**计算机组成原理实验报告**

**五级流水线处理器开发**

班级：1618402

组员：161840201印玥

161840202尚怡帆

161840203刘孟怡

161840204张爔文

专业：物联网工程

**2020.06.26**

# 一、功能设计说明

### 1. 完成的指令集（中级，共45条指令）

算术运算指令：

add, addi, addu, addiu, sub, subu, slt, slti, sltu, sltiu

逻辑运算指令：

and, andi, or, ori, lui, nor, xor, xori

移位指令：

sll, sllv, srl, srlv, sra, srav

分支跳转指令：

Beq, bne, bgez, bgtz, blez, bltz, j, jr, jal

访存指令：

lb, lbu, lh, lhu, lw, sb, sh, sw

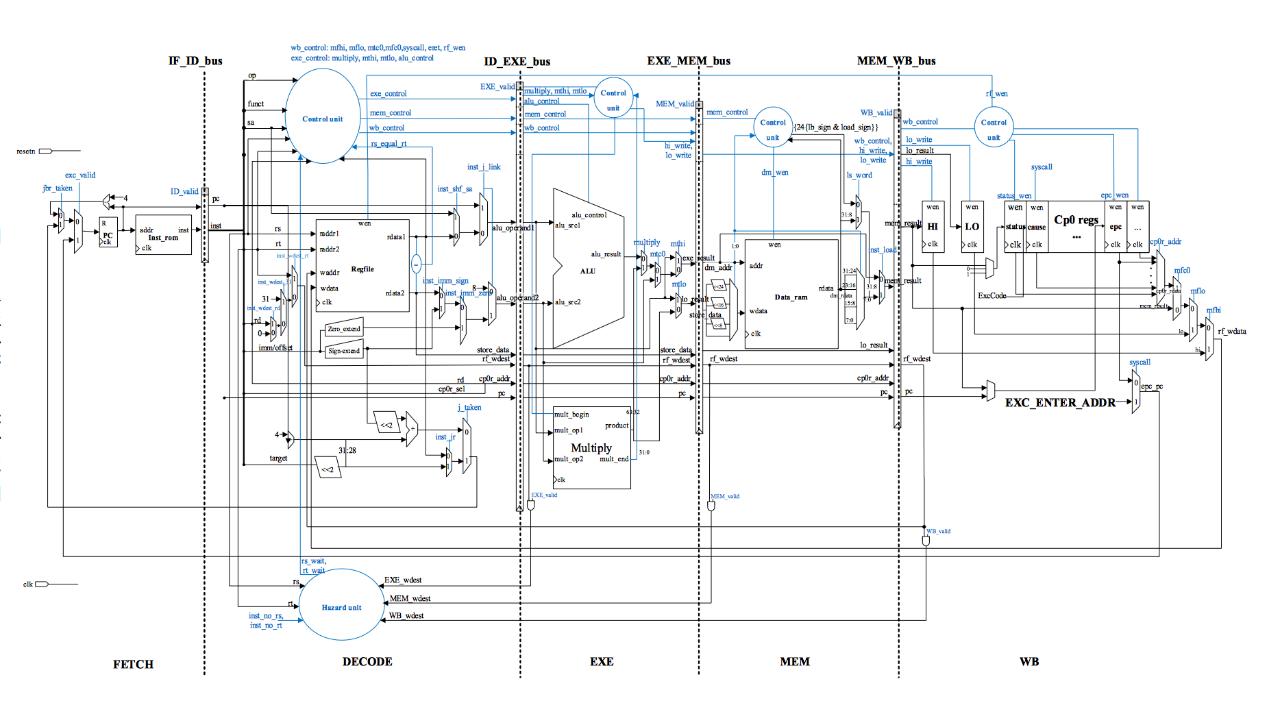
数据移动指令：mfhi, mflo

特权指令：mfc0, mtc0

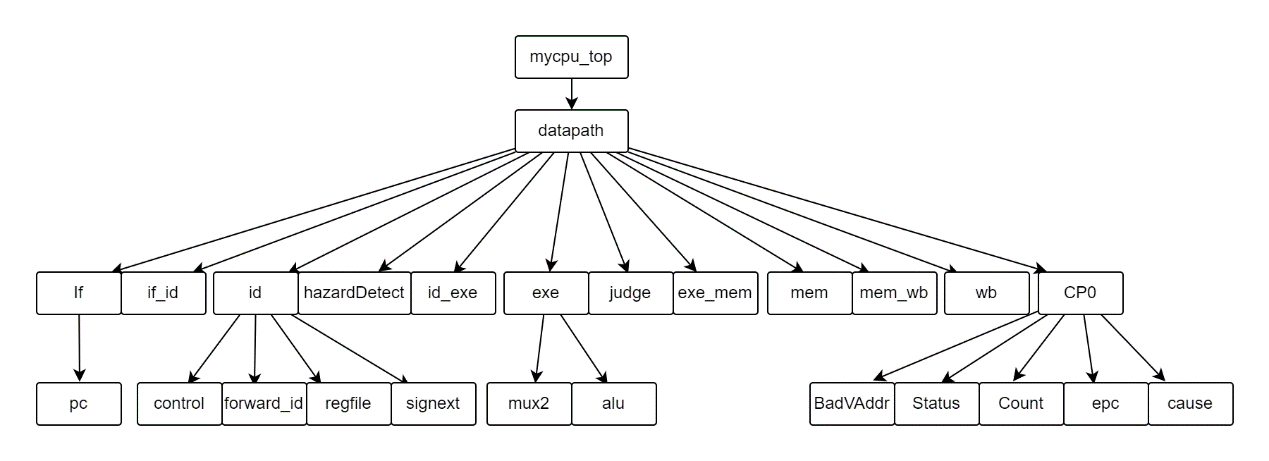
### 2. 处理器支持溢出异常，保留指令例外。

### 3. 处理器为流水线设计，支持各类冒险与转发。

### 4. 参考的指示图如下，并做出了一定的修改。



# 二、模块化和层次化设计说明



# 三、模块定义

### 1. If模块

（1）基本描述

根据输入的控制信号，选择正确的npc后，将当前指令的地址和下一条指令的地址一起传入下一级。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号，  1：复位， 0无效 |
| address[31:0] | I | 跳转指令的跳转地址 |
| error\_address[31:0] | I | 当发生溢出异常或保留指令例外时跳转的指令地址 |
