计算机组成原理实验报告

1. 数据通路设计
   1. IM

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| IM.v | IM(jr, jal, nPc\_Sel, zero, clk, reset, imm, imm26, jr31, zhiring, PC);  Input jr; // 是不是jr指令  Input jal; // 是不是jal指令  Input nPc\_Sel; // 是不是Beq指令  Input zero; // rs和rt相等的话1  Input reset; // reset  Input [31:0] imm; // 32位的立即数（用beq指令）  Input [25:0] imm26; // 26位的立即数（用jal指令）  Input [31:0] jr31; // 31号的寄存器  Output [31:0] zhiling; //指令  Output [31:0] PC; // PC |

* 1. GRF

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| GRF.v | GRF(jal, clk, reset, RegWrite, ExtOp, ALUSrc, AL\_adress, A2\_adress, WD\_adress, WD, PC, imm16, A1, A2, imm32, MemData, jr31);  Input jal; // 是不是jal指令  Input clk; // clock  Input reset; // reset  Input RegWrite; //寄存器写信号  Input [1:0] ExtOp; // 控制 EXT  Input ALUSrc; // 控制ALU的第2个输入  Input [4:0] A1\_adress; // rs的地址  Input [4:0] A2\_adress; // rt的地址  Input [4:0] WD\_adress; // rd的地址  Input [31:0] WD; // 写的数据  Input [31:0] PC; // PC  Input [15:0] imm15; //16位的立即数（0到15）  Output [31:0] A1; // A1  Output [31:0] A2; // A2  Output [31:0] imm32; //16位的立即数扩展到32位  Output [31:0] MemData; // 写DM的数据  Output [31:0] jr31; // 31号的寄存器的数 |

* 1. ALU

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| ALU.v | ALU(clk, A1, A2, AluOp, A3, zero, MemAddr);  Input clk; // clock  Input [31:0] A1; // A1  Input [31:0] A2; // A2  Input [1:0] AluOp; // 控制ALU  Output [31:0] A3; // ALU后的数  Output zero; // A1=A2的话1  Ouput [11:2] MemAddr; // DM写的地址 |

* 1. DM

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| DM.v | DM(clk, reset, MemWrite, MemAddr, MemData, PC, dout);  Input clk; // clock  Input reset; // reset  Input MemWrite; // DM写信号  Input [11:2] MemAddr; // 写的地址  Input [31:0] MemData; // write data  Input [31:0] PC; // PC  Output [31:0] dout; // read data |

* 1. MUX

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| MUX.v | MUX(mux\_op, A1, A2, A3);  Input mux\_op; // 控制信号1的话A3=A2,else A3=A1;  Input [31:0] A1; // A1  Input [31:0] A2; // A2  Output [31:0] A3; // A3 |

* 1. EXT

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| EXT.v | IM(ExtOp, imm, A2);  Input [1:0] ExtOp; // 控制信号  Input [15:0] imm; // 16位的立即数  Output [31:0] A2; // Ext后的32位立即数 |

* 1. RegAddr

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| RegAddr.v | RegAddr(zhiling, RegDst, RegDst\_1,WD\_adress);  Input [31:0] zhiling; // 指令  Input RegDst; // ADDU，SUBU的话1  Input RegDst\_1; // LW, LUI, ORI的话1  Output [4:0] WD\_adress; //写寄存器号 |

1. 控制器设计
   1. CTRL

|  |  |
| --- | --- |
| 文件 | 模块接口定义 |
| CTRL.v | CTRL(clk, op, func, RegDst, ALUSrc, MemtoReg, RegWrite, nPC\_Sel, ExtOp, AluOp, MemWrite, RegDst\_1, jal, jr);  Input clk; // 指令  Input [5:0] op; // opcode(26~31)  Input [5:0] func; // func (0~5）  Ouput RegDst; // RegAddr 控制信号  Output ALUSrc; // 控制ALU的第2的数  Output MemtoReg; // 控制信号是不是SW  Output RegWrite; // GRF写信好  Output nPC\_Sel; // 是不是Beq指令  Ouput [1:0] ExtOp; // EXT 控制信号  Ouput [1:0] AluOp; // ALU 控制信号  Output MemWrite; // DM写信号  Output RegDst\_1; // RegAddr 控制信号  Output jal; // 是不是jal指令  Output jr; // 是不是jr指令 |

* 1. ADDU=~op[5]&~op[4]&~op[3]&~op[2]&~op[1]&~op[0]&func[5]&~func[ 4]&~func[3]&~func[2]&~func[1]&func[0];

