**单周期cpu设计文档**

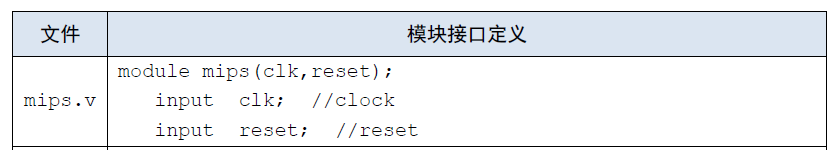
王郁含 16182672

**目录**

1. 设计与测试说明
2. CPU设计文档
3. 控制器设计
4. 测试程序
5. 思考题
   1. 设计与测试说明

* 处理器应支持指令集为：{addu, subu, ori, lw, sw, beq, lui, jal, jr,nop}。
* addu,subu可以不支持溢出。
* 处理器为单周期设计。
* 不需要考虑延迟槽。
* 顶层文件为mips.v，接口定义如下：

表1 模块接口定义



1. 模块规格
   1. IFU（取指令单元）
   * 包括 PC （程序计数器）、IM（指令存储器），及相关逻辑。
   * IM容量为4KB（32bit×1024字）。
   * 端口定义：

表2 IFU端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Clk | I | 时钟信号 |
| Reset | I | 复位信号 |
| [31:0]EXT | I | 32位偏移量 |
| BEQOp | I | 转移指令信号 |
| BEQZero | I | 转移比较判断信号 |
| JALOp | I | 跳转判断信号 |
| JROp | I | 跳转至寄存器判断信号 |
| [31:0]JRAdd | I | 跳转至寄存器的目标地址 |
| [31:0]PC | O | 当前PC值 |
| [31:0]PC4 | O | 下一个PC值 |
| Instr | O | 32位指令信号 |

* 功能描述：

表3 IFU功能描述

|  |  |
| --- | --- |
| 功能 | 功能描述 |
| 复位清零 | 当Reset上升沿时将PC置为0x00003000，如果时钟上升沿到来时信号仍有效，则仍然置零 |
| 取下一条地址上的指令 | 当时钟上升沿到来且跳转控制失效时，PC + 4以获取下一地址，使IM读取下一条指令 |
| 转移至对应地址获取指令 | 当时钟上升沿到来且转移控制有效，即为BranchE和zero均为1时，使PC + 4 + sign\_extend(offset||02)以转移至目标地址获取该地址内指令 |
| 跳转至对应地址获取指令 | 当时钟上升沿到来且跳转控制有效，即为jump为1时，跳转至31..28||target地址获取其中指令 |
| 跳转至寄存器中地址取指令 | 当时钟上升沿到来且跳转至寄存器控制有效，即为jr为1时，跳转至Addr地址获取其中指令 |

2. GRF（通用寄存器组）

* + 用32个具有写使能的寄存器实现。
  + 0号寄存器保持为0。
  + 端口定义：

表4 GRF端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [4:0]RA1 | I | 读取寄存器1 |
| [4:0]RA2 | I | 读取寄存器2 |
| [4:0]WA | I | 写入寄存器地址 |
| [31:0]WD | I | 写入数据 |
| Reset | I | 复位信号 |
| clk | I | 时钟信号 |
| RWE | I | 写使能信号 |
| [31:0]PC | I | 当前指令信号 |
| [31:0]D1 | O | 读取数据1 |
| [31:0]D2 | O | 读取数据2 |

* + 功能描述：

表5 GRF 功能描述

|  |  |
| --- | --- |
| 功能 | 功能描述 |
| 复位清零 | 当Reset上升沿时，将所有寄存器清零，如果时钟上升沿到来时信号仍有效，则仍然置零 |
| 读取寄存器中的数据 | 当时钟上升沿到来且写使能WE失效时，读取RA1和RA2两个对应号的寄存器中的数据并分别输出至RD1和RD2 |
| 读取寄存器中的数据并将数据写入寄存器 | 当时钟上升沿到来且写使能WE有效时，读取RA1和RA2两个对应号的寄存器中的数据并分别输出至RD1和RD2同时将WD的数据写入WA对应号的寄存器 |

3. ALU（算术逻辑单元）

* + 提供32位加、减、或运算及大小比较功能。
  + 可以不支持溢出。
  + 端口定义：

表6 ALU 端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [31:0] A | I | 32位被运算数据 |
| [31:0] B | I | 32位被运算数据 |
| [1:0] ALUOp | I | 功能选择信号  00:加运算  01:减运算  10:按位或运算  11:BEQ比较 |
| [31:0] ALUOut | O | 计算结果 |
| BEQZero | O | 比较A、B，相等时为1 |

* + 功能描述：

表7 ALU 功能描述

|  |  |
| --- | --- |
| 功能 | 功能描述 |
| 加运算 | ALUOut = A + B |
| 减运算 | ALUOut = A - B |
| 按位或运算 | ALUOut = A | B |
| 相等比较 | if(A == B) BEQZero = 1 |

