计算机组成原理实验报告（Verilog单周期）

1. 设计通路设计
2. PC

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 处理器时钟信号 |
| reset | I | 复位信号 |
| next\_PC [31:0] | I | 下一个PC的值 |
| PC [31:0] | O | 输出当前指令的地址 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取出指令地址 | 取出将要执行的指令地址 |

1. PC\_calculator

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PC [31:0] | I | 当前PC值 |
| PCOp [1:0] | I | 下条PC输出控制信号 |
| equal | I | 寄存器值是否相等 |
| instr\_index [25:0] | I | 指令的后26位 |
| rs [31:0] | I | 寄存器储存的值 |
| imm [31:0] | I | 扩展后的立即数 |
| PC\_plus\_four [31:0] | O | PC+4 |
| next\_PC [31:0] | O | 下条PC的值 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计算下一条PC | 计算下一条指令的地址 |

1. im

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| addr [9:0] | I | 当下PC的值 |
| Instr [31:0] | O | 取出的32位指令 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 取出指令 | 取出将要执行的指令 |

1. grf

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 处理器时钟信号 |
| reset | I | 清零信号 |
| RA1[4:0] | I | 读寄存器时第一个寄存器的编号（地址） |
| RA2[4:0] | I | 读寄存器时第二个寄存器的编号（地址） |
| WD[31:0] | I | 寄存器写入数据 |
| WA[4:0] | I | 写寄存器时的编号（地址） |
| RegWrite | I | 寄存器写使能信号 |
| WPC | I | 当前PC的值 |
| RD1[31:0] | O | 读寄存器时第一个寄存器的输出数据 |
| RD2[31:0] | O | 读寄存器时第二个寄存器的输出数据 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读寄存器 | RD1输出RA1所寻址的寄存器中的数据  RD2输出RA2所寻址的寄存器中的数据 |
| 2 | 写寄存器 | 当时钟上升沿到来且RegWrite信号有效时，WD被写入WA所寻址的寄存器 |

1. alu

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 参与ALU计算的第一个值 |
| B[31:0] | I | 参与ALU计算的第二个值 |
| ALUOp[2:0] | I | ALU功能的选择信号：  000：ALU进行加法运算  001：ALU进行减法运算  010：ALU进行或运算  011：ALU进行与运算  100：ALU进行逻辑移位  101：ALU进行算术移位 |
| C[31:0] | O | ALU的计算结果 |
| Equal | O | 判断两数是否相等：  0：A!=B  1：A=B |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 无符号加运算 | Result = A + B |
| 2 | 无符号减运算 | Result = A - B |
| 3 | 或运算 | Result = A | B |
| 4 | 与运算 | Result = A & B |
| 5 | 算术移位 | Result=A>>B; |
| 6 | 逻辑移位 | Result=$signed(A)>>>B |
| 7 | 判断是否相等 | If(A == B) Equal=1 |

1. dm

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 控制器时钟信号 |
| reset | I | 存储器复位 |
| MemWrite | I | 写使能信号 |
| Addr [9:0] | I | 读或写时对应存储器时地址 |
| WD[31:0] | I | 写存储器时写入数据 |
| PC[31:0] | I | 存储器写使能信号 |
| RD[31:0] | O | 读存储器时的输出 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读存储器 | RD输出存储器中地址Addr存储的数据 |
| 2 | 写存储器 | 当时钟上升沿到来并且MemWrite有效时，WD被写入存储器中地址Addr的位置 |

1. mux32\_4

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| In0 [31:0] | I | 第一个输入 |
| In1 [31:0] | I | 第二个输入 |
| In2 [31:0] | I | 第三个输入 |
| In3 [31:0] | I | 第四个输入 |
| Select [1:0] | I | 选择信号 |
| Out[31:0] | O | 输出结果 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择输出 | Select=0：输出In0  Select=1：输出In1  Select=2：输出In2  Select=3：输出In3 |

1. mux 5\_4

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| In0 [4:0] | I | 第一个输入 |
| In1 [4:0] | I | 第二个输入 |
| In2 [4:0] | I | 第三个输入 |
| In3 [4:0] | I | 第四个输入 |
| Select [1:0] | I | 选择信号 |
| Out[4:0] | O | 输出结果 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择输出 | Select=0：输出In0  Select=1：输出In1  Select=2：输出In2  Select=3：输出In3 |

1. mux 32\_2

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| In0 [31:0] | I | 第一个输入 |
| In1 [31:0] | I | 第二个输入 |
| Select | I | 选择信号 |
| Out[31:0] | O | 输出结果 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择输出 | Select=0：输出In0  Select=1：输出In1 |