| pc\_in[31:0] | I | 当前pc的值 |
| PcWrite | I | 是否更新当前pc，  1：是， 0：否 |
| pc\_src | I | npc选择控制信号，  1：选择跳转地址， 0：选择pc+4 |
| pc\_flush | I | flush控制信号  1：保持当前pc不变，0：pc正常变化 |
| if\_lw | I | 前前一条指令是否是lw，并存在load\_use数据冒险  1：是， 0：否 |
| error | I | 是否发生溢出异常或保留指令例外  1：是， 0：否 |
| pc\_4[31:0] | O | 保存并输出下一条指令的地址 |
| pc\_out | O | 保存并输出当前指令的地址 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 选择正确的npc | 根据控制信号选择跳转地址或者当前pc+4作为npc |
| 2 | 复位 | 当复位信号有效时，pc被设置为32'hbfc0\_0000 |
| 3 | 处理异常 | 当发生溢出异常或保留指令例外时，将会跳转到处理这些异常的指令地址 |
| 4 | 暂停 | 当pc\_flush或if\_lw为1时，停止读入新的npc |
| 5 | 输出 | 将当前指令的地址和下一条指令的地址保存并一起传入下一级 |

### 2. pc模块

（1）基本描述

完成输出当前指令地址并保存下一条指令地址。复位后，pc指向32'hbfc0\_0000，此处为了适应trace对比机制。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| ena | I | 使能信号，为1时npc赋值给pc\_out |
| clk | I | 时钟信号 |
| rst | I | 复位信号，  1：复位， 0无效 |
| npc[31:0] | I | 下一条指令的地址 |
| pc\_out[31:0] | O | 保存并输出当前指令的地址 |

