CPU设计文档

目录：

[一、设计要求 1](#_Toc531206403)

[1.模块化与层次化设计要求 1](#_Toc531206404)

[2.设计说明 1](#_Toc531206405)

[二、模块设计 2](#_Toc531206406)

[1. IFU（取指令单元） 2](#_Toc531206407)

[2.NPC（计算下一PC） 2](#_Toc531206408)

[3. GRF（寄存器堆） 3](#_Toc531206409)

[4. ALU（算术逻辑单元） 4](#_Toc531206410)

[5. DM（数据存储器） 4](#_Toc531206411)

[6. EXT（位扩展器） 5](#_Toc531206412)

[三、数据通路设计 6](#_Toc531206413)

[四、控制器设计 7](#_Toc531206414)

[五、测试程序 9](#_Toc531206415)

[六、思考题 10](#_Toc531206416)

[L0.T2 10](#_Toc531206417)

[L0.T4 10](#_Toc531206418)

[L0.T5 11](#_Toc531206419)

一、设计要求

1.模块化与层次化设计要求

（1）单周期处理器包括控制器和数据通路，将其放入mips.v的层次下。code.txt中存储相应的指令码。

（2）control模块占一个独立的Verilog HDL文件，实现控制器这个单一的职责，保持模块不受污染。

（3）datapath中的每个module都由一个独立的Verilog HDL文件组成。

（4）所有mux都建模在一个mux.v中。

2.设计说明

（1）处理器应支持指令集为：{addu, subu, ori, lw, sw, beq, lui, jal, jr,nop}。

（2）addu,subu不支持溢出。

（3）处理器为单周期设计。

（4）不考虑延迟槽。

二、模块设计

1. IFU（取指令单元）

（1）PC（程序计数器）

PC用[31:0] reg实现，具有复位功能。起始地址：0x00003000。

（2）IM（指令存储器）

IM用[31:0] reg [1023:0]实现，容量为32bit×1024。以读取文件的方式将code.txt中指令加载至IM中。

由于IM地址宽度为10位，PC为32位且起始地址为0x00003000，故通过PC取IM中指令时需要选择PC的2-11位。

（3）端口定义与功能描述

表2.1.1 IFU模块端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| NPC[31:0] | I | 由NPC计算出的下一个PC的值 |
| clk | I | 时钟信号 |
| reset | I | 同步复位信号 |
| PC[31:0] | O | 当前PC的值 |
| Instr[31:0] | O | 输出当前PC所指的指令 |

表2.1.2 IFU模块功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能描述 | 描述 |
| 1 | 复位 | 时钟上升沿时，若reset信号有效，PC指向0x0000\_3000 |
| 2 | 取指令 | 根据PC取出IM中相应指令并输出 |

2.NPC（计算下一PC）

表2.2.1 NPC模块端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| Imm[15:0] | I | 由上一条指令提取出的16位beq指令立即数 |
| Jal[25:0] | I | 由上一条指令提取出的26位jal指令立即数 |
| rs[31:0] | I | 由GRF输出用于jr指令的32位数值 |
| Br[1:0] | I | 控制信号，确定下一指令的计算方法 |
| PC[31:0] | I | 当前执行指令地址 |
| PC4[31:0] | O | 输出PC+4的值 |
| NPC[31:0] | O | 下一条应执行的指令地址 |

表2.2.2 NPC模块功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能描述 | 描述 |
| 1 | 连续指令 | 执行下一条指令，NPC = PC + 4 |
| 2 | 分支指令beq | 跳转，NPC = PC + 4 + {14{Imm[15]}，Imm，0，0} |
| 3 | 跳转指令jal | 跳转，NPC = {PC[31:28] ，Jal，0，0} |
| 4 | 跳转指令jr | 跳转，NPC = rs |

3. GRF（寄存器堆）

用具有写使能的寄存器实现，寄存器总数为32个，0号寄存器的值始终保持为0，其他寄存器初始值均为0。

表2.3.1 GRF模块端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 同步复位信号，将32个寄存器中的值全部清零 |
| WE | I | 写使能信号。 |
| A1[4:0] | I | 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RD1 |
| A2[4:0] | I | 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RD2 |
| A3[4:0] | I | 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 |
| WD[31:0] | I | 32位数据输入信号 |
| RD1[31:0] | O | 输出A1指定的寄存器中的32位数据 |
| RD2[31:0] | O | 输出A2指定的寄存器中的32位数据 |

表2.3.2 GRF模块功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能描述 | 描述 |
| 1 | 复位 | 时钟上升沿时，若reset信号有效，所有寄存器数值清零 |
| 2 | 读数据 | 读出A1，A2地址对应寄存器中所存储的数据到RD1，RD2 |
| 3 | 写数据 | 当WE有效且时钟上升沿来临时，将WD写入A3所对应的寄存器中 |