1. ext

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| imm[15:0] | I | 需要被扩展的16位立即数 |
| EOp[1:0] | I | 扩展方式选择信号：  00：符号扩展到32位  01：高位0扩展到32位  10：将立即数加载到高位，低位补0  11：符号扩展之后左移两位 |
| ext\_imm | O | 相应扩展后的立即数 |

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 符号扩展 | 将imm进行符号扩展到32位 |
| 2 | 零扩展 | 将imm进行高位补0扩展到32位 |
| 3 | 加载到高位 | 将imm加载到高位，低位补0 |
| 4 | 符号扩展之后左移2位 | 将imm进行符号扩展之后左移2位 |

1. 控制器模块规格

首先给出各个控制信号对应指令的真值表：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | addu | subu | jr | nop |
| Opcode | I | 000000 | 000000 | 000000 | 000000 |
| Funct | I | 100001 | 100011 | 001000 | 000000 |
| RegDst | O | 01 | 01 | X | X |
| PCOp | O | 00 | 00 | 11 | 00 |
| ALUOp | O | 000 | 001 | X | X |
| EXTOp | O | X | X | X | X |
| MemWrite | O | 0 | 0 | 0 | 0 |
| RegWrite | O | 1 | 1 | 0 | 0 |
| ALUSrc | O | 0 | 0 | X | X |
| MemtoReg | O | 01 | 01 | X | X |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | jal | ori | beq | lui | lw | sw |
| Opcode | I | 000011 | 001101 | 000100 | 001111 | 100011 | 101011 |
| RegDst | O | 10 | 00 | 00 | 00 | 00 | 00 |
| PCOp | O | 10 | 00 | 01 | 00 | 00 | 00 |
| ALUOp | O | X | 011 | X | X | 000 | 000 |
| EXTOp | O | X | 01 | 00 | 10 | 00 | 00 |
| MemWrite | O | 0 | 0 | 0 | 0 | 0 | 1 |
| RegWrite | O | 1 | 1 | 0 | 1 | 1 | 0 |
| ALUSrc | O | X | 1 | 0 | X | 1 | 1 |
| MemtoReg | O | 11 | 01 | X | 10 | 00 | X |

1. 测试程序与结果

lui $0,100

lui $1,100

lui $2,200

nop

lui $5,300

lui $6,400

lui $7,1000

lui $8,1000

nop

beq $7,$1,label #并不跳转

nop

sw $5,0($s6)

sw $7,4($s5)

lw $9,0($s6)

lw $10,4($s5)

lw $11,20

beq $7,$8,label #跳转

lui $9,100

lui $10,110

lui $12,120

label:

addu $9,$1,$2

addu $10,$2,$4

jal label2

nop

subu $9,$2,$3

subu $10,$3,$2

label2:

ori $9,$1,15

ori $10,$2,0

ori $11,$2,2

jr $ra

nop

机器码：

3c000064

3c010064

3c0200c8

00000000

3c05012c

3c060190

3c0703e8

3c0803e8

00000000

10e1000a

00000000

aec50000

aea70004

8ec90000

8eaa0004

8c0b0014

10e80003

3c090064

3c0a006e

3c0c0078

00224821

00445021

0c000c1a

00000000

00434823

00625023

3429000f

344a0000

344b0002

03e00008

00000000

1. 思考题

T2:

1. 存储器地址按字存储，为实现字对齐，默认最低两位为0。Addr应该是sw指令下，alu计算结果的11到2位，即ALUOut[11:2]。
2. Reset会清零PC，寄存器堆grf和数据存储器dm。清零是必须的，因为我们在使用时往往需要重启系统，这时必须清零才能继续操作，否则相当于是一次性的。

T4：

1. 三种方法代码如下：
2. Assign示例：

assign MemWrite=(opcode==101011)?1:0;

assign ALUSrc=(opcode==001101||opcode==100011||opcode==101011)?1:0;

1. case

module controller(

input [5:0] opcode,

input [5:0] funct,

output reg [1:0] RegDst,

output reg [1:0] PCOp,

output reg [2:0] ALUOp,

output reg [1:0] EXTOp,

output reg MemWrite,

output reg RegWrite,

output reg ALUSrc,

output reg [1:0] MemtoReg

);

always @(\*)

begin

if(opcode==6'b000000)

begin

case(funct)

6'b100001://addu

begin

RegDst=2'b01;

PCOp=2'b00;

ALUOp=3'b000;

MemWrite=0;

RegWrite=1;

ALUSrc=0;

MemtoReg=2'b01;

end

6'b100011://subu

begin

RegDst=2'b01;

PCOp=2'b00;

ALUOp=3'b001;

MemWrite=0;

RegWrite=1;

ALUSrc=0;

MemtoReg=2'b01;

end

6'b001000://jr

begin

PCOp=2'b11;