（3）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32'hbfc0\_0000 |
| 2 | 保存npc并输出 | 在每个clk的上升沿保存npc，并输出 |

### 3. if\_id模块

（1）基本描述

if级和id级中间的流水线寄存器，用于保存前一级流水线的数据并输出至下一级。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| if\_flush | I | 清零信号，为1时输出皆为0 |
| if\_lw | I | 前前一条指令是否是lw，并存在load\_use数据冒险  1：是， 0：否 |
| if\_id\_write | I | if\_id流水线控制信号  1：当前流水线寄存器正常写入  0：当前流水线寄存器不写入 |
| pc[31:0] | I | 当前pc的值 |
| pc\_4[31:0] | I | 下一条指令的地址 |
| pc\_out[31:0] | O | 保存并输出当前pc的值 |
| pc\_4\_out[31:0] | O | 保存并输出下一条指令的地址 |

（3）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，流水线寄存器清零 |
| 2 | 保存并输出前一级指令的数据 | 在每个clk的上升沿保存前一级的数据并输出 |
| 3 | 控制写入 | 通过if\_id\_write控制该寄存器的写入 |

### 4. id模块

（1）基本描述

根据指令从32个32位的寄存器中读写对应的值，得到当前指令对应的控制信号，同时使用forward\_id模块解决数据冒险。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| ctrl | I | 当前指令是否为跳转指令（jal除外）  1：是， 0：否 |
| id\_lw | I | 前一条指令是否为lw，并存在load\_use数据冒险  1：是， 0：否 |
| id\_lw\_exe | I | exe级的id\_lw |
| pc[31:0] | I | 当前pc的值 |
| pc\_4[31:0] | I | 当前pc\_4的值 |
| instruction[31:0] | I | 当前指令的值 |
| Aluout\_exe[31:0] | I | exe级的alu计算后的结果，用于解决数据冒险 |
| Aluout\_mem[31:0] | I | mem级的alu计算后的结果，用于解决数据冒险 |
| Rdata\_mem[31:0] | I | mem级读内存得到的数据 |
| data\_wb[31:0] | I | wb级写入寄存器的数据 |
| rd\_exe[4:0] | I | exe级的rd寄存器 |
| rd\_mem[4:0] | I | mem级的rd寄存器 |
| rd\_wb[4:0] | I | wb级的rd寄存器 |
| RegWrite\_exe | I | exe级的RegWrite控制信号 |
| RegWrite\_mem | I | mem级的RegWrite控制信号 |
| RegWrite\_wb | I | wb级的RegWrite控制信号 |
| MemRead\_mem[2:0] | I | mem级的读内存控制信号 |
| RegDst，  Branch，  MemtoReg，  Alusrc1，  Alusrc2，  RegWrite，  Jump[1:0]，  keep，  MemWrite[1:0],  MemRead[2:0],  Aluctr[4:0],  mtc0,  mfc0 | O | 当前指令对应control模块的控制信号，  keep是用来处理保留指令例外 |
| rs[4:0],  rt[4:0],  rd[4:0] | O | 当前指令对应的rs，rt，rd寄存器 |
| immi\_1[31:0],  immi\_2[31:0] | O | immi\_1是instruction[10:6]零扩展得到的数，用来处理移位指令；  immi\_2是根据控制信号Extop扩展之后得到的立即数 |
| out1[31:0],  out2[31:0] | O | 从寄存器读出并解决数据冲突后的值 |
| pc\_4\_out[31:0] | O | 输出pc\_4 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 执行指令 | 解析读入的指令，读取并输出相应的值和控制信号 |
| 2 | 解决数据冒险 | 判断是否发生数据冒险并从后两级读取相应的值 |
| 3 | 立即数扩展 | 直接根据控制信号Extop扩展，得到32位立即数 |

