P4设计文档

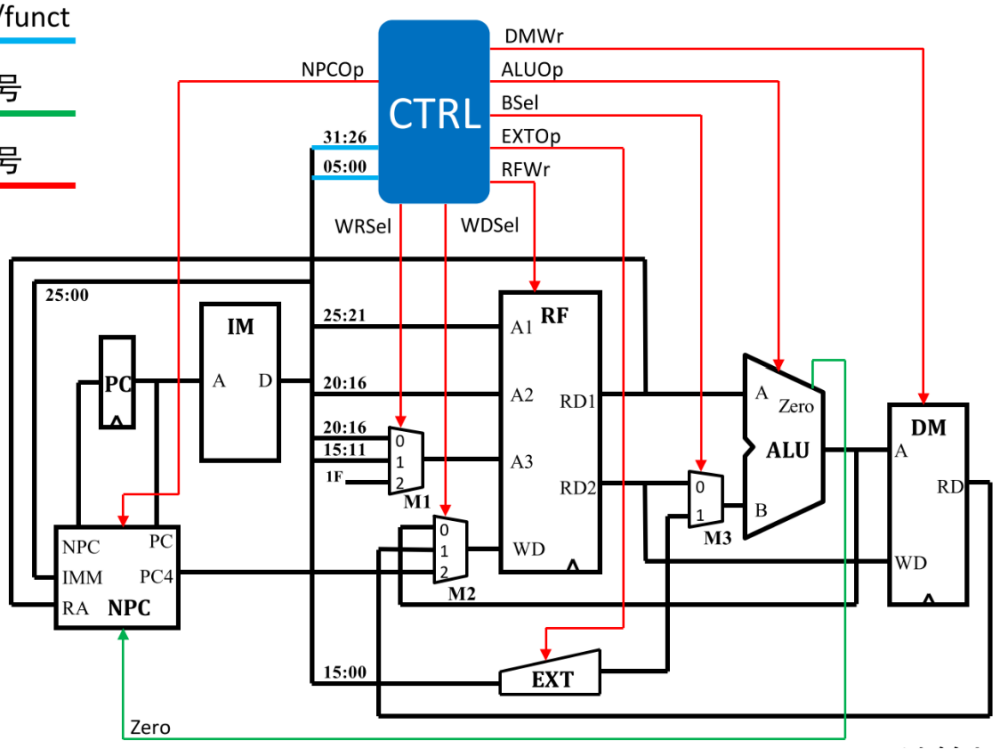
概述

本次通过Verilog设计单周期CPU架构支持了基本要求的指令add、sub、ori、lw、sw、beq、lui、nop、jal、jr指令，以此为基础又新增了lb、sb功能，即对DM进行了一些修改。首先给出基本框架图：

输入：opcode/funct

输入：状态信号

输出：控制信号



数据通路模块

IFU（取指令单元）

模块内部包含PC（程序计数器）以及IM（指令存储器）。

端口定义

信号名	方向	位宽	描述
clk	I	1	时钟信号
reset	I	1	同步复位信号
nextPC	I	32	下一条要被执行的指令的地址
PC	O	32	输出当前正在执行的指令的地址
inStr	O	32	输出当前正在执行的指令

功能定义

序号	功能名称	功能描述
1	复位	当时钟上升沿到来且reset信号有效时，将PC的值置为 0x00003000
2	更新PC	当 时钟上升沿来临时，将下一条指令的地址（nextPC）传递给PC

序号	功能名称	功能描述
3	取指令	根据当前PC的值从 IM（指令存储器）中读出对应的指令到inStr端口

GRF（通用寄存器组）

该模块内部包含 32 个具有写使能 32 位寄存器，分别对应 MIPS 架构中0~31通用寄存器（其中 0 号寄存器中的值恒为 0，即不具备写使能）。GRF 可以实现同步复位，同时可以根据输入的 5 位地址（0~31）向寄存器堆存取数据，实现定向访存寄存器。

端口定义

信号名	方向	位宽	描述
PC	I	32	用于输出指定信息
clk	I	1	时钟信号
reset	I	1	同步复位信号
A1	I	5	地址输入信号，指定 32 个寄存器中的一个，将其中的数据读出到 RD1
A2	I	5	地址输入信号，指定 32 个寄存器中的一个，将其中的数据读出到 RD2
A3	I	5	地址输入信号，指定 32 个寄存器中的一个，将其作为写入目标
WD	I	32	数据输入信号
regWrite	I	1	写使能信号
RD1	O	32	输出 A1 指定的寄存器中的 32 位数据
RD2	O	32	输出 A2 指定的寄存器中的 32 位数据

功能定义

序号	功能名称	功能描述
1	复位	时钟上升沿到来且reset信号有效时，所有寄存器中储存的值均被清零
2	读数据	读出 A1、A2 地址对应的寄存器中储存的数据，将其加载到 RD1 和 RD2
3	写数据	当 WE 信号有效且时钟上升沿来临时，将 WD 中的数据写入到 A3 地址对应的寄存器

NPC

负责计算下一条指令的地址并传递给PC。

信号名	方向	位宽	描述
PC	I	32	当前指令地址

信号名	方向	位宽	描述
raGPR	I	32	用于jr指令，传入地址
imm	I	25	立即数来源
NPCOp	I	2	跳转选择信号
equal	I	1	比较是否相等
nextPC	O	32	输出下一指令地址
PC4	O	32	传递PC + 4

