CPU 设计文档

一、 数据通路设计

（1）pc（程序计数器）

模块端口说明如下：

表 1　pc 端口说明

序号	信号名	方向	描述
1	Addr[25:0]	I	当前32位指令的低26位
2	R_Addr[31:0]	I	保存在寄存器中的地址
3	nPC_Op[1:0]	I	选择下一个pc的值 00 ： 选择PC+4作为下一个PC的值 01 ： 选择PC + sign_extend(Addr[15:0]||02)作为下一个PC的值 10 ：选择{PC[31:28],Addr}作为下一个PC的值 11： 选择R_Addr作为下一个PC的值
4	Clk	I	时钟信号
5	Reset	I	复位信号
6	PC[31:0]	O	当前的PC值
7	PC_plus4[31:0]	O	当前的PC值加4

模块功能定义如下：

表 2　pc 功能定义

序号	功能名称	功能描述
1	更新PC值	当时钟上升沿到来时，根据选择信号更新当前PC的值
2	输出	输出当前PC的值和PC+4的值

（2） im（指令存储器）

模块端口说明如下：

表 3　im 端口说明

序号	端口名	方向	描述
1	Addr[11:2]	I	当前PC的[11:2]位
2	Instr[31:0]	O	指令存储器中以Addr为地址的指令

模块功能定义如下：

表 4　im 功能定义

序号	功能名称	功能描述
1	初始化	将code.txt中的内容读入指令存储器中
2	输出	输出指令存储器中Addr所对应地址的指令的值

（2）grf（通用寄存器组）

模块端口说明如下：

表 5　grf 端口说明

序号	信号名	方向	描述
1	RAddr1[4:0]	I	读寄存器地址1
2	RAddr2[4:0]	I	读寄存器地址2
3	WAddr[4:0]	I	写寄存器地址
4	WData[31:0]	I	写入寄存器的数据
5	RegWrite	I	寄存器写使能信号
6	Clk	I	时钟信号
7	RData1[31:0]	O	输出地址RAddr1的寄存器中的数据
8	RData2[31:0]	O	输出地址RAddr2的寄存器中的数据

模块功能定义如下：

表 6　grf 功能定义

序号	功能名称	功能描述
1	读寄存器	输出端口RData1和RData2分别输出以输入信号RAddr1和RAddr2为地址的寄存器中的数据
2	写寄存器	当始终上升沿到来时，若写使能信号为1，则将输入信号WData中的数据写入以输入信号WAddr为地址的寄存器中

（3）alu（算术逻辑单元）

模块端口说明如下：

表 7　alu 端口说明

序号	信号名	方向	描述
1	Data1[31:0]	I	参与ALU运算的第一个值
2	Data2[31:0]	I	参与ALU运算的第二个值
3	ALUOp[1:0]	I	ALU功能的选择信号 00 ： ALU进行加法运算 01 ： ALU进行减法运算 10 ： ALU进行或运算 11 ： ALU进行与运算
4	ALUResult[31:0]	O	ALU的计算结果
5	Zero	O	如果ALU的计算结果为0，则输出1，否则输出0

模块功能定义如下：

表 8　alu 功能定义

序号	功能名称	功能描述
1	加法运算	ALUResult = Data1 + Data2
2	减法运算	AUResult = Data1 - Data2
3	与运算	ALUResult = Data1 & Data2
4	或运算	ALUResult = Data1 | Data2

（3）dm（数据存储器）

模块端口说明如下：

表 9　dm 端口说明

序号	信号名	方向	描述
1	Addr[4:0]	I	数据存储器读写的地址
2	WData[31:0]	I	将要写进数据存储器的数据
3	MemWrite	I	数据存储器的写使能端
4	Clk	I	时钟信号
5	Reset	I	复位信号
6	RData[31:0]	O	输出从数据存储器中读取的值

模块功能定义如下：

表 10　dm 功能定义

序号	功能名称	功能描述
1	读数据存储器	输出端口Data输出数据存储器在地址为MemAddr处的数据
2	写数据存储器	当时钟上升沿到来时，若MemWrite为1，且Reset信号为0，则将输入信号MemData中的数据写入数据存储器在MemAddr所对应的地址中
3	复位	当时钟上升沿到来时，若Reset信号为1，将数据存储器的内容置为0