### 5. control模块

（1）基本描述

根据输入的指令产生对应的控制信号，控制各个部件的执行。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| ctrl | I | 当前指令是否为跳转指令（jal除外），用来保证跳转指令的id级之后的控制信号均为0 |
| op[5:0] | I | 当前指令的op字段 |
| func[5:0] | I | 当前指令的func字段 |
| instruction[31:0] | I | 当前指令的值 |
| RegDst | O | 控制写入数据的目标寄存器 |
| Branch | O | 是否为beq/bne/bgez/bgtz/blez/bltz指令 |
| MemtoReg | O | 控制写入寄存器数据的来源 |
| Alusrc1 | O | 控制进行运算的第一个操作数的数据来源 |
| Alusrc2 | O | 控制进行运算的第二个操作数的数据来源 |
| RegWrite | O | 能否写入数据到寄存器组中 |
| Jump[1:0] | O | 01：是j指令， 10：是jr指令  11：是jal指令，00：不是上述三种指令 |
| Extop | O | 控制立即数的扩展方式 |
| keep | O | 是否存在保留指令例外，为0时说明指令集里有当前指令 |
| MemWrite[1:0] | O | 01：sb指令的写内存方式  10：sh指令的写内存方式  11：sw指令的写内存方式  00：不写数据到内存 |
| MemRead[2:0] | O | 001：lb指令的读内存方式  101：lbu指令的读内存方式  010：lh指令的读内存方式  110：lhu指令的读内存方式  011：lw指令的读内存方式  000：不读内存中的数据 |
| ALUctr[4:0] | O | 控制alu进行何种运算 |
| mtc0 | O | 当前指令是否为mtc0指令  1：是， 0：否 |
| mfc0 | O | 当前指令是否为mfc0指令  1：是， 0：否 |
| mfhi | O | 当前指令是否为mfhi指令  1：是， 0：否 |
| mflo | O | 当前指令是否为mflo指令  1：是， 0：否 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 总控制 | 向各个模块发出控制信号 |
| 2 | 控制清零 | 当ctrl为1，或者当前指令是nop时，将控制信号全部置零 |

### 6. forward\_id模块

（1）基本描述

在id级解决R指令可能发生的数据冒险，输出选择信号在id模块中对数据进行正确地选择。主要处理写后读数据冒险和前前一条指令发生lw数据冒险。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| rs\_id[4:0],  rt\_id[4:0] | I | 当前指令对应的rs和rt寄存器 |
| rd\_exe[4:0] | I | exe级指令对应的rd寄存器 |
| rd\_mem[4:0] | I | mem级指令对应的rd寄存器 |
| RegWrite\_exe, | I | exe级指令是否会写回寄存器 |
| RegWrite\_mem | I | mem级指令是否会写回寄存器 |
| MemRead\_mem[2:0] | I | mem级指令是否是lw/lh/lb指令 |
| hd\_rs[1:0],  hd\_rt[1:0] | O | 用于解决rs和rt寄存器数据冒险的控制信号  00：没有发生数据冒险  01：前一条指令发生R指令数据冒险  10：前前一条指令发生R指令数据冒险  11：前前一条指令发生lw数据冒险 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 检测R指令的数据冒险 | 当前一条指令或前前一条指令与当前指令发生数据冒险时，输出一个正确的选择信号 |
| 2 | 检测lw的数据冒险 | 当前前一条指令是lw且发生数据冒险时，输出一个正确的选择信号（若lw是前一条指令，则在hazardDetect模块中进行阻塞） |

### 7. regfile模块

（1）基本描述

完成寄存器读取数据的操作以及将结果写回寄存器，由32个32位的寄存器组成。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| we | I | 控制器给出，1表示可以写数据到寄存器 |
| rs[4:0], | I | instruction[25:21],rs寄存器的地址 |
| rt[4:0], | I | instruction[20:16],rt寄存器的地址 |
| rd[4:0] | I | 目标寄存器的地址 |
| wd[31:0] | I | 待写入数据，从ALU计算结果和数据存储器的数据中二选一 |
| busA[31:0] | O | rs寄存器中的值 |
| busB[31:0] | O | rt寄存器中的值 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读寄存器 | 读取指定寄存器中的数据 |
| 2 | 写数据 | 将外来数据写入到指定的目标寄存器中 |

