即设置 RAM 的 Data Interface 属性为 Separate load and store ports。

表4 DM

|  |  |  |
| --- | --- | --- |
| 序号 | Input/output | 功能描述 |
| 1 | Add | 访问地址 |
| 2 | Data | 输入数据 |
| 3 | Write | 写使能信号，当为1时，会在时钟上升沿存储数据 |
| 4 | Read | 读使能信号，从RAM中读出数据 |
| 5 | Clk | 时钟信号 |
| 6 | Reset | 同步重置信号，为1时清除RAM中所有数据 |
| 7 | Select | 选择信号，为1时选择从RAM中读出的信号作为输出，否则将输入add作为输出 |
| 8 | Out | 输出数据 |
| 9 | Memadd | 输出RAM被访问的地址 |

设计：

module DM(

input [31:0] wpc,

input [31:0] add,

input [31:0] data,

input clk,

input reset,

input write,

input read,

input select,

output [31:0] out

);

reg[31:0] DMem[1023:0];

reg [31:0] dm\_out,temp;

reg [10:0] yj;

wire[21:0] n0;

wire[9:0] add\_DM;

assign {n0,add\_DM}=add/4;

assign out = (select)?dm\_out:add;

initial begin

for(yj=0;yj<1024;yj=yj+1)

DMem[yj]=0;

end

always @(posedge clk)begin

if(reset==1)begin

for(yj=0;yj<1024;yj=yj+1)

DMem[yj]=0;

end

else if(write==1)begin

DMem[add\_DM]=data;

temp= add\_DM;

$display("@%h: \*%h <= %h",wpc,temp\*4,data);

end

end

always @(read)begin

dm\_out=DMem[add\_DM];

end

endmodule

5. EXT以及ALUController：

表5 extender

|  |  |  |
| --- | --- | --- |
| 序号 | Input/output | 功能描述 |
| 1 | dataIn | 输入指令码 |
| 2 | Ori | 当指令为ori时为1，进行0扩展，否则进行符号扩展 |
| 3 | Lui | 当指令为lui时为1，在低位扩展16个0，否则进行其他扩展 |
| 4 | Out | 输出拓展后的立即数 |

表6 ALUController

|  |  |  |
| --- | --- | --- |
| 序号 | Input/output | 功能描述 |
| 1 | Func | 输入指令码后6位 |
| 2 | ALUOp | 连接controller中的输出信号ALUOp |
| 3 | ALUout | 输出ALU操作码 |

设计：

module extender(

input [31:0] data,

input ori,

input lui,

output [31:0] out

);

wire [4:0] fake1,fake2;

wire [15:0] one,two;

wire [31:0] zero\_ext,sign\_ext,lui\_ext,three;

assign {one,two}=data;

assign zero\_ext={16'h0000,two};

assign sign\_ext={{16{two[15]}},two};

assign lui\_ext={two,16'h0000};

mux m1(32,1,ori,sign\_ext,zero\_ext,fake1,fake1,three,fake2);

mux m2(32,1,lui,three,lui\_ext,fake1,fake1,out,fake2);

endmodule

module ALUcontrol(

input [1:0] ALUOp,

input [5:0] func,

output reg [3:0] ALUfunc

);

always @(\*)begin

if(ALUOp==2'b00)

ALUfunc=4'b0010;

else if(ALUOp==2'b01)

ALUfunc=4'b0001;

else if(ALUOp==2'b10)begin

if(func==6'b100001) ALUfunc=4'b0010;

else if(func==6'b100011) ALUfunc=4'b0110;

else if(func==6'b001000) ALUfunc=4'b0010;

end

end

endmodule

## 控制器设计

真值表：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | R-mat | Ori | Lw | Sw | Beq | Lui |
| RegDst | 1 | 0 | 0 | X | x | 0 |
| ALUSrc | 0 | 1 | 1 | 1 | 0 | 1 |
| MentoReg | 0 | 0 | 1 | X | x | 0 |
| RegWrite | 1 | 1 | 1 | 0 | 0 | 1 |
| MenRead | 0 | 0 | 1 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 1 | 0 | 0 |
| Branch | 0 | 0 | 0 | 0 | 1 | 0 |
| ALUOp1 | 1 | 0 | 0 | 0 | 0 | 0 |
| ALUOp2 | 0 | 1 | 0 | 0 | 0 | 0 |
| LUI | 0 | 0 | 0 | 0 | 0 | 1 |

