单周期CPU设计文档(P3)

17373252 丁禹衡

1. 模块规格
2. IFU（取指令单元）

端口说明：

表1 IFU端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号，将PC置为0x00003000  1：复位  0：无效 |
| nPC[31:0] | I | 下一条指令地址 |
| PC[31:0] | O | 当前指令地址 |
| Instr[31:0] | O | 输出PC指定的当前指令 |

功能定义：

表2 IFU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，PC被设置为0x00003000 |
| 2 | 取指令 | 当时钟上升沿到来时，IM根据PC输出的地址取出一条指令 |

1. GRF（通用寄存器文件）

端口说明：

表3 GRF端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号，将32个寄存器中的值全部清零  1：复位  0：无效 |
| WriteEn | I | 写使能信号  1：可向GRF中写入数据  0：不能向GRF中写入数据 |
| Addr1[4:0] | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到ReadData1 |
| Addr2[4:0] | I | 5位地址输入信号，指定32个寄存器中的一个，将其中存储的数据读出到ReadData2 |
| Addr3[4:0] | I | 5位地址输入信号，指定32个寄存器中的一个，作为写入的目标寄存器 |
| WriteData[31:0] | I | 32位数据输入信号 |
| ReadData1[31:0] | O | 输出Addr1指定的寄存器中的32位数据 |
| ReadData2[31:0] | O | 输出Addr2指定的寄存器中的32位数据 |

功能定义：

表4 GRF功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | reset信号有效时，所有寄存器存储的数值清零 |
| 2 | 读数据 | 读出Addr1,Addr2地址对应寄存器中所存储的数据到ReadData1,ReadData2 |
| 3 | 写数据 | 当WriteEn有效且时钟上升沿来临时，将WriteData写入Addr3所对应的寄存器中 |

1. ALU（算术逻辑单元）

端口说明：

表5 ALU端口说明

|  |  |  |
| --- | --- | --- |
| 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：ALUResult = A + B  001：ALUResult = A - B  010：ALUResult = A & B  011：ALUResult = A | B  100：未使用  101：未使用  110：未使用  111：未使用 |
| ALUResult[31:0] | O | ALU的计算结果 |
| Zero | O | 零标志位  1：ALUResult为0  0：ALUResult不为0 |

功能定义：

表6 ALU功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 加运算 | ALUResult = A + B |
| 2 | 减运算 | ALUResult = A - B |
| 3 | 与运算 | ALUResult = A & B |
| 4 | 或运算 | ALUResult = A | B |
| 5 | 判断运算结果是否为零 | Zero = (ALUResult == 0) ? 1 : 0 |

1. DM（数据存储器）

端口说明：

表7 DM端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号，将DM清零  1：复位  0：无效 |
| WriteEn | I | 写使能信号  1：可向DM中写入数据  0：不能向DM中写入数据 |
| Addr[4:0] | I | 5位地址输入信号，指定读出或写入数据的地址 |
| WriteData[31:0] | I | 32位数据输入信号 |
| ReadData[31:0] | O | 输出Addr指定的32位数据 |

功能定义：

表8 DM功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当复位信号有效时，将DM中数据清零 |
| 2 | 读数据 | 读出Addr所指定的数据到ReadData |
| 3 | 写数据 | 当WriteEn有效且时钟上升沿到来时，将输入数据WriteData写入Addr所指定的地址 |

5. Ext（位扩展单元）

端口说明：

表9 Ext端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Src[15:0] | I | 16位输入数据 |
| ExtOp | I | 位扩展方式选择信号  1：零扩展  0：符号扩展 |
| Dst[31:0] | O | 扩展后的32位输出数据 |

功能定义：

表10 Ext功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 零扩展 | 将16位输入数据进行零扩展，输出32位数据 |
| 2 | 符号扩展 | 将16位输入数据进行符号扩展，输出32位数据 |

二、控制器设计

端口说明：

表11 Ctrl端口说明

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Op[5:0] | I | 输入指令的Op字段 |
| Funct[5:0] | I | 输入指令的Funct字段 |
| nPCSrc[1:0] | O | IFU的nPC端口数据源选择信号  00：来源为PC + 4  01：来源为PC + 4或PC + 4 + sign\_extend(offset || 00)  10：来源为PC[31:28] || instr\_index || 00  11：来源为GRF的ReadData1端口输出 |
| RegSrc[1:0] | O | GRF的WriteData端口数据源选择信号  00：来源为ALU的计算结果  01：来源为DM的输出数据  10：来源为加载至高位的立即数  11：来源为PC + 4 |
| MemWrite | O | DM写使能信号  1：可向DM中写入数据  0：不能向DM中写入数据 |
| ALUOp[2:0] | O | ALU功能选择信号（详见ALU端口说明） |
| ALUSrc | O | ALU的B端口数据源选择信号  1：来源为立即数  0：来源为GRF的输出数据 |
| ExtOp | O | 位扩展方式选择信号  1：零扩展  0：符号扩展 |
| RegDst[1:0] | O | GRF的Addr3端口数据源选择信号  00：来源为Instr[20:16]  01：来源为Instr[15:11]  10：来源为常量31（$ra寄存器地址） |
| RegWrite | O | GRF写使能信号  1：可向GRF中写入数据  0：不能向GRF中写入数据 |