### 8. signext模块

（1）基本描述

对输入的16位立即数进行符扩展或零扩展。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| a[15:0] | I | 从指令中取出的16位立即数 |
| f | I | 控制信号，为0时进行零扩展，为1时进行符扩展 |
| y[31:0] | O | 16位立即数扩展成32位得到的结果 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 立即数扩展 | 根据控制信号f对立即数进行零扩展或符扩展 |

### 9. hazardDetect模块

（1）基本描述

根据读出的数据判断是否会发生控制冒险或load\_use数据冒险，若是，则向if级输出flush信号并输出对应的正确的跳转地址。主要处理load\_use数据冒险和beq、j指令的控制冒险。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| rs\_id[4:0],  rt\_id[4:0] | I | id级指令对应的rs和rt寄存器 |
| rt\_exe[4:0] | I | exe级指令对应的rt寄存器 |
| lw\_pre[2:0] | I | 前一条指令是否为读内存指令 |
| branch | I | 当前指令是否为分支指令 |
| jump[1:0] | I | 当前指令是否为跳转指令 |
| instruction\_id[31:0] | I | id级指令的值 |
| bpc[31:0] | I | beq对应的32位（由16位立即数扩展而来）跳转地址 |
| pc[31:0] | I | 当前指令的pc值 |
| a[31:0],  b[31:0] | I | 从寄存器读出且解决过数据冒险后的两个值 |
| PcWrite | O | if级的pc控制信号  1：pc更新， 0：pc不更新 |
| if\_idWrite | O | if\_id流水线寄存器的控制信号  1：寄存器正常写入， 0：不写入 |
| pc\_flush | O | pc模块的flush信号 |
| pc\_src | O | npc选择控制信号，  1：选择跳转地址， 0：选择pc+4 |
| ctrl | O | 当前指令是否为跳转指令（jal除外），1：是， 0：否 |
| id\_lw | O | 前一条指令是否为lw，且存在数据冒险 |
| if\_lw | O | if级指令的id\_lw |
| address\_out[31:0] | O | beq或j指令跳转的32位地址 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 检测load\_use数据冒险 | 发生load\_use数据冒险时，控制if级pc不变 |
| 2 | 检测beq控制冒险 | 当beq发生时，将正确的地址传送到if级（在trace对比机制中无需阻止下一条指令执行），阻止if\_id流水线寄存器的写入 |
| 3 | 检测j指令 | 当遇到j指令时，和beq一样处理 |
| 4 | 检测jal指令 | 当遇到jal指令时，只需将跳转地址送到if级，if\_id流水线寄存器正常写入 |

### 10. id\_exe模块

（1）基本描述

id级和exe级中间的流水线寄存器，用于保存前一级流水线的数据并输出至下一级。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| ctrl | I | 当前指令的ctrl |
| id\_flush | I | 控制id\_exe流水线寄存器正常写入 |
| id\_lw | I | 前一条指令是否为lw，且存在数据冒险 |
| RegDst,  Branch,  MemtoReg,  Alusrc1,  Alusrc2,  MemWrite[1:0],  MemRead[2:0],  RegWrite,  Aluctr[4:0]，  mtc0,  mfc0 | I | 当前指令对应的control模块的控制信号 |
| rt[4:0],  rd[4:0] | I | 当前指令对应的rt、rd寄存器 |
| immi1[31:0],  immi2[31:0] | I | 当前指令对应的扩展之后的立即数 |
| busA[31:0],  busB[31:0] | I | 处理过数据冒险后的两个数据 |
| pc\_4[31:0],  pc[31:0] | I | id级指令的pc和pc\_4 |
| RegDst\_out,  Branch\_out,  MemtoReg\_out,  Alusrc1\_out,  Alusrc2\_out,  MemWrite\_out[1:0],  MemRead\_out[2:0],  RegWrite\_out,  Aluctr\_out[4:0],  mtc0\_out,  mfc0\_out | O | 保存并输出当前指令对应的control模块的控制信号 |
| id\_lw\_out | O | 保存并输出当前指令对应的id\_lw |
| rt\_out[4:0],  rd\_out[4:0] | O | 保存并输出当前指令对应的rt、rd寄存器 |
| pc\_4\_out[31:0],  pc\_out[31:0] | O | 保存并输出id级指令的pc和pc\_4 |
| busA\_out[31:0],  busB\_out[31:0] | O | 保存并输出处理过数据冒险后的两个数据 |
| immi1\_out[31:0],  immi2\_out[31:0] | O | 保存并输出当前指令对应的扩展之后的立即数 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当reset为1或者id\_flush为1时，流水线寄存器清零 |
| 2 | 保存并输出前一级指令的数据 | 在每个clk的上升沿保存前一级的数据并输出 |