4. ALU（算术逻辑单元）

提供32位加、减、或运算及大小比较功能，其中加减运算不检测溢出。

表2.4.1 ALU模块端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | ALU的第一个操作数 |
| B[31:0] | I | ALU的第二个操作数 |
| ALUOp[1:0] | I | 控制ALU的计算种类 |
| C[31:0] | O | ALU的运算结果 |

表2.4.2 ALU模块功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能描述 | 描述 |
| 1 | 加运算 | C = A + B |
| 2 | 减运算 | C = A - B |
| 3 | 或运算 | C = A | B |
| 4 | 大小比较 | 若A = B，则C = 1，否则C = 0 |

5. DM（数据存储器）

使用[31:0] reg [1023:0]实现，容量为32bit×1024。起始地址：0x00000000。

表2.5.1 DM模块端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[9:0] | I | 指向DM的地址 |
| Input[31:0] | I | 将要写入DM的数据 |
| clk | I | 时钟信号 |
| reset | I | 异步复位信号 |
| MemWr | I | DM写使能信号 |
| D[31:0] | O | DM输出的数据 |

表2.5.2 DM模块功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能描述 | 描述 |
| 1 | 复位 | 时钟上升沿时，若reset信号有效，所有寄存器数值清零 |
| 2 | 读数据 | 读出A地址中所存储的数据到D |
| 3 | 写数据 | 当MemWr信号有效且时钟上升沿来临时，将Input输入的数据写入A所对应的地址中 |

6. EXT（位扩展器）

表2.6.1 EXT模块端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[15:0] | I | 待扩展数 |
| EXTOp[1:0] | I | 控制扩展类型 |
| B[31:0] | O | 输出扩展结果 |

表2.6.2 EXT模块功能描述

|  |  |  |
| --- | --- | --- |
| 序号 | 功能描述 | 描述 |
| 1 | 高位0扩展 | B = {16{0}，A} |
| 2 | 低位0扩展 | B = {A，16{0}} |
| 3 | 符号扩展 | B = {16{A[15]}，A} |
| 4 | 地址扩展 | B = {14{A[15]}，A，0，0} |