功能定义：

表12 Ctrl真值表（1）

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Funct | 100001 | 100011 |  | | | | | | 001000 |
| Op | 000000 | 000000 | 001101 | 100011 | 101011 | 000100 | 001111 | 000011 | 000000 |
|  | addu | subu | ori | lw | sw | beq | lui | jal | jr |
| nPCSrc[1:0] | 00 | 00 | 00 | 00 | 00 | 01 | 00 | 10 | 11 |
| RegSrc[1:0] | 00 | 00 | 00 | 01 | XX | XX | 10 | 11 | XX |
| MemWrite | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ALUOp[2:0] | 000 | 001 | 011 | 000 | 000 | 001 | XXX | XXX | XXX |
| ALUSrc | 0 | 0 | 1 | 1 | 1 | 0 | X | X | X |
| ExtOp | X | X | 1 | 0 | 0 | 0 | X | X | X |
| RegDst[1:0] | 01 | 01 | 00 | 00 | XX | XX | 00 | 10 | XX |
| RegWrite | 1 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 |

表12 Ctrl真值表（2）

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Funct |  | 001001 |  |  | |  |  |  |  | |  |
| Op | 000010 | 000000 |  | |  |  |  |  | |  |  |
|  | j | jalr |  | |  |  |  |  | |  |  |
| nPCSrc[1:0] | 10 | 11 |  | |  |  |  |  | |  |  |
| RegSrc[1:0] | XX | 11 |  | |  |  |  |  | |  |  |
| MemWrite | X | 0 |  | |  |  |  |  | |  |  |
| ALUOp[2:0] | XXX | XXX |  | |  |  |  |  | |  |  |
| ALUSrc | X | X |  | |  |  |  |  | |  |  |
| ExtOp | X | X |  | |  |  |  |  | |  |  |
| RegDst[1:0] | XX | 01 |  | |  |  |  |  | |  |  |
| RegWrite | 0 | 1 |  | |  |  |  |  | |  |  |

三、测试程序

先将数据存储器中地址为0x00000000、0x00000004、0x00000008、0x0000000c的值分别设置为1、2、3、4。

MIPS汇编代码：

.data

one: .word 1

two: .word 2

three: .word 3

four: .word 4

.text

lw $t0, 0($zero)

lw $t1, 4($zero)

lw $t2, 8($zero)

lw $t3, 12($zero)

jal function

sw $s0, 16($zero)

sw $s1, 20($zero)

sw $s2, 24($zero)

sw $s3, 28($zero)

jal end

function:

addu $s0, $t0, $t1

subu $s1, $t3, $t0

beq $s0, $s1, label

ori $s2, $s0, 123

label:

lui $s2, 123

ori $s3, $s2, 321

nop

jr $ra

end:

测试期望：

GRF中：

$8 = 0x00000001

$9 = 0x00000002

$10 = 0x00000003

$11 = 0x00000004

$16 = 0x00000003

$17 = 0x00000003

$18 = 0x007b0000

$19 = 0x007b0141

$31 = 0x00003028

DM中：

Mem[0x00000000] = 0x00000001

Mem[0x00000004] = 0x00000002

Mem[0x00000008] = 0x00000003

Mem[0x0000000c] = 0x00000004

Mem[0x00000010] = 0x00000003

Mem[0x00000014] = 0x00000003

Mem[0x00000018] = 0x007b0000

Mem[0x0000001c] = 0x007b0141

思考题

1. 若PC（程序计数器）位数为30位，试分析其与32位PC的优劣。

优：每个时钟周期PC的增量仅需加1，寻址时的计算更加简便；且可以使用更小容量的寄存器存储同样多的指令地址。

劣：无

1. 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为指令存储器，一般是只读的，在运行过程中不需要写入，所以适合使用ROM，且即使断电其中内容也不会丢失。DM为数据存储器，既需要读出也需要写入，所以要使用RAM。GRF为通用寄存器文件，因为修改频率高，每个时钟周期都可能会更新数据，所以用寄存器实现更加合适。

1. 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~op0func5~func4~func3~func2~func1~func0 + ~op5~op4~op3~op2~op1~op0func5~func4~func3~func2func1~func0

ALUSrc = ~op5~op4op3op2~op1op0 + op5~op4~op3~op2op1op0 + op5~op4op3~op2op1op0

MemtoReg = op5~op4~op3~op2op1op0

RegWrite = ~op5~op4~op3~op2~op1~op0func5~func4~func3~func2~func1~func0 + ~op5~op4~op3~op2~op1~op0func5~func4~func3~func2func1~func0 + ~op5~op4op3op2~op1op0 + op5~op4~op3~op2op1op0

nPC\_sel = ~op5~op4~op3op2~op1~op0

ExtOp = op5~op4~op3~op2op1op0 + op5~op4op3~op2op1op0

1. 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~op2~op1~op0func5~func4~func3~func2~func0

ALUSrc = ~op5~op4op3op2~op1op0 + op5~op4~op3~op2op1op0 + op5~op4op3~op2op1op0

MemtoReg = op5~op4~op3~op2op1op0

RegWrite = ~op5~op4~op3~op2~op1~op0func5~func4~func3~func2~func0 + ~op5~op4op3op2~op1op0 + op5~op4~op3~op2op1op0

nPC\_sel = ~op5~op4~op3op2~op1~op0

ExtOp = op5~op4~op2op1op0

1. 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。

不将nop加入控制信号真值表并不影响对其的译码，因为0x00000000不会使与逻辑中任何一个信号为真，因而也不会使或逻辑中任何一个控制信号为真。所有的写使能信号都为假，寄存器或存储器的状态都不会发生改变，相当于仅仅让PC增加4而没有进行任何其他有效操作，从而实现了空指令的功能。

1. 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。

将DM分成多片，地址的高位用作片选信号，则可以从某一片的零地址开始访问，使得两者保持一致。

1. 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。

优：能对所有可能的情况进行验证，保证绝对的正确性；验证时间短，可以很快发现和改正电路设计中的错误，缩短设计周期。

劣：无法考虑到某些实际情况，如电路延迟、元件质量问题等等。