### 11. exe模块

（1）基本描述

根据id级传来的数据，选择立即数或者处理后的寄存器数据作为alu的输入数据，在alu模块中进行正确的运算。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| exe\_flush | I | 为1时将exe的输出结果清零 |
| RegDst,  Branch,  MemtoReg,  Alusrc1,  Alusrc2,  MemWrite[1:0],  MemRead[2:0],  RegWrite,  Aluctr[4:0],  mtc0,  mfc0 | I | 当前指令对应的control模块的控制信号 |
| rt[4:0],  rd[4:0] | I | 当前指令对应的rt、rd寄存器 |
| pc\_4[31:0] | I | 当前指令对应的pc\_4 |
| busA[31:0],  busB[31:0] | I | 处理过数据冒险后的两个数据 |
| immi1[31:0],  immi2[31:0] | I | 当前指令对应的扩展之后的立即数 |
| Branch\_out,  MemtoReg\_out,  MemWrite\_out[1:0],  MemRead\_out[2:0],  RegWrite\_out,  mtc0\_out,  mfc0\_out | O | 输出当前指令对应的部分control模块的控制信号 |
| Aluout[31:0] | O | alu运算结果 |
| busB\_out[31:0] | O | 输出busB |
| zero | O | alu的两个输入数据是否相等，  1:相等， 0：不相等 |
| overflow | O | 运算结果是否发生溢出异常  1：是， 0：否 |
| rd\_out[4:0] | O | 根据控制信号RegDst得到的目标寄存器 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | alu运算 | 根据输入选择正确的数据在alu模块中进行运算 |
| 2 | 输出目标寄存器 | 根据控制信号RegDst从rt和rd中选择目标寄存器 |

### 12. mux2模块

（1）基本描述

二路选择器，根据控制信号对输入的数据二选一。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| d0[31:0]  d1[31:0] | I | 两个输入数据 |
| s | I | 控制信号，为0时选择d0，为1时选择d1 |
| y[31:0] | O | 二路选择器的选择结果 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 二路选择器 | 根据控制信号对两个输入数据进行二选一输出 |

### 13. alu模块

（1）基本描述

alu的主要功能是完成数据运算，alu可以完成加、减、与、或等运算，同时可以判断两个进行运算的数据是否相等、运算结果是否发生溢出异常。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| op[4:0] | I | 控制alu执行何种运算 |
| a[31:0],  b[31:0] | I | 进行运算的两个数据 |
| zero | I | 判断进行运算的数据是否相等 |
| overflow | O | 判断是否发生溢出异常 |
| result[31:0] | O | 运算结果 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | alu运算 | 根据控制信号将两个输入数据进行相应运算 |
| 2 | 溢出判断 | 判断运算结果是否发生溢出异常 |