1. DM（数据存储器）

* DM容量为4KB（32bit×1024字）。
* 端口定义：

表8 DM 端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [4:0] Add | I | 访问的地址 |
| [31:0] WD | I | 写入地址的数据 |
| DWE | I | 写入使能端 |
| DRE | I | 读取使能端 |
| Reset | I | 复位信号 |
| Clk | I | 时钟信号 |
| [31:0] RD | O | 读取地址的数据 |

* 功能描述

表9 DM 功能描述

|  |  |
| --- | --- |
| 功能 | 功能描述 |
| 复位清零 | 当Reset上升沿时，将所有地址清零，如果时钟上升沿到来时信号仍有效，则仍然置零 |
| 写入地址 | 当时钟上升沿且WE为1时在Addr所指地址中写入WD所代表数据 |
| 读取地址 | 当时钟上升沿且且RE为1时将Addr所指地址中的数据读出至RD |

1. EXT(位数扩展器)

* 端口定义：

表10 EXT 端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| [15:0] Imm | I | 16位立即数/偏移量输入 |
| [1:0]EXTOp | I | EXT控制端：  00：无符号扩展  01：有符号扩展  10：有符号扩展+左移两位  11: 左移16位 |
| [31:0] EXT | O | 扩展后的32位立即数/偏移量输出 |

* 功能描述：

表11 EXT 功能描述

|  |  |
| --- | --- |
| 功能 | 功能描述 |
| 无符号扩展 | 将16位输入进行无符号扩展至32位 |
| 有符号扩展 | 当指令为sw/lw时，对16位输入进行有符号扩展至32位 |
| 有符号扩展+左移两位 | 当指令为beq时，对16位输入进行有符号扩展至32位 |
| 左移16位 | 当指令为lui时，将16位输入扩展至32位后移至高16位 |

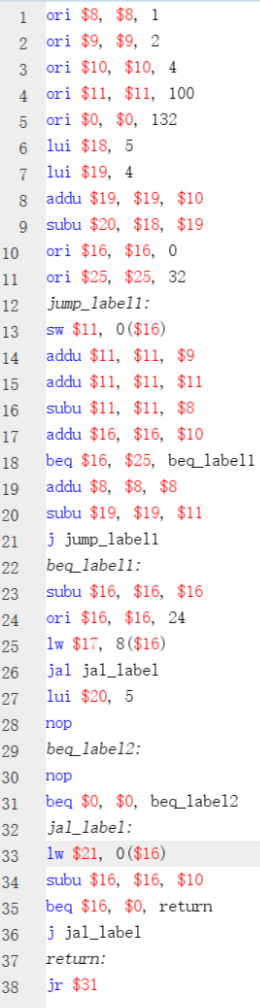
1. 控制器设计

表12 控制信号

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| func | | | 100001 | | 100011 | | 001000 | | n/a | | |  | |
| op | 000000 | 000000 | | 000000 | | 001101 | 100011 | 101011 | | 000100 | 001111 | | 000011 |
| 指令 | addu | subu | | jr | | ori | lw | sw | | beq | lui | | jal |
| WAOp | 01 | 01 | | x | | 00 | 00 | x | | x | 00 | | 10 |
| WDOp | 00 | 00 | | x | | 00 | 01 | x | | x | 00 | | 10 |
| BEQOp | 0 | 0 | | 0 | | 0 | 0 | 0 | | 1 | 0 | | 0 |
| ALUBOp | 1 | 1 | | x | | 0 | 0 | 0 | | 1 | 0 | | x |
| EXTOp | x | x | | x | | 00 | 01 | 01 | | 10 | 11 | | x |
| RWE | 1 | 1 | | 0 | | 1 | 1 | 0 | | 0 | 1 | | 1 |
| DWE | 0 | 0 | | 0 | | 0 | 0 | 1 | | 0 | 0 | | 0 |
| RE | 0 | 0 | | 0 | | 0 | 1 | 0 | | 0 | 0 | | 0 |
| JALOp | 0 | 0 | | 0 | | 0 | 0 | 0 | | 0 | 0 | | 1 |
| JROp | 0 | 0 | | 1 | | 0 | 0 | 0 | | 0 | 0 | | 0 |
| ALUOP | Add00 | Subtract01 | | x | | Or10 | Add00 | Add00 | | Cmp11 | Add00 | | x |

1. 测试程序

35080001 Mars中等效MIPS指令：

35290002

354a0004

356b0064

34000084

3c120005

3c130004

026a9821

0253a023

36100000

37390020

ae0b0000

01695821

016b5821

01685823

020a8021

12190003

01084021

026b9823

08000c0b

02108023

36100018

8e110008

0c000c1c

3c140005

00000000

00000000

1000fffe

8e150000

020a8023

12000001

08000c1c

03e00008

期望输出

$ 8 <= 00000001

$ 9 <= 00000002

$10 <= 00000004

$11 <= 00000064

$18 <= 00050000

$19 <= 00040000

$19 <= 00040004

$20 <= 0000fffc

$16 <= 00000000

$25 <= 00000020

\*00000000 <= 00000064

$11 <= 00000066

$11 <= 000000cc

$11 <= 000000cb

$16 <= 00000004

beq $16, $25, 3 不转移

$ 8 <= 00000002

$19 <= 0003ff39

j 0x0000302c 跳转至 sw $11, 0($16)