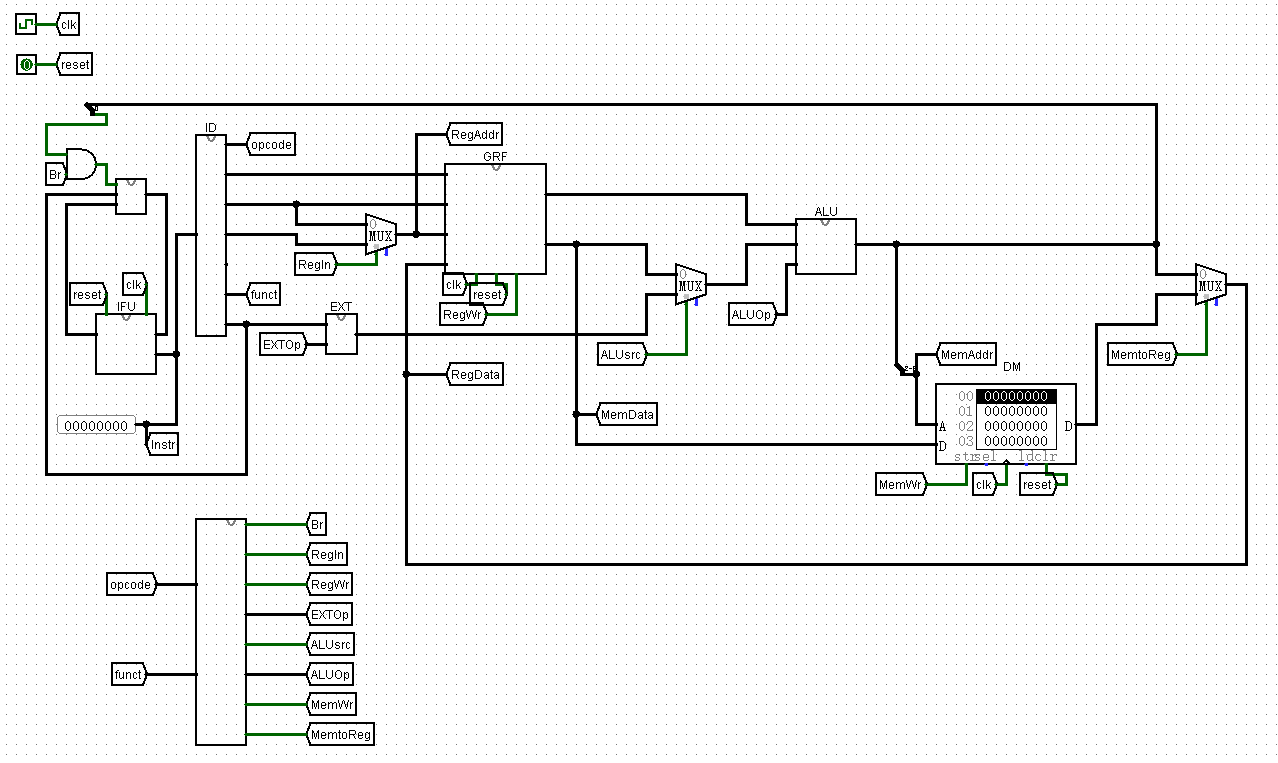
三、数据通路设计

根据P3时设计的数据通路架构图以及各个部件模块的输入输出端口，在datapath.v中定义一些内部的变量，利用模块间的逻辑关系，将各个部件模块串接在一起，使之成为一个整体，并预留出控制器信号的输入和输出端口。

表3.1 datapath模块端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| reset | I | 同步复位信号 |
| clk | I | 时钟信号 |
| instr[31:0] | O | 当前指令 |
| Br[1:0] | I | 分支/跳转判断信号 |
| RegIn, | I | GRF输入地址来源控制信号 |
| RegWr, | I | GRF写使能信号 |
| EXTOp[1:0] | I | EXT扩展方式控制信号 |
| ALUsrc, | I | ALU第二个运算数来源控制信号 |
| ALUOp[1:0] | I | ALU计算方式控制信号 |
| MemWr | I | DM写使能信号 |
| MemtoReg | I | GRF输入来源控制信号 |

图3.1 P3数据通路架构图

****

四、控制器设计

表4.1 控制信号的意义

|  |  |
| --- | --- |
| 控制信号 | 功能名称 |
| Br | Br = 00时，该指令为常规指令，NPC = PC + 4；  Br = 01时，该指令为分支指令beq，NPC = PC + 4 + sign\_ext(Imm||00)。  Br = 10时，该指令为跳转指令jal，NPC = {PC[31:28] ，Jal，0，0}。  Br = 11时，该指令为跳转指令jr，NPC = rs。 |
| RegIn | RegIn = 00时，数据写入rt寄存器；  RegIn = 01时，数据写入rd寄存器。  RegIn = 10时，数据写入31号寄存器。 |
| RegWr | RegWr = 0时，GRF不能写入数据；RegWr = 1时，GRF可以写入数据。 |
| EXTOp | EXTOp = 00时，进行高位0扩展，B = {16{0}，A}；  EXTOp = 01时，进行低位0扩展，B = {A，16{0}}；  EXTOp = 10时，进行符号扩展，B = {16{A[15]}，A}；  EXTOp = 11时，进行地址扩展，B = {14{A[15]}，A，0，0}。 |
| ALUSrc | ALUSrc = 0时，ALU的第二个运算数来自rt寄存器；  ALUSrc = 1时，ALU的第二个运算数来自位扩展之后的立即数。 |
| ALUOp | ALUOp = 00时，进行加运算，C = A + B；  ALUOp = 01时，进行减运算，C = A - B；  ALUOp = 10时，进行或运算，C = A | B；  ALUOp = 11时，进行大小比较，若A = B，则C = 1，否则C = 0。 |
| MemWr | MemWr = 0时，DM不能写入数据；MemWr = 1时，DM可以写入数据。 |
| ToReg | ToReg = 00时，写入寄存器的数据来自ALU；  ToReg = 01时，写入寄存器的数据来自DM。  ToReg = 01时，写入寄存器的数据来自NPC。 |

表43.2控制信号产生的真值表

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | opcode | funct | Br | RegIn | RegWr | EXTOp | ALUSrc | ALUOp | MemWr | ToReg |
| addu | 000000 | 100001 | 00 | 01 | 1 | X | 0 | 00 | 0 | 00 |
| subu | 000000 | 100011 | 00 | 01 | 1 | X | 0 | 01 | 0 | 00 |
| ori | 001101 |  | 00 | 00 | 1 | 00 | 1 | 10 | 0 | 00 |
| lw | 100011 |  | 00 | 00 | 1 | 10 | 1 | 00 | 0 | 01 |
| sw | 101011 |  | 00 | X | 0 | 10 | 1 | 00 | 1 | X |
| beq | 000100 |  | 01 | X | 0 | X | 0 | 11 | 0 | X |
| lui | 001111 |  | 00 | 00 | 1 | 01 | 1 | 00 | 0 | 00 |
| nop | 000000 |  | 00 | 00 | 0 | 0 | 0 | 0 | 0 | 00 |
| jal | 000011 |  | 10 | 10 | 1 | X | X | X | 0 | 10 |
| jr | 000000 | 001000 | 11 | X | 0 | X | X | X | 0 | X |

