计算机组成原理P3实验报告

彭杰奇15061169

1. 模块定义
2. IFU模块定义

（1）基本描述

IFU主要功能是完成取指令功能。IFU内部包括了PC、IM(指令存储器)以及其他相关逻辑。IFU除了能执行顺序取指令外，还能根据PCSrc反映的beq指令的执行情况来决定是顺序取指令还是转移取指令。

（2）模块接口

表1 IFU模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| PCSrc | I | 当前指令是否是beq指令  1:当前指令是beq指令  0:当前指令不是beq指令 |
| Imm[31:0] | I | 若为beq指令，需要进行移位计算的立即数 |
| Reset | I | 复位信号，1:有效，0:无效 |
| Clk | I | 时钟信号 |
| Instr[31:0] | O | 当前指令输出 |

（3）功能定义

表2 IFU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 复位信号有效时，PC设置为0x00000000 |
| 2 | 取指令 | 根据PC当前值从IM中取指令输出 |
| 3 | 计算下一条指令地址 | beq == 0 时，PC 🡨 PC + 4；  beq == 1 时，PC 🡨 PC + 4 + Imm||0^2 |

1. GRF模块定义

（1）基本描述

GRF模块为通用寄存器堆，主要由32个具有写使能端的32位寄存器组成，能够同时根据由R1和R2输入的地址从其中两个寄存器中读出数据，并根据WR中输入的地址向其中一个寄存器写入数据。

（2）模块接口

表3 GRF模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| R1[4:0] | I | rs寄存器的地址 |
| R2[4:0] | I | rt寄存器的地址 |
| WR[4:0] | I | 要写入的寄存器的地址 |
| WriteData[31:0] | I | 要写入的数据 |
| Clk | I | 时钟信号 |
| RegWrite | I | 写使能信号，1:有效，0:无效 |
| RD1[31:0] | O | rs寄存器的值 |
| RD2[31:0] | O | rt寄存器的值 |

（3）功能定义

表4 GRF功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | RD1🡨(R1)  RD2🡨(R2) |
| 2 | 写数据 | RegWrite有效时，(WR)🡨WriteData |

1. ALU模块定义

（1）基本描述

ALU为算数逻辑单元，可以对输入的两个数据进行加、减、按位与和按位或操作，并能够判断输入数据是否相等。

（2）模块接口

表5 ALU模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A1[31:0] | I | 第一个运算数 |
| A2[31:0] | I | 第二个运算数 |
| ALUCtr[1:0] | I | ALU控制信号  2’b00:加法运算  2’b01:减法运算  2’b10:按位与运算  2’b11:按位或运算 |
| Zero | O | 输入数据是否相等  1:相等  2:不相等 |
| ALUResult[31:0] | O | ALU运算结果 |

（3）功能定义

表6 ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加法运算 | ALUResult 🡨 A1+A2 |
| 2 | 减法运算 | ALUResult 🡨 A1-A2 |
| 3 | 按位与运算 | ALUResult 🡨 A1&A2 |
| 4 | 按位或运算 | ALUResult 🡨 A1|A2 |
| 5 | 等于判断 | Zero 🡨 (A1-A2)==0?1:0 |

1. DM模块定义

（1）基本描述

DM模块为数据存储器，由一个32bit \* 32字的RAM构成，起始地址为0x00000000用于存储数据。

（2）模块接口

表7 DM模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Addr[6:2] | I | 读/写DM的地址 |
| WriteData[31:0] | I | 要写入DM的数据 |
| MemWrite | I | 写DM的控制信号 |
| MemRead | I | 读DM的控制信号 |
| ReadData[31:0] | O | 从DM读出的数据 |

（3）功能定义

表8 DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读数据 | 当MemRead为1时，ReadData 🡨 RAM(Addr) |
| 2 | 写数据 | 当MemWrite为1时，RAM(Addr) 🡨 WriteData |

1. EXT模块定义

（1）基本描述

EXT模块的作用是将16位立即数扩展为32位。

（2）模块接口

表9 EXT模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Imm\_16[15:0] | I | 要扩展的16位立即数 |
| ExtOp[1:0] | I | 扩展方式选择信号  2’b00:符号扩展  2’b01:后接16位0  2’b10:无符号扩展 |
| Imm\_32[31:0] | O | 扩展后的32位立即数 |

（3）功能定义

表10 EXT功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 位数扩展 | ExtOp为2’b00时，16位立即数正常符号扩展为32位  ExtOp为2’b01时，16为立即数后接16位0扩展为32位  ExtOp为2’b10时，16为立即数无符号扩展为32位 |

1. 控制器设计
2. Controller模块定义

（1）基本描述

Controller模块为CPU控制器，可以根据输入指令的opcode和funct值输出各种控制信号。

（2）模块接口

表11 Controller模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| opcode[5:0] | I | 指令中的opcode,即[31:26]位 |
| Funct[5:0] | I | 指令中的funct,即[5:0]位 |
| RegDst | O | 寄存器写入端地址控制  1:选择rd字段  0:选择rt字段 |
| ALUSrc | O | ALU输入端A2选择  1:选择Imm\_32  0:选择RD2 |
| MemtoReg | O | 寄存器堆写入端WD选择  0:来自ALU输出  1:来自DM输出 |
| RegWrite | O | 写寄存器控制信号 |
| MemWrite | O | 写DM控制信号 |
| MemRead | O | 读DM控制信号 |
| ExtOp[1:0] | O | EXT扩展方式控制信号 |
| nPC\_Sel | O | 判断是否为beq指令 |
| ALUCtr[1:0] | O | ALU控制信号 |