\*00000004 <= 000000cb

$11 <= 000000cd

$11 <= 0000019a

$11 <= 00000198

$16 <= 00000008

beq $16, $25, 3 不转移

$ 8 <= 00000004

$19 <= 0003fda1

j 0x0000302c 跳转至 sw $11, 0($16)

\*00000008 <= 00000198

$11 <= 0000019a

$11 <= 00000334

$11 <= 00000330

$16 <= 0000000c

beq $16, $25, 3 不转移

$ 8 <= 00000008

$19 <= 0003fa71

j 0x0000302c 跳转至 sw $11, 0($16)

\*0000000c <= 00000330

$11 <= 00000332

$11 <= 00000664

$11 <= 0000065c

$16 <= 00000010

beq $16, $25, 3 不转移

$ 8 <= 00000010

$19 <= 0003f415

j 0x0000302c 跳转至 sw $11, 0($16)

\*00000010 <= 0000065c

$11 <= 0000065e

$11 <= 00000cbc

$11 <= 00000cac

$16 <= 00000014

beq $16, $25, 3 不转移

$ 8 <= 00000020

$19 <= 0003e769

j 0x0000302c 跳转至 sw $11, 0($16)

\*00000014 <= 00000cac

$11 <= 00000cae

$11 <= 0000195c

$11 <= 0000193c

$16 <= 00000018

beq $16, $25, 3 不转移

$ 8 <= 00000040

$19 <= 0003ce2d

j 0x0000302c 跳转至 sw $11, 0($16)

\*00000018 <= 0000193c

$11 <= 0000193e

$11 <= 0000327c

$11 <= 0000323c

$16 <= 0000001c

beq $16, $25, 3 不转移

$ 8 <= 00000080

$19 <= 00039bf1

j 0x0000302c 跳转至 sw $11, 0($16)

\*0000001c <= 0000323c

$11 <= 0000323e

$11 <= 0000647c

$11 <= 000063fc

$16 <= 00000020

beq $16, $25, 3 转移至subu $16, $16, $16

$16 <= 00000000

$16 <= 00000018

$17 <= 00000000

$31 <= 00003060

jal 0x00003070 跳转至lw $21, 0($16)

$21 <= 0000193c

$16 <= 00000014

beq $16, $0, 1,不转移

$21 <= 00000cac

$16 <= 00000010

beq $16, $0, 1,不转移

$21 <= 0000065c

$16 <= 0000000c

beq $16, $0, 1,不转移

$21 <= 00000330

$16 <= 00000008

beq $16, $0, 1,不转移

$21 <= 00000198

$16 <= 00000004

beq $16, $0, 1,不转移

$21 <= 000000cb

$16 <= 00000000

beq $16, $0, 1,转移至jr $31

$20 <= 00050000

1. 思考题
2. 根据你的理解，在下面给出的DM的输入示例中，地址信号addr位数为什么是[11:2]而不是[9:0]？这个addr信号又是从哪里来的？

答：因为地址信号是4的整数倍，而存的时候是+1而不是+4，所以后两位可以不看。Addr来源是aluout，也就是偏移量+寄存器读出的第一个值。

1. 在相应的部件中，reset的优先级比其他控制信号（不包括clk信号）都要高，且相应的设计都是同步复位。清零信号reset是针对哪些部件进行清零复位操作？这些部件为什么需要清零？

答：对寄存器，pc，dm。因为他们是被修改的。所以reset时候都要reset。反之im就不需要。

1. 列举出用Verilog语言设计控制器的几种编码方式（至少三种），并给出代码示例。

答：

第一种：

If(R\_format) begin

……

end else if(beq) begin

……

end

……

第二种：

assign R\_format = ……;

assign beq = ……;

……

第三种：

`define R\_format ……

`define beq ……

……

1. 根据你所列举的编码方式，说明他们的优缺点。

答：第一种通过分支语句实现，优点在于逻辑清晰，但由于分支语句需要放在always语句块中，会导致控制信号变成一个没有时钟的时序电路；第二种优点在于其满足组合逻辑的特点，即输出一直与输入有关，缺点在于需要大量的导线。第三种可以将每一种/条指令作为一个状态使用，在每种状态下相关信号变动可一目了然。

1. 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：addi和add指令执行时会有一步判断临时值的temp32是否等于temp31，即判断是否有符号位溢出，若符号位在32位且不在31位时，则说明溢出，在不考虑溢出位时，通过add或addi计算出的temp31..0和通过addu或addiu计算出的temp31..0是相同的，所以add和addu等价，addi和addiu等价。

1. 根据自己的设计说明单周期处理器的优缺点。

答：单周期cpu控制简单，逻辑清晰，不存在冲突。但是单条指令执行时间较长，效率低。

1. 简要说明jal、jr和堆栈的关系。

答：jal指令实现进栈，jr实现出栈。