（4）EXT（数据扩展单元）

模块端口说明如下：

表 11　ext 端口说明

序号	信号名	方向	描述
1	In[15:0]	I	扩展单元的输入信号
2	ExtOp[1:0]	I	扩展方式的选择信号 00 ： 进行符号扩展 01 ： 进行零扩展 10 ： 进行低位零扩展
3	Out[31:0]	O	扩展单元的输出信号

模块功能定义如下：

表 12　ext 功能定义

序号	功能名称	功能描述
1	符号扩展	输出信号的低16位与输入信号相同， 高16位为输入信号的符号位
2	零扩展	输出信号的低16位与输入信号相同，高16位为0
3	低位零扩展	输出信号的高16位与输入信号相同，低16为为0

二、控制器设计

控制器端口说明如下：

表 13　控制器端口说明

序号	信号名	方向	描述
1	Op[5:0]	I	当前指令的Op字段（高6位）
2	Funct[5:0]	I	当前指令的Funct字段（低6位）
3	ALU_Zero	I	表示Alu运算结果是否为零的信号
4	nPC_Op[1:0]	O	选择PC的下一个值的信号
5	RegWrite	O	寄存器写使能信号
6	RegDst[1:0]	O	寄存器写入地址选择信号
7	RegSrc[1:0]	O	寄存器写入数据选择信号
8	ExtOp[1:0]	O	扩展单元功能选择信号
9	ALUOp[1:0]	O	ALU功能选择信号
10	ALUSrc	O	ALU的运算数据选择信号
11	MemWrite	O	数据存储器写使能信号

控制信号真值表如下：

表 14　控制信号真值表

	addu	subu	jr	ori	lw	sw	beq	lui	jal
Op	000000	000000	000000	001101	100011	101011	000100	001111	000011
Funct	100001	100011	001000						
nPC_Op[1]	0	0	1	0	0	0	0	0	1
nPC_Op[0]	0	0	1	0	0	0	Zero	0	0
RegWrite	1	1	0	1	1	0	0	1	1
RegDst[1]	0	0	x	0	0	x	x	0	1
RegDst[0]	1	1	x	0	0	x	x	0	0
RegSrc[1]	0	0	x	0	0	x	x	0	1
RegSrc[0]	0	0	x	0	1	x	x	0	0
ExtOp[1]	x	x	x	0	0	0	x	1	x
ExtOp[0]	x	x	x	1	0	0	x	0	x
ALUOp[1]	0	0	x	1	0	0	0	1	x
ALUOp[0]	0	1	x	0	0	0	0	0	x
ALUSrc	0	0	x	1	1	1	0	1	x
MemWrite	0	0	0	p	0	1	0	0	0

控制信号意义如下：

表 15　控制信号意义

序号	控制信号	意义
1	nPC_Op[1:0]	控制分支的信号，分支指令需要将该信号置为1
2	RegWrite	寄存器写使能信号，但需要些寄存器时将此信号置为1
3	RegDst[1:0]	选择寄存器的写入地址， 当此信号为00时，选择指令的rt字段（[20:16])为寄存器的写入地址； 当此信号为01时，选择指令的rd字段（[15:11]）为寄存器的写入地址； 当此信号为10时，选择0x1f为寄存器的写入地址
4	RegSrc[1:0]	选择寄存器的写入数据， 当此信号为00时，选择ALU的计算结果作为寄存器堆的写入值； 当此信号为01时，选择从数据存储器中取出的信号作为寄存器堆的写入值； 当此信号为10时，选择PC+4作为寄存器堆的写入值
5	ExtOp[1:0]	Ext功能选择信号，根据指令需要进行的扩展类型来设置为相应的值
6	ALUOp[2:0]	ALU功能选择信号，根据指令需要执行的运算种类来设置相应的值
7	ALUSrc	当此信号为0时，选择指令的rt字段（[20:16])为地址的寄存器中的数据作为ALU的第二个运算数；　当此信号为1时，选择经扩展后的立即数作为ALU的第二个运算数。
8	MemWrite	数据存储器写使能信号，当需要写数据存储器时将此信号置为1