（3）功能定义

表12 Controller功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | addu指令 | 当前指令为addu时，RegDst、RegWrite信号为1，其他全为0 |
| 2 | subu指令 | 当前指令为subu时，RegDst、RegWrite、ALUCtr[1]信号为1，其他全为0 |
| 3 | ori指令 | 当前指令为ori时，ALUSrc、RegWrite、ALUCtr[0]、ALUCtr[1]、ExtOp[1]信号为1，其他全为0 |
| 4 | lw指令 | 当前指令为lw时，ALUSrc、MemtoReg、RegWrite、MemRead信号为1，其他全为0 |
| 5 | sw指令 | 当前指令为sw时，ALUSrc、MemWrite信号为1，其他全为0 |
| 6 | beq指令 | 当前指令为beq时，nPC\_Sel、ALUCtr[1]信号为1，其他全为0 |
| 7 | lui指令 | 当前指令为lui时，ALUSrc、RegWrite、ExtOp[0]信号为1，其他全为0 |

1. Controller真值表

表13 Controller真值表

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instr | | addu | subu | ori | lw | sw | beq | lui |
| opcode | I | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 |
| funct | 100001 | 100011 | N/A | | | | |
| RegDst | O | 1 | 1 | 0 | 0 | X | X | 0 |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| MemtoReg | 0 | 0 | 0 | 1 | X | X | 0 |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 1 |
| nPC\_Sel | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| ExtOp[1] | X | X | 1 | 0 | 0 | 0 | 0 |
| ExtOp[0] | X | X | 0 | 0 | 0 | 0 | 1 |
| MemRead | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| ALUOp[1] | | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| ALUOp[0] | | 0 | 0 | 1 | 0 | 0 | 1 | 0 |
| ALUCtr[1] | O | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ALUCtr[0] | 0 | 1 | 1 | 0 | 0 | 1 | 0 |

1. ALU控制器设计

ALU控制器是控制器的一部分，它利用funct[5:0]的值和过渡信号ALUOp[1:0]得到控制ALU运算的ALU控制信号ALUCtr[1:0]

表14 ALU控制器真值表

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ALUOp | Funct | ALUCtr | 运算 | 描述 |
| 00 | X | 00 | 加法 | 针对lw、sw、lui指令 |
| 01 | X | 01 | 减法 | 针对beq指令 |
| 11 | X | 11 | 按位或 | 针对ori指令 |
| 10 | 100001(addu) | 00 | 加法 | 针对addu指令 |
| 10 | 100011(subu) | 01 | 减法 | 针对subu指令 |

1. 测试程序设计

ori $1,$0,20

lui $2,1

addu $0,$1,$1

subu $3,$2,$1

addu $4,$3,$0

beq $3,$2,target

target:

ori $5,$0,100

ori $6,$3,200

lui $6,0x0f0e

addu $7,$6,$5

sw $7,0($5)

sw $6,4($5)

subu $8,$7,$6

lw $9,($5)

lw $10,4($5)

预期结果

$at 0x00000014

$v0: 0x00010000

$v1: 0x0000ffec

$a0: 0x0000ffec

$a1: 0x00000064

$a2: 0x0f0e0000

$a3: 0x0f0e0064

$t0: 0x00000064

$t1: 0x0f0e0064

$t2: 0x0f0e0000

Mem[4]: 0x0f0e0064

Mem[8]: 0x0f0e0000

1. 思考题

1. 在上个学年的计组课程中，PC（程序计数器）位数被规定为30位，试分析其与32位PC的优劣。

PC（程序计数器）位数被规定为30位，和32位相比其实是省略了最后两位0，这样PC在计算下一条指令的时候，只需要PC+1或者PC+1+Imm，而且beq指令的立即数也无需左移2位，因而相对来说更加简单；而32位PC计算下一条指令需要PC+4，与MIPS实际是相符的，方便直接应用，但是计数器是地址4倍，计算下一条指令没那么方便。

2. 现在我们的模块中 IM使用ROM， DM使用RAM， GRF使用寄存器，这种做法合理吗？ 请给出分析，若有改进意见也请一并给出。

合理。

IM使用ROM，ROM是只读存储器，使用它存储指令，可以避免在运行过程中误操作造成指令更改；

DM使用RAM，RAM是随机存储器，可以写入也可以读出数据，满足DM的要求，而且有读写使能端，能够防止对数据的非法操作；

GRF使用寄存器堆，可以满足存储和读出指定地址的数据，且寄存器具有使能端，能够满足GRF的写使能。

3. 结合上文给出的样例真值表，给出RegDst， ALUSrc， MemtoReg，RegWrite, nPC\_Sel, ExtOp与op和func有关的布尔表达式（表达式中只能使用“与、或、非”3 种基本逻辑运算。）

