P4 单周期CPU设计

一、设计草稿

1.数据通路模块建模

（1）IFU（取指令单元）

信号	方向	描述
CLK	I	时钟信号
Reset	I	同步复位信号
imm16[15:0]	I	分支偏移量16位输入
BEQ	I	是否为分支指令控制
imm26[25:0]	I	跳转偏移量26位输入
JAL	I	是否为跳转指令控制
ra[31:0]	I	返回地址32位输入
JR	I	是否为返回指令控制
Zero	I	rt与rs是否相等控制
RD[31:0]	O	指令32位输出
PC+4[31:0]	O	返回地址32位输出
PC[31:0]	O	指令储存地址32位输出

（a）PC（程序计数器）

信号	方向	描述
CLK	I	时钟信号
Reset	I	同步复位信号
NPC[31:0]	I	下轮指令地址32位输入
PC[31:0]	O	本轮指令地址32位输出

（b）IM（指令储存器）

信号	方向	描述
A[31:0]	I	指令地址32位输入
RD[31:0]	O	指令32位输出

(c) NPC

信号	方向	描述
PC[31:0]	I	本轮指令地址32位输入
imm16[15:0]	I	分支偏移量16位输入
BEQ	I	是否为分支指令控制
imm26[25:0]	I	跳转偏移量26位输入
JAL	I	是否为跳转指令控制
ra[31:0]	I	返回地址32位输入
JR	I	是否为返回指令控制
Zero	I	rt与rs是否相等控制
NPC[32:0]	O	下轮指令地址32位输出

实现方法：若BEQ=1且Zero=1，则NPC=PC+4+sign_ext(imm16||00)，否则若JAL=1,则NPC=PC[31:28]||imm26||00，否则若JR=1，则NPC=ra，否则NPC=PC+4

(2) GRF（通用寄存器文件）

信号	方向	描述
CLK	I	时钟信号
Reset	I	同步复位信号
WE	I	写使能信号
A1[4:0]	I	读取数据一地址5位输入
A2[4:0]	I	读取数据二地址5位输入
A3[4:0]	I	写入数据地址5位输入
WD[31:0]	I	写入数据32位输入
PC[31:0]	I	指令储存地址32位输入
RD1[31:0]	O	读取数据一32位输出
RD2[31:0]	O	读取数据二32位输出

(3) ALU（算数逻辑单元）

信号	方向	描述
SA[31:0]	I	数据一32位输入
SB[31:0]	I	数据二32位输入
ALUOp[2:0]	I	计算控制信号
Zero	O	是否相等输出
ALUResult[31:0]	O	计算结果32位输出

（4）DM（数据储存器）

信号	方向	描述
CLK	I	时钟信号
A[31:0]	I	读取/写入地址32位输入
WD[31:0]	I	写入数据32位输入
WR	I	写使能信号
Reset	I	同步复位信号
PC[31:0]	I	指令储存地址32位输入
RD[31:0]	O	读取数据32位输出

（5）EXT（扩展单元）

信号	方向	描述
imm[15:0]	I	待扩展的立即数16位输入
EXTOp[1:0]	I	扩展方式控制信号
EXTResult[31:0]	O	扩展结果32位输出

2.数据通路组装与指令建模（工程化方法）

指令	IFU.imm16	IFU.imm26	IFU.ra	IFU.Zero	GRF.A1	GRF.A2	GRF.A3	GRF.WD	GRF.PC	ALU.SA	ALU.SB	DM.A	DM.WD	DM.PC	EXT.imm
add					IFU.RD[25:21]	IFU.RD[20:16]		ALU.ALUResult	IFU.PC	GRF.RD1	GRF.RD2				
sub					IFU.RD[25:21]	IFU.RD[20:16]		ALU.ALUResult	IFU.PC	GRF.RD1	GRF.RD2				
ori					IFU.RD[25:21]		IFU.RD[20:16]	ALU.ALUResult	IFU.PC	GRF.RD1	EXT.EXTResult				IFU.RD[15:0]
lw					IFU.RD[25:21]		IFU.RD[20:16]	DM.RD	IFU.PC	GRF.RD1	EXT.EXTResult	ALU.ALUResult			IFU.RD[15:0]
sw					IFU.RD[25:21]	IFU.RD[20:16]				GRF.RD1	EXT.EXTResult		GRF.RD2	IFU.PC	IFU.RD[15:0]
lui					IFU.RD[25:21]		IFU.RD[20:16]	ALU.ALUResult	IFU.PC	GRF.RD1	EXT.EXTResult				IFU.RD[15:0]
nop															
beq	IFU.RD[15:0]			ALU.Zero	IFU.RD[25:21]	IFU.RD[20:16]				GRF.RD1	GRF.RD2				
jal		IFU.RD[25:0]					0x1F	IFU.PC4	IFU.PC						
jr			RF.RD1		IFU.RD[25:21]										