三、测试程序

测试程序源代码如下：

```
   lui $s0, 0x2333
   ori $s1, $s0, 0x6666
   addu $s2, $s1, $s0
   subu $s3, $s1, $s0
   lui $0, 0x7777
   ori $0, $0, 0x5555
   sw $s2, 0($0)
   sw $s3, 4($0)
   lw $s4, 4($0)
   addu $s5, $0, $0
   beq $s5, $0, next
   addu $s5, $0, $s1
next:
   addu $s6, $0, $s1
   lui $s7, 0
   jal next2
   jal end
   lui $s7, 0x2222
next2:
   ori $s7, 0x3333
   jr $ra
end:
   nop
beq $0, $0, end   nop
```

期望运行结果：

寄存器 s0 - s7 的值分别为：

0x23330000、 0x23336666、 0x46666666、 0x00006666、

0x00006666、 0x00000000、 0x23336666、 0x00002333

数据存储器中地址 0x0、0x4 中的值分别为 0x46666666、 0x00006666

思考题

1、addr 信号截取了 32 信号的 11-2 位，用[11:2]比[9:0]更加清楚直观。
addr 信号的来源为 ALU 的运算结果。

2、清零信号针对 PC、寄存器堆、数据存储器进行清零复位操作。
这些部件保存了当前的状态信息，因而需要进行清零。

3、 coding 方式

（1） 利用 case 完成操作码和控制信号之间的对应

```
reg [12:0] Output_Bus;

assign nPC_Op = Output_Bus[12:11];
assign RegWrite = Output_Bus[10];
assign RegDst = Output_Bus[9:8];
assign RegSrc = Output_Bus[7:6];
assign EXTOp = Output_Bus[5:4];
assign ALUOp = Output_Bus[3:2];
assign ALUSrc = Output_Bus[1];
assign MemWrite = Output_Bus[0];

always @ (Op or Funct or ALU_Zero) begin
    case (Op)
        6'b000000 :
            case (Funct)
                6'b100001 : Output_Bus = 13'b 00_1_01_00_xx_00_0_0;
                6'b100011 : Output_Bus = 13'b 00_1_01_00_xx_01_0_0;
                6'b001000 : Output_Bus = 13'b 11_0_xx_xx_xx_xx_x_0;
                default :   Output_Bus = 13'b 00_0_xx_xx_xx_xx_x_0;
            endcase
        6'b001101 :         Output_Bus = 13'b 00_1_00_00_01_10_1_0;
        6'b100011 :         Output_Bus = 13'b 00_1_00_01_00_00_1_0;
        6'b101011 :         Output_Bus = 13'b 00_0_xx_xx_00_00_1_1;
       6'b000100 :    Output_Bus = {1'b0, ALU_Zero, 11'b0_xx_xx_xx_00_0_0};
        6'b001111 :         Output_Bus = 13'b 00_1_00_00_10_10_1_0;
        6'b000011 :         Output_Bus = 13'b 10_1_10_10_xx_xx_x_0;
        default :           Output_Bus = 13'b 00_0_xx_xx_xx_xx_x_0;
    endcase
end
```