ALU

该模块主要实现了加法、减法、按位或、立即数加载至高位（LUI）运算。

端口定义

信号名	方向	位宽	描述
aluOp	I	3	ALU 功能选择信号
Src1	I	32	参与 ALU 计算的第一个值
Src2	I	32	参与 ALU 计算的第二个值
isEqual	O	1	相等判断信号
result	O	32	输出 ALU 计算结果

功能定义

序号	功能名称	ALU_Op	功能描述
1	减法	0'b000	result = Src1 - Src2
2	加法	0'b001	result = Src1 + Src2
3	按位或	0'b010	result = Src1 | Src2
4	加载至高位	0'b011	result = Src2 << 16

DM（数据存储器）

该模块主要通过RAM实现，具有读写功能以及同步复位功能。

端口定义

信号名	方向	位宽	描述
PC	I	32	用于输出指定信息
clk	I	1	时钟信号
reset	I	1	同步复位信号

信号名	方向	位宽	描述
Addr	I	5	地址输入信号，指向数据储存器中某个存储单元
Data	I	32	数据输入信号
Memwrite	I	1	写使能信号
DataOut	O	32	输出 Addr 指定的存储单元中的 32 位数据

Controller（控制模块）

在控制模块中，我们对指令中 Opcode 域和 Funct 域中的数据进行解码，输出 ALUOp,MemtoReg 等8条控制指令，从而对数据通路进行调整，满足不同指令的需求。为实现该模块，我们又在内部设计了两部分 —— 和逻辑（AND Logic）和或逻辑（OR Logic）。前者的功能是识别，将输入的 Opcode 和 Funct 数据识别为对应的指令，后者的功能是生成，根据输入指令的不同产生不同的控制信号。

控制信号

序号	信号名	位宽	描述	触发指令（信号为 1）
1	memToReg	1	GRF 中 WD 接口输入数据选择	lw、lb、jal
2	memWrite	1	DM 写入使能信号	sw、sb
3	aluSrc	1	ALU 中 Src2 接口输入数据选择	ori、lw、sw、lui、lb、sb
4	regWrite	1	GRF 写入使能信号	add、sub、ori、lw、lui、lb、jal
5	extOp	1	立即数符号扩展选择	lw、sw、beq、lb、sb
6	regDst	1	GRF 中 A3 接口输入数据选择	add、sub、jal
7	NPCOp	1	NPC计算类型判断信号	beq、jal、jr
8	aluOp	3	ALU 功能选择信号	略

测试方案

基于Pre教程中的测试样例：

```
ori $a0, $0, 123
ori $a1, $a0, 456
lui $a2, 123            # 符号位为 0
lui $a3, 0xffff         # 符号位为 1
ori $a3, $a3, 0xffff    # $a3 = -1
add $s0, $a0, $a2      # 正正
add $s1, $a0, $a3      # 正负
add $s2, $a3, $a3      # 负负
ori $t0, $0, 0x0000
```

```
sw $a0, 0($t0)
sw $a1, 4($t0)
sw $a2, 8($t0)
sw $a3, 12($t0)
sw $s0, 16($t0)
sw $s1, 20($t0)
sw $s2, 24($t0)
lw $a0, 0($t0)
lw $a1, 12($t0)
sw $a0, 28($t0)
sw $a1, 32($t0)
ori $a0, $0, 1
ori $a1, $0, 2
ori $a2, $0, 1
beq $a0, $a1, loop1     # 不相等
beq $a0, $a2, loop2     # 相等
loop1:sw $a0, 36($t0)
loop2:sw $a1, 40($t0)
```

进行了一些补充：

```
ori $a0, $0, 65535
ori $a1, $0, 1234
ori $a1, $a0, 0
lui $t1, 0xffff
ori $t1, $0, 0xffff
ori $t0, 1
add $t2, $t1, $t0
add $t3, $t0, $0
lui $0, 1111
```

并且测试了p4教程中所给出的样例。

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

答：Addr由ALU的计算结果得出，由于1个字占4个字节，所以Addr后两位恒为0，为了节省空间从而取[11:2]。

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。

指令对应的信号如何取值，如：

```
case(instruction_type)
        0: begin
            MemWrite <= 0;
            ctrl_skip_type <= 0;
            RegWrite <= 0;
        end
        1: begin
            MemtoReg <= 1;
            MemWrite <= 0;
            ctrl_skip_type <= 0;
            ALUCtrl <= 1;
            ALUSrc <= 0;
            RegDst <= 1;
            RegWrite <= 1;
            issigned_extend <= 0;
        end
        // ...
```

控制信号每种取值所对应的指令，如：

```
assign memToReg[0] = lw | lb;
assign memToReg[1] = jal;
assign memWrite = sw | sb;
assign regDst[0] = add | sub;
assign regDst[1] = jal;
assign aluOp[0] = add | lui | lw | sw | lb | sb;
assign aluOp[1] = ori | lui;
// ...
```

前者的优势是在设计时思路较为清晰，后者的优势是代码便于理解，修改或调试时更加方便。

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。

答：同步复位时clk信号优先级高于reset信号，只有clk上升沿到来时reseti信号有效才会复位；异步复位时reset信号优先级高于clk信号，无论clk信号为何值，只要reset信号有效就会复位。

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。

答：在忽略溢出的前提下，add与addi均不需要考虑对于溢出的判断，且保存结果的后32位，从而与addu、addiu等价。