### 14. judge模块

（1）基本描述

根据memW和Aluout来得到dm的写使能信号data\_sram\_wen、要写入的数据data\_sram\_wdata和dm的读/写地址data\_sram\_addr。同时根据rd寄存器来实现从CP0的寄存器取值和向CP0的寄存器存值。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| memW[1:0] | I | 当前指令对应的memWrite控制信号 |
| Aluout[31:0] | I | 当前指令的alu运算结果 |
| busB[31:0] | I | 当前指令的busB |
| rd[4:0] | I | 当前指令的rd寄存器 |
| mtc0，  mfc0 | I | 当前指令对应的control模块的控制信号 |
| badAddr\_out[31:0] | I | CP0的BadVAddr寄存器中的值 |
| count\_out[31:0] | I | CP0的Count寄存器中的值 |
| status\_out[31:0] | I | CP0的Status寄存器中的值 |
| cause\_out[31:0] | I | CP0的Cause寄存器中的值 |
| epc\_out[31:0] | I | CP0的EPC寄存器中的值 |
| data\_sram\_wen[3:0] | O | dm写使能信号 |
| data\_sram\_wdata[31:0] | O | dm写入的数据 |
| data\_sram\_addr[31:0] | O | dm读/写的地址 |
| badAddr[31:0] | O | CP0的BadVAddr寄存器写入的值 |
| count[31:0] | O | CP0的Count寄存器写入的值 |
| status[31:0] | O | CP0的Status寄存器写入的值 |
| cause[31:0] | O | CP0的Cause寄存器写入的值 |
| epc[31:0] | O | CP0的EPC寄存器写入的值 |
| except\_data[31:0] | O | 根据rd从CP0的相应寄存器中取出的值 |
| alu\_forward[31:0] | O | exe级的alu计算结果 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 求出dm的写使能信号和写入的数据 | 根据memW和Aluout的最低两位来得到dm的写使能信号data\_sram\_wen和要写入的数据data\_sram\_wdata |
| 2 | CP0的寄存器的访存 | 根据rd寄存器来实现从CP0的寄存器取值和向CP0的寄存器存值 |

### 15. exe\_mem模块

（1）基本描述

exe级和mem级中间的流水线寄存器，用于保存前一级流水线的数据并输出至下一级。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| exe\_flush | I | 控制exe\_mem流水线寄存器正常写入 |
| Branch,  MemtoReg,  MemWrite[1:0],  MemRead[2:0],  RegWrite，  mfc0 | I | 当前指令对应的control模块的控制信号 |
| except\_data[31:0] | I | 从CP0的相应寄存器中取出的值 |
| Aluout[31:0] | I | 当前指令对应的alu运算结果 |
| busB[31:0] | I | 当前指令对应的busB |
| pc[31:0] | I | 当前指令对应的pc |
| zero | I | 当前指令对应的zero |
| rd[4:0] | I | 当前指令对应的目标寄存器 |
| Branch\_out,  MemtoReg\_out,  MemWrite\_out[1:0],  MemRead\_out[2:0],  RegWrite\_out,  mfc0\_out | O | 保存并输出当前指令对应的control模块的控制信号 |
| Aluout\_out[31:0] | O | 保存并输出alu运算结果 |
| busB\_out[31:0] | O | 保存并输出busB |
| pc\_out[31:0] | O | 保存并输出当前指令对应的pc |
| zero\_out | O | 保存并输出当前指令对应的zero |
| rd\_out[4:0] | O | 保存并输出当前指令对应的目标寄存器 |
| except\_data\_out[31:0] | O | 保存并输出从CP0的相应寄存器中取出的值 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当reset为1或者exe\_flush为1时，流水线寄存器清零 |
| 2 | 保存并输出前一级指令的数据 | 在每个clk的上升沿保存前一级的数据并输出 |

### 16. mem模块

（1）基本描述

根据alu运算的地址和传入的数据得到从内存中读到的最终数据。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| Branch,  MemtoReg,  RegWrite,  memR[2:0] | I | 当前指令对应的control模块的控制信号 |
| Aluout[31:0] | I | 当前指令对应的alu运算结果 |
| data\_sram\_rdata | I | dm读出的数据，32位 |
| zero | I | 当前指令对应的zero |
| rd[4:0] | I | 当前指令对应的目标寄存器 |
| br | O | br=Branch&zero |
| MemtoReg\_out  RegWrite\_out | O | 当前指令对应的control模块的控制信号 |
| Aluout\_out[31:0] | O | alu模块的运算结果 |
| rdata\_mem[31:0] | O | 读dm，最终得到的数据 |
| rd\_out[4:0] | O | 写回数据的目标寄存器 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 读取数据 | 根据Aluout、memR和data\_sram\_rdata得到从内存中读到的最终数据 |