使用Verilog行为级描述输出即可：

module control(

input [31:0]ins,

input [5:0] mat,

output reg RegDst,

output reg ALUSrc,

output reg MemtoReg,

output reg RegWrite,

output reg MemRead,

output reg MemWrite,

output reg Branch,

output reg ALUOp1,

output reg ALUOp2,

output reg Lui,

output reg jal,

output reg jr

);

always @(\*)begin

case(mat)

6'b000000:

begin

RegDst=1;

ALUSrc=0;

MemtoReg=0;

RegWrite=1;

MemRead=0;

MemWrite=0;

Branch=0;

ALUOp1=1;

ALUOp2=0;

Lui=0;

jal=0;

end

6'b001101:

begin

RegDst=0;

ALUSrc=1;

MemtoReg=0;

RegWrite=1;

MemRead=0;

MemWrite=0;

Branch=0;

ALUOp1=0;

ALUOp2=1;

Lui=0;

jal=0;

end

6'b100011:

begin

RegDst=0;

ALUSrc=1;

MemtoReg=1;

RegWrite=1;

MemRead=1;

MemWrite=0;

Branch=0;

ALUOp1=0;

ALUOp2=0;

Lui=0;

jal=0;

end

6'b101011:

begin

RegDst=0;

ALUSrc=1;

MemtoReg=0;

RegWrite=0;

MemRead=0;

MemWrite=1;

Branch=0;

ALUOp1=0;

ALUOp2=0;

Lui=0;

jal=0;

end

6'b000100:

begin

RegDst=0;

ALUSrc=0;

MemtoReg=0;

RegWrite=0;

MemRead=0;

MemWrite=0;

Branch=1;

ALUOp1=0;

ALUOp2=0;

Lui=0;

jal=0;

end

6'b001111:

begin

RegDst=0;

ALUSrc=1;

MemtoReg=0;

RegWrite=1;

MemRead=0;

MemWrite=0;

Branch=0;

ALUOp1=0;

ALUOp2=0;

Lui=1;

jal=0;

end

6'b000011:

begin

RegDst=0;

ALUSrc=0;

MemtoReg=0;

RegWrite=1;

MemRead=0;

MemWrite=0;

Branch=0;

ALUOp1=0;

ALUOp2=0;

Lui=0;

jal=1;

end

endcase

if(mat==6'b000000&&ins[5:0]==6'b001000) jr=1;

else jr=0;

end

endmodule

其中，lui、ALUOp2(ori)用于extender中对扩展方式的选择，可视为ExtOp信号。

ALUOp指明ALU的运算类型：

00：访存指令所需的加法 

01：ori指令所需的或运算 

10：R型指令功能码字段决定

## 测试程序

ori $t0,$0,2

ori $t1,$0,4

beq $t0,$t1,jump1

addu $a0,$t0,$t1

subu $a0,$t1,$t0

jump1:

ori $t3,$0,3

ori $a1,$0,2

beq $a0,$a1,jump2

ori $t7,$0,8

ori $t7,$0,9

jump2:

ori $ra,$0,0x303c

jr $ra

ori $a0,$0,1

ori $a0,$0,2

ori $a0,$0,3

ori $a0,$0,4

ori $a0,$0,2026

sw $a0,0($0)

lw $a1,0($0)

lw $a2,0($0)

beq $a1,$a2,branch

ori $t0,$0,1

ori $t1,$0,2

branch:

ori $t1,$0,3

jal jump

lui $s1,1

lui $s2,2

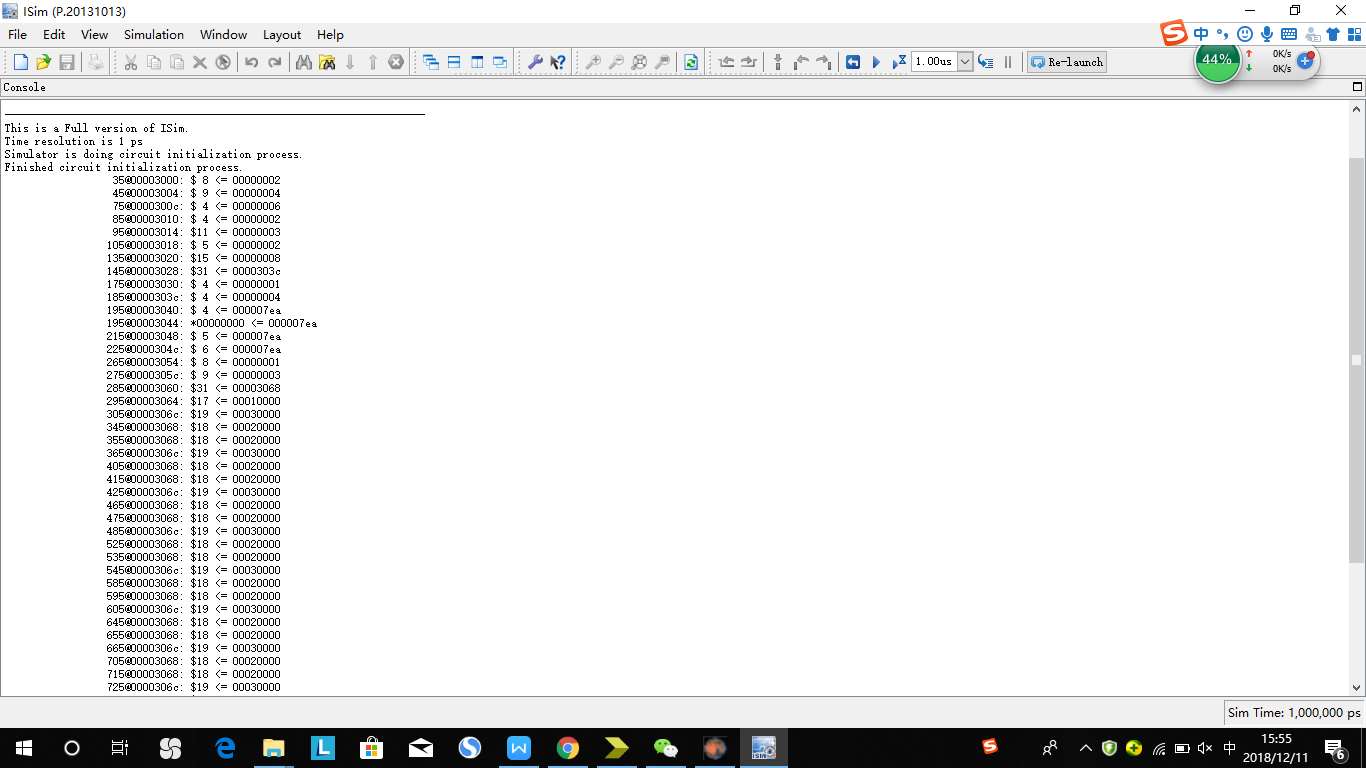
jump:

lui $s3,3

j finish

finish:

jr $ra



思考题

数据通路设计思考题：

1. 根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？

32=8\*4，所以地址需要除以四。来自ALU的输出

1. 在相应的部件中，reset的优先级比其他控制信号（不包括clk信号）都要高，且相应的设计都是同步复位。清零信号reset是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

针对IFU、DM和GRF。这些部件清零后CPU才能回到初始状态。

控制器设计思考题：

1. （1）用case语句直接对所有输出端赋值，代码见上文。

（2）用if-else语句直接对所有输出赋值，代码样例如下：

If(mat==6'b000000)

begin

RegDst=1;

ALUSrc=0;

MemtoReg=0;

RegWrite=1;

MemRead=0;

MemWrite=0;

Branch=0;

ALUOp1=1;

ALUOp2=0;

Lui=0;

jal=0;

end

（3）使用assign，代码样例如下：

Assign R\_mat=~mat[0]~mat[1]~mat[2]~mat[3]~mat[4]~mat[5];

……

Assign RegDst=R\_mat;

……..

1. 1，2方式无太大区别，相比3更加清晰明了，但是修改或者增加信号比较麻烦。

­