RegDst=~op5&~op4&~op3&~op2&~op1&~op0

ALUSrc=~(~op5&~op4&~op3&~op2&~op1&~op0)&~(~op5&~op4&~op3&op2&~op1&~op0)

MemtoReg=op5&~op4&~op3&~op2&op1&op0

RegWrite=~(op5&~op4&op3&~op2&op1&op0)&~(~op5&~op4&~op3&op2&~op1&~op0)

MemWrite=op5&~op4&op3&~op2&op1&op0

nPC\_sel=~op5&~op4&~op3&op2&~op1&~op0

ExtOp=~(~op5&~op4&op3&op2&~op1&op0)

4. 充分利用真值表中的 X 可以将以上控制信号化简为最简单的表达式， 请给出化简后的形式。

RegDst=~op5&~op4&~op3&~op1&~op0 (110001)

ALUSrc=op5|op4|op3|op1|op0

MemtoReg=op5&~op4&~op2&op1&op0 (000110)

RegWrite=~(op5&~op4&op3&~op2&op1&op0)&~(~op5&~op4&~op3&op2&~op1&~op0)

MemWrite=op5&~op4&op3&~op2&op1&op0

nPC\_sel=~op5&~op4&~op3&op2&~op1&~op0

ExtOp=op5|op4|~op3|~op2|op1|~op0

5. 事实上，实现nop空指令，我们并不需要将它加入控制信号真值表，为什么？请给出你的理由。

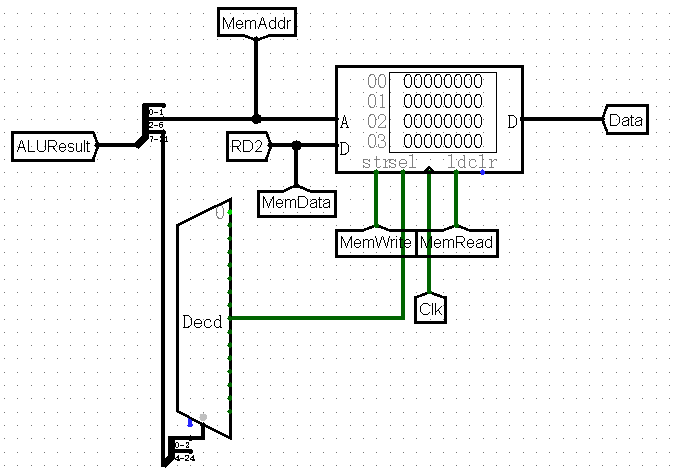
因为nop空指令机器码是0x00000000，它除了使PC+4之外不起到其他作用，在Controller的控制信号真值表中，他对所有输出端口都是X，X可以看做0或1任意一个，所以没有必要加入真值表中。

6. 前文提到，“可能需要手工修改指令码中的数据偏移”，但实际上只需再增加一个 DM片选信号,就可以解决这个问题。请阅读相关资料并设计一个 DM 改造方案使得无需手工修改数据偏移。

因为我们输入IM的指令是从ROM的0位地址开始的，这样在遇到J指令时可能出现问题，我们在MARS上写MIPS代码，如果把指令设在0x00000300开始，数据在0x00000000开始，DM没有问题，beq指令因为是相对寻址也没有问题，但是J指令是绝对寻址，就可能超出IM地址，所以我们要在MARS中设置指令在0x00000000开始，而数据段就要从0x00002000开始，我们原本用的DM是32位\*32字的，地址位从0x00000000开始，只到了0x00000020,而我们不需要进行位扩展，因此只需要使用现在的32\*32位的DM进行字扩展然后通过片选信号选择我们现在使用的这片，模拟是从0x0000200开始的地址就好

0x00002000转换成总共是2^16位，现在我们每个字单位是32位，所以总共是2^8个字单位，我们现在用的DM是32个也就是2^5个字单位，所以进行字扩展总共需要2^8/2^5=8个DM,又因为我们的地址是要从0x00002000往后延伸，所以我们需要更多的DM，所以就把地址延伸到0x00004000，这样是2^9个字单位，需要16个DM，然后我们要选取的就是第9个DM，那就把我们现在的DM安排到第9个，当片选信号选择第9个时就可以了。这就相当于把32\*32的存储器扩展为512\*32的存储器并选其中第9个DM

我们需要9位地址管脚，5位连接DM地址，4位产生16个片选信号，我们需要选择第9个，1000。



7. 除了编写程序进行测试外，还有一种验证CPU设计正确性的办法——形式验证。 形式验证的含义是根据某个或某些形式规范或属性，使用数学的方法证明其正确性或非正确性。请搜索“形式验证（Formal Verification)"了解相关内容后，简要阐述相比与测试，形式验证的优劣。

形式验证是对指定描述的所有可能的情况进行验证，而不是仅仅对其中的一个子集进行多次试验，因此有效地克服了模拟验证的不足,验证时间短，有利于尽早尽快地发现和改正电路错误，缩短设计周期；但是形式验证不能有效验证电路的性能，如电路的时延和功耗。