### 17. mem\_wb模块

（1）基本描述

mem级和wb级中间的流水线寄存器，用于保存前一级流水线的数据并输出至下一级。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| MemtoReg，  RegWrite，  mfc0 | I | 当前指令对应的control模块的控制信号 |
| except\_data[31:0] | I | 从CP0的相应寄存器中取出的值 |
| Aluout[31:0] | I | 当前指令对应的alu运算结果 |
| pc[31:0] | I | 当前指令对应的pc |
| rdata[31:0] | I | 当前指令对应的从dm中读取的数据 |
| rd[4:0] | I | 当前指令对应的写回数据的目标寄存器 |
| MemtoReg\_out,  RegWrite\_out,  mfc0\_out | O | 保存并输出当前指令对应的control模块的控制信号 |
| Aluout\_out[31:0] | O | 保存并输出alu的计算结果 |
| pc\_out[31:0] | O | 保存并输出当前指令对应的pc |
| rdata\_out[31:0] | O | 保存并输出从dm中读取的数据 |
| rd\_out[4:0] | O | 保存并输出写回数据的目标寄存器 |
| except\_data\_out[31:0] | O | 保存并输出从CP0的相应寄存器中取出的值 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 复位 | 当reset为1时，流水线寄存器清零 |
| 2 | 保存并输出前一级指令的数据 | 在每个clk的上升沿保存前一级的数据并输出 |

### 18. wb模块

（1）基本描述

将前面经过alu运算的数据和读取的数据通过mfc0和memToReg信号进行选择后写回寄存器。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| MemtoReg，  RegWrite，  mfc0 | I | 当前指令对应的control模块的控制信号 |
| except\_data | I | 从CP0的相应寄存器中取出的值 |
| Aluout[31:0] | I | 当前指令对应的alu运算结果 |
| rdata[31:0] | I | 当前指令对应的从dm中读取的数据 |
| rd[4:0] | I | 当前指令对应的写回数据的目标寄存器 |
| RegWrite\_out | O | 控制写回寄存器的信号  1：写回， 0：不写回 |
| dataout[31:0] | O | 写回目标寄存器的数据 |
| rd\_out[4:0] | O | 写回数据的目标寄存器 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 数据写回 | 当RegWrite有效时，将rdata写回寄存器 |

### 19. CP0模块

（1）基本描述

协处理器模块，负责处理中断和异常。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| overflow | I | 运算结果是否发生溢出异常  1：是， 0：否 |
| keepinstr | I | 是否存在保留指令例外  1：是， 0：否 |
| mtc0 | I | 是否是mtc0指令  1：是， 0：否 |
| id\_pc[31:0] | I | id级指令对应的pc值 |
| exe\_pc[31:0] | I | exe级指令对应的pc值 |
| address\_in[31:0] | I | 错例外虚地址输入 |
| count\_in[31:0] | I | 计时器输入 |
| status\_in[31:0] | I | 处理器状态输入 |
| pc\_in[31:0] | I | 例外触发PC |
| cause\_in[31:0] | I | 例外触发原因输入 |
| bad\_address[31:0] | O | 错例外虚地址输出 |
| count\_out[31:0] | O | 计时器输出 |
| status\_out[31:0] | O | 处理器状态输出 |
| pc\_out[31:0] | O | 查询异常地址 |
| pc\_back[31:0] | O | 恢复异常地址 |
| cause\_back[31:0] | O | 例外触发原因输出 |
| if\_flush | O | if\_id寄存器的清零信号 |
| id\_flush | O | id\_exe寄存器的清零信号 |
| exe\_flush | O | exe\_mem寄存器的清零信号 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 解决异常 | 判断异常并控制系统做出反应 |

### 20. BadVAddr模块

（1）基