3.控制模块建模

（1）指令真值表

指令	opcode	funct	BEQ	JAL	JR	WRSel	WDSel	EXTOp	RFWE	BSel	ALUOp	DMWR
add	000000	100000	0	0	0	01	00	xx	1	0	000	0
sub	000000	100010	0	0	0	01	00	xx	1	0	001	0
ori	001101	xxxxxx	0	0	0	00	00	00	1	1	010	0
lw	100011	xxxxxx	0	0	0	00	01	01	1	1	011	0
sw	101011	xxxxxx	0	0	0	xx	xx	01	0	1	100	1
lui	001111	xxxxxx	0	0	0	00	00	10	1	1	101	0
nop	000000	000000	0	0	0	xx	xx	xx	0	0	xxx	0
beq	000100	xxxxxx	1	0	0	xx	xx	xx	0	0	xxx	0
jal	000011	xxxxxx	0	1	0	10	10	xx	1	x	xxx	0
jr	000000	001000	0	0	1	xx	xx	xx	0	x	xxx	0

（2）CTRL（控制器）模块建模

信号	方向	描述
opcode[5:0]	I	指令opcode6位输入
funct[5:0]	I	指令funct6位输入
BEQ	O	是否为分支指令信号
JAL	O	是否为跳转指令信号
JR	O	是否为返回指令信号
WRSel[1:0]	O	GRF.A3选择信号
WDSel[1:0]	O	GRF.WD选择信号
EXTOp[1:0]	O	EXT扩展选择信号
RFWE	O	GRF写使能信号
BSel	O	ALU.B选择信号
ALUOp[2:0]	O	ALU运算控制信号
DMWR	O	DM写使能信号

二、思考题

1.阅读下面给出的 DM 的输入示例中（示例 DM 容量为 4KB，即 32bit × 1024 字），根据你的理解回答，这个 addr 信号又是从哪里来的？地址信号 addr 位数为什么是 [11:2] 而不是 [9:0]？

文件	模块接口定义

dm.v	dm(clk,reset,MemWrite,addr,din,dout);
	input clk; //clock
	input reset; //reset
	input MemWrite; //memory write enable
	input [11:2] addr; //memory's address for write
	input [31:0] din; //write data
	output [31:0] dout; //read data

addr信号来源：ALU运算结果（sw指令）。

addr信号位数为[11:2]原因：DM中储存信息是字对齐的，需要按字寻址。建模时是通过32位寄存器建模，因此只需要找到寄存器编号，即第几个字即可。

2.思考上述两种控制器设计的译码方式，给出代码示例，并尝试对比各方式的优劣。

指令对应的控制信号如何取值：

```
always @ (*) begin
    case (Instr)
        ADD : begin
            WRSel = 2'b01;
            WDSel = 2'b00;
            BSel = 1'b0;
            ALUOp = 3'b000;
            RFWE = 1'b1;
            DMWR = 1'b0;
        end
    endcase
end
```

优点：便于观察每条指令的行为，便于添加新的指令。

控制信号每种取值所对应的指令：

```
assign RFWE = (ADD || SUB || ORI || LW || LUI || JAL) ? 1'b1 : 1'b0;
```

优点：便于观察每个信号的状态，便于添加新的信号。

3.在相应的部件中，复位信号的设计都是同步复位，这与 P3 中的设计要求不同。请对比同步复位与异步复位这两种方式的 reset 信号与 clk 信号优先级的关系。

同步复位：clk优先级高于reset

```
always @ (posedge clk) begin
    if (reset) begin
        //...
    end else begin
        //...
    end
end
```

异步复位：reset优先级高于clk

```
always @ (posedge clk or posedge reset) begin
    if (reset) begin
        //...
    end else begin
        //...
    end
end
```