附表 支持指令集编码与操作

|  |  |  |
| --- | --- | --- |
| 指令 | 编码形式 | 操作 |
| addu | 000000 rs rt rd 00000 100001 | GPR[rd] = GPR[rs] + GPR[rt] |
| subu | 000000 rs rt rd 00000 100011 | GPR[rd] = GPR[rs] - GPR[rt] |
| ori | 100011 base rt offset | GPR[rt] = GPR[rs] OR zero\_extend(immediate) |
| lw | 100011 base rt offset | Addr = GPR[base] + sign\_ext(offset)  GPR[rt] = memory[Addr] |
| sw | 101011 base rt offset | Addr = GPR[base] + sign\_ext(offset)  memory[Addr] = GPR[rt] |
| beq | 000100 rs rt offset | if (GPR[rs] == GPR[rt])  PC = PC + 4 + sign\_extend(offset||00)  else  PC = PC + 4 |
| lui | 001111 00000 rt immediate | GPR[rt] = immediate||0^16 |
| jal | 000011 instr\_index | PC = PC[31:28] || instr\_index || 00  GPR[31] = PC + 4 |
| jr | 000000 rs 10{0} 5{0} 001000 | PC = GPR[rs] |
| nop | 0x00000000 |  |

五、测试程序

表5 CPU测试程序与期望输出

|  |  |
| --- | --- |
| **测试程序** | **期望输出或现象** |
| 1 ori $t0,$t0,0xffff | 00003000: $ 8 <= 0000ffff |
| 2 lui $t1,0xffff | 00003004: $ 9 <= ffff0000 |
| 3 addu $t2,$t0,$t1 | 00003008: $10 <= ffffffff |
| 4 subu $t3,$t1,$t0 | 0000300c: $11 <= fffe0001 |
| 5 nop | 无现象 |
| 6 beq $t1,$t1,yes | 跳转至第13行继续执行 |
| 7 *jjj：* |  |
| 8 ori $t4,4 | 00003018: $12 <= 0000000c |
| 9 sw $t3,0($t4) | 0000301c: \*0000000c <= fffe0001 |
| 10 lw $t5,0($t4) | 00003020: $13 <= fffe0001 |
| 11 jr $ra | 跳转至第18行继续执行 |
| 12 *yes:* |  |
| 13 beq $t0,$t1,jjj | 不跳转，顺序执行 |
| 14 ori $t4,8 | 0000302c: $12 <= 00000008 |
| 15 sw $t3,0($t4) | 00003030: \*00000008 <= fffe0001 |
| 16 lw $t5,0($t4) | 00003034: $13 <= fffe0001 |
| 17jal jjj | 00003038: $31 <= 0000303c |
| 18 ori $t6,$t6,0x1111 | 0000303c: $14 <= 00001111 |

六、思考题

L0.T2

Q1、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？



A1：方便连接，直观。addr信号从ALU的输出经过位数处理后得到。

Q2、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？

A2： GRF和DM。因为内部存在寄存器，在复位时需要清零。IFU里的PC寄存器需要复位，但复位时不清零，而是变成0x00003000。

L0.T4

Q1、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。

A1、（1）利用if-else（或case）完成操作码和控制信号的值之间的对应；

if (op == `lw || op == `sw) EXTOp = `EXT\_SIGN;

else if( op == `lui ) EXTOp =`EXT\_LO0 ;

else EXTOp =`EXT\_UP0 ;

（2）利用assign语句完成操作码和控制信号的值之间的对应；

assign EXTOp = (op == `lw || op == `sw) ? `EXT\_SIGN :

op == `lui ? `EXT\_LO0 : `EXT\_UP0 ;

（3）利用宏定义

`define ori op == 6’b001101

Q2、根据你所列举的编码方式，说明他们的优缺点。

A2:第一种直观但是麻烦，第二种简介但是难懂，第三种为了好看？？

L0.T5

Q1、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 。

A1、addi与addiu的区别是addi考虑溢出而addiu不考虑溢出，因此在忽略溢出的前提下，addi与addiu是等价的。add与addu同理。

Q2、根据自己的设计说明单周期处理器的优缺点。

A2、优点：结构简单，一周期处理一条指令，不需要考虑前后代码关系。

缺点：时钟周期长，慢。

Q3、简要说明jal、jr和堆栈的关系。

A3、在调用函数时，jal和jr所需要的地址一般存储在堆栈中。