（2） 利用 assign 语句完成操作码和控制信号的值之间的对应

```
wire r, addu, subu, jr, ori, lw, sw, beq, lui, jal;

 assign r = &(~Op);
 assign addu = r & Op[5] & ~Op[4] & ~Op[3] & ~Op[2] & ~Op[1] & Op[0];
 assign subu = r & Op[5] & ~Op[4] & ~Op[3] & ~Op[2] & Op[1] & Op[0];
 assign jr = r & ~Op[5] & ~Op[4] & Op[3] & ~Op[2] & ~Op[1] & ~Op[0];

 assign ori = ~Funct[5] & ~Funct[4] & Funct[3] & Funct[2] & ~Funct[1] &
Funct[0];
 assign lw = Funct[5] & ~Funct[4] & ~Funct[3] & ~Funct[2] & Funct[1] &
Funct[0];
 assign sw = Funct[5] & ~Funct[4] & Funct[3] & ~Funct[2] & Funct[1] &
Funct[0];
 assign beq = ~Funct[5] & ~Funct[4] & ~Funct[3] & Funct[2] & ~Funct[1] &
~Funct[0];
 assign lui = ~Funct[5] & ~Funct[4] & Funct[3] & Funct[2] & Funct[1] &
Funct[0];
 assign jal = ~Funct[5] & ~Funct[4] & ~Funct[3] & ~Funct[2] & Funct[1] &
Funct[0];

 assign nPC_Op = {jr & lui, jr & ALU_Zero};
 assign RegWrite = addu & subu & ori & lw & lui & jal;
 assign RegDst = {jal, addu & subu};
 assign RegSrc = {jal, lw};
 assign EXTOp = {lui, ori};
 assign ALUOp = {ori & lui, subu};
 assign ALUSrc = ori & lw & sw & beq;
 assign MemWrite = sw;
```

（3） 利用宏定义

```
`define ADDU_FUNCT 6'b100001
`define SUBU_FUNCT 6'b100011
`define JR_FUNCT 6'b001000

`define R_OP 6'b000000

`define ORI_OP 6'b001101
`define LW_OP 6'b100011
`define SW_OP 6'b101011
`define BEQ_OP 6'b000100
`define LUI_OP 6'b001111
`define JAL_OP 6'b000011
```

```
reg [12:0] Output_Bus;

assign nPC_Op = Output_Bus[12:11];
assign RegWrite = Output_Bus[10];
assign RegDst = Output_Bus[9:8];
assign RegSrc = Output_Bus[7:6];
assign EXTOp = Output_Bus[5:4];
assign ALUOp = Output_Bus[3:2];
assign ALUSrc = Output_Bus[1];
assign MemWrite = Output_Bus[0];

always @ (Op or Funct or ALU_Zero) begin
    case (Op)
        `R_OP :
            case (Funct)
                `ADDU_FUNCT : Output_Bus = 13'b 00_1_01_00_xx_00_0_0;
                `SUBU_FUNCT : Output_Bus = 13'b 00_1_01_00_xx_01_0_0;
                `JR_FUNCT :   Output_Bus = 13'b 11_0_xx_xx_xx_xx_x_0;
                default :     Output_Bus = 13'b 00_0_xx_xx_xx_xx_x_0;
            endcase
        `ORI_OP :           Output_Bus = 13'b 00_1_00_00_01_10_1_0;
        `LW_OP :            Output_Bus = 13'b 00_1_00_01_00_00_1_0;
        `SW_OP :            Output_Bus = 13'b 00_0_xx_xx_00_00_1_1;
       `BEQ_OP :     Output_Bus = {1'b0, ALU_Zero, 11'b0_xx_xx_xx_00_0_0};
        `LUI_OP :           Output_Bus = 13'b 00_1_00_00_10_10_1_0;
        `JAL_OP :           Output_Bus = 13'b 10_1_10_10_xx_xx_x_0;
        default :           Output_Bus = 13'b 00_0_xx_xx_xx_xx_x_0;
    endcase
end
```

4、各种方法的优缺点

（1） 利用 case 语句完成操作码和控制信号之间的对应

优点：编码方式简单易行

（2） 利用 assign 语句完成操作码和控制信号的值之间的对应

优点：可以看到控制器电路的门级结构

缺点：编码方式不够清晰易懂

（3） 利用宏定义：

优点：可以使语句更加清晰易懂

5、 add 和 addi 指令会检测运算是否发生了溢出，而 addu 和 addiu 指令不检测。因而在忽略溢出的前提下，addi 与 addiu 是等价的，add 与 addu 是等价的。

6、 单周期处理器优点：设计简单易行

单周期处理器缺点：

（1）需要足够长的周期来完成最慢的指令，因而效率太低。

（2）难以支持包含浮点或更复杂指令的指令集。

7、jal、jr 指令主要用于函数调用，在函数调用过程中需要把一些变量以堆栈的形式存储到内存中。在函数调用中，用 jal 跳转后进行压栈，将栈释放后用 jr 指令跳回。