MemWrite=0;

RegWrite=0;

end

6'b000000://nop

begin

PCOp=2'b00;

MemWrite=0;

RegWrite=0;

end

endcase

end

else

begin

case(opcode)

6'b000100://beq

begin

RegDst=2'b00;

PCOp=2'b01;

EXTOp=2'b00;

MemWrite=0;

RegWrite=0;

ALUSrc=0;

end

6'b001101://ori

begin

RegDst=2'b00;

PCOp=2'b00;

ALUOp=3'b011;

EXTOp=2'b01;

MemWrite=0;

RegWrite=1;

ALUSrc=1;

MemtoReg=2'b01;

end

6'b100011://lw

begin

RegDst=2'b00;

PCOp=2'b00;

ALUOp=3'b000;

EXTOp=2'b00;

MemWrite=0;

RegWrite=1;

ALUSrc=1;

MemtoReg=2'b00;

end

6'b101011://sw

begin

RegDst=2'b00;

PCOp=2'b00;

ALUOp=3'b000;

EXTOp=2'b00;

MemWrite=1;

RegWrite=0;

ALUSrc=1;

end

6'b001111://lui

begin

RegDst=2'b00;

PCOp=2'b00;

EXTOp=2'b10;

MemWrite=0;

RegWrite=1;

MemtoReg=2'b10;

end

6'b000011://jal

begin

RegDst=2'b10;

PCOp=2'b10;

MemWrite=0;

RegWrite=1;

MemtoReg=2'b11;

end

endcase

end

end

endmodule

1. 宏定义

module a(

);

`define addu 6'b100001

`define subu 6'b100011

`define jr 6'b001000

`define nop 6'b000000

`define beq 6'b000100

`define lw 6'b100011

`define sw 6'b101011

`define jal 6'000011

`define lui 6'b001111

`define ori 6'b001101

always @(\*)

begin

if(opcode==6'b000000)

begin

case(funct)

addu:

begin

RegDst=2'b01;

PCOp=2'b00;

ALUOp=3'b000;

MemWrite=0;

RegWrite=1;

ALUSrc=0;

MemtoReg=2'b01;

end

subu:

begin

RegDst=2'b01;

PCOp=2'b00;

ALUOp=3'b001;

MemWrite=0;

RegWrite=1;

ALUSrc=0;

MemtoReg=2'b01;

end

jr:

begin

PCOp=2'b11;

MemWrite=0;

RegWrite=0;

end

nop:

begin

PCOp=2'b00;

MemWrite=0;

RegWrite=0;

end

endcase

end

else

begin

case(opcode)

beq:

begin

RegDst=2'b00;

PCOp=2'b01;

EXTOp=2'b00;

MemWrite=0;

RegWrite=0;

ALUSrc=0;

end

ori:

begin

RegDst=2'b00;

PCOp=2'b00;

ALUOp=3'b011;

EXTOp=2'b01;

MemWrite=0;

RegWrite=1;

ALUSrc=1;

MemtoReg=2'b01;

end

lw:

begin

RegDst=2'b00;

PCOp=2'b00;

ALUOp=3'b000;

EXTOp=2'b00;

MemWrite=0;

RegWrite=1;

ALUSrc=1;

MemtoReg=2'b00;

end

sw:

begin

RegDst=2'b00;

PCOp=2'b00;

ALUOp=3'b000;

EXTOp=2'b00;

MemWrite=1;

RegWrite=0;

ALUSrc=1;

end

lui:

begin

RegDst=2'b00;

PCOp=2'b00;

EXTOp=2'b10;

MemWrite=0;

RegWrite=1;

MemtoReg=2'b10;

end

jal:

begin

RegDst=2'b10;

PCOp=2'b10;

MemWrite=0;

RegWrite=1;

MemtoReg=2'b11;

end

endcase

end

end

endmodule

T4:

1. 主要原因在于add与addi本身执行的时候就会判断是否溢出，如果溢出则执行错误，因此在不溢出时和addu,addui相同。
2. 单周期处理器主要缺陷在于时间较长，每个时钟周期必须设定为最长的指令所花时间，尽管有些指令的周期明显短于该时间。此外单周期必须采用哈佛结构，即需要DM与IM分离，不像多周期可以共用一个。

单周期的优势在于相比于多周期与流水线不用添加各种流水寄存器与中间存储的寄存器，节约了空间成本。此外设计更加简洁，易懂，方便修改调整。

简言之，单周期相当于用时间换取空间。

（3） 递归的时候jal，jr需要用到堆栈。因为此时为保险起见必须把返回地址存入栈中，同样地jr也应从栈中取地址。一般而言，jal与jr将地址存取于寄存器，并不需要用到栈堆。