SUBU=~op[5]&~op[4]&~op[3]&~op[2]&~op[1]&~op[0]&func[5]&~func[4] &~func[3]&~func[2]&func[1]&func[0];

LW=op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];

SW=op[5]&~op[4]&op[3]&~op[2]&op[1]&op[0];

BEQ=~op[5]&~op[4]&~op[3]&op[2]&~op[1]&~op[0];

ORI=~op[5]&~op[4]&op[3]&op[2]&~op[1]&op[0];

LUI=~op[5]&~op[4]&op[3]&op[2]&op[1]&op[0];

JAL=~op[5]&~op[4]&~op[3]&~op[2]&op[1]&op[0];

JR=~op[5]&~op[4]&~op[3]&~op[2]&~op[1]&~op[0]&~func[5]&~func[4]& func[3]&~func[2]&~func[1]&~func[0];

NOP=~op[5]&~op[4]&~op[3]&~op[2]&~op[1]&~op[0]&~func[5]&~func[4] &~func[3]&~func[2]&~func[1]&~func[0];

RegDst\_1=ORI|LUI|LW;

RegDst=ADDU|SUBU;

ALUSrc=ORI|SW|LUI|LW;

MemtoReg=LW;

RegWrite= ADDU|SUBU|LUI|LW|ORI|JAL;

nPC\_Sel=BEQ;

ExtOp=extop;

AluOp=aluop;

MemWrite=SW;

jal=JAL;

jr=JR;

1. 测试程序

341c0000 3000 ori $28,$0,0

341d0000 3004 ori $29,$0,0

34013456 3008 ori $1,$0,0x3456

00210821 300c addu $1,$1,$1

8c010004 3010 lw $1,4($0)

ac010004 3014 sw $1,4($0)

10650003 3034 beq 000100 00011 00101 0000 000 0000 0011

10000011 303c beq 000100 00000 00000 0000 0000 0001 0001

10e3000e 3048 beq 000100 00111 00011 0000 0000 0000 1111

0c000c22 3078 jar 0000 1100 0000 0000 0000 1100 0010 0010

建立testbench然后比较MARS的测试。

1. 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？**
      1. 答案：因为需要地址信号除以4，所以[11:2]的话不需要除以4。
   2. **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？**
      1. **答案： GRF和DM.因为GRF和DM有寄存器堆。表明有存储空间。**
   3. **列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例。**
      1. 答案：第一是直接写表达式

(比如SW=op[5]&~op[4]&op[3]&~op[2]&op[1]&op[0];）

第二是用if-else 句子

（比如if(ADDU|LW)begin

aluop=0;

end

else if(SUBU)begin

aluop=1;

end）

第三是用assign语法

（比如assign RegDst=ADDU|SUBU;)

* 1. **根据你所列举的编码方式，说明他们的优缺点**
     1. 答案：直接写表达式的话，代码简单然后可以一起用 always句子，但是可读性不好。

If-else的话。代码的长度一点长，但是可读性很 好。

Assign的话。代码简单。但是不能一起用 always句子**。**

* 1. **C语言是一种弱类型程序设计语言。C语言中不对计算结果溢出进行处理，这意味着C语言要求程序员必须很清楚计算结果是否会导致溢出。因此，如果仅仅支持C语言，MIPS指令的所有计算指令均可以忽略溢出。 请说明为什么在忽略溢出的前提下，addi与addiu是等价的，add与addu是等价的。提示：阅读《MIPS32® Architecture For Programmers Volume II: The MIPS32® Instruction Set》中相关指令的Operation部分**
     1. 答案：因为addi和addiu区别是

temp=(GPR[rs]31||GPR[rs])+sign\_extend(immediate) if temp32≠temp31 then SignalException(IntegerOverflow)

else

GPR[rt] ← temp31..0

endif

所以没有溢出。GPR[rt] ←GPR[rs] + sign\_extend(immediate).一定等价的。

所以add和addu也是同理。没有溢出的话等价的。

* 1. **根据自己的设计说明单周期处理器的优缺点**

答案：我觉得单周期CPU都是相同的时间去执行，这造成了时间上的浪费。但是优点是我觉得复杂度低了，建立单周期的时候不太难。

* 1. **简要说明jal、jr和堆栈的关系**。
     1. 答案：Jal指令表示直接PC的真改变，然后31号的寄存器存改变前真+4的真.jr 指令是PC的真改变寄存器的真，所以用jr $31的话PC=Jal下面的指令的地址.