因为事实上nop指令对控制器中的控制信号控制输出为0，与不加入真值表控制信号输出相同，均不对GRF和DM进行改变。

4.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。

忽略溢出时：

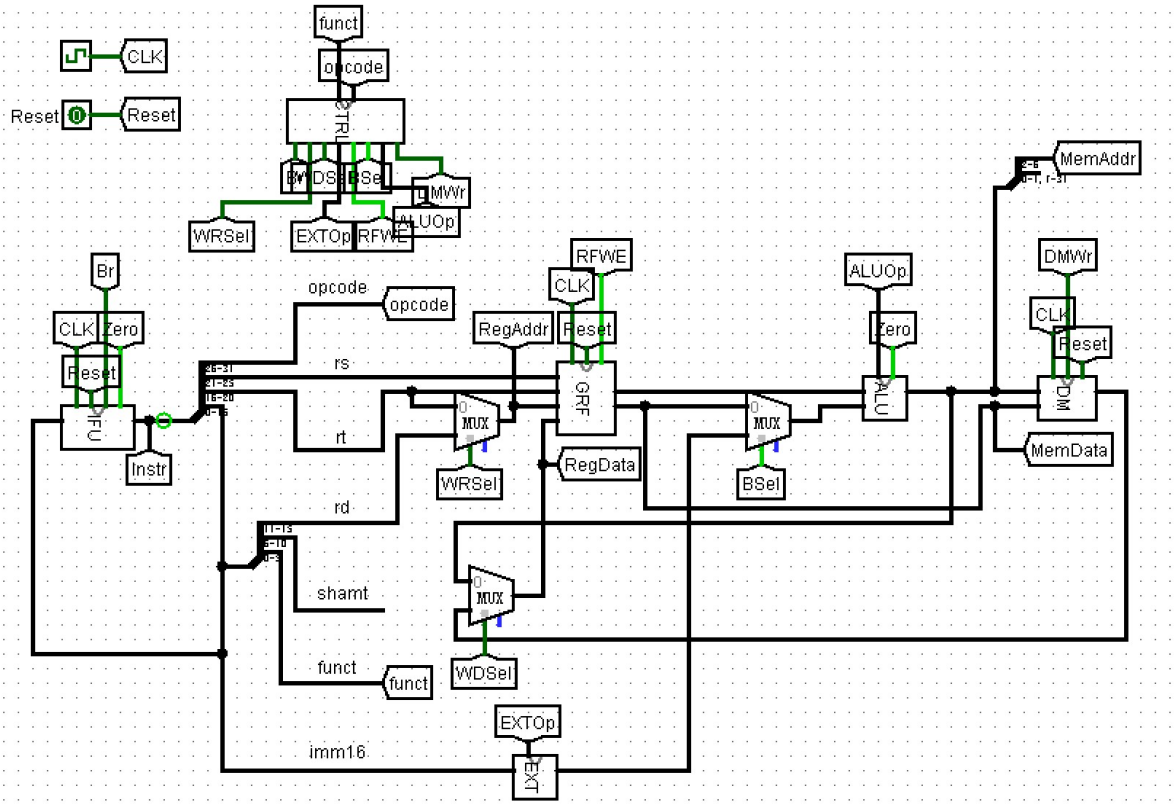
addi只执行else，即把temp[31:0]存入GPR[rt]，其中temp[31:0]=GPR[rs]+sign_extend(imm)，与addiu相同。

add只执行else，即把temp[31:0]存入GPR[rt]，其中temp[31:0]=GPR[rs]+GPR[rt]，与addu相同。

三、测试数据代码

```
.text
    ori $t1,$t1,4
    ori $t2,$t2,32
    ori $t3,$t3,8
A:  add $t1,$t1,$t1
    beq $t1,$t3,A
    jal B
    add $t1,$t1,$t1
    add $t1,$t1,$t1
    beq $t1,$t2,C
B:  sub $t1,$t1,$t3
    jr $ra
C:
```

附：P3电路图


funct
opcode
CTRL
CLK
Reset
Reset
RegWr
DSel
BSel
MWr
ALUOp
WRSel
EXTOp
RFWE
MemAddr
Br
CLK
Zero
Reset
IFU
Instr
opcode
opcode
rs
rt
rd
shamt
funct
funct
imm16
RFWE
CLK
RegAddr
Reset
GRF
RegData
MUX
WRSel
MUX
BSel
MUX
WDSel
EXTOp
EXT
ALUOp
Zero
ALU
DMWr
CLK
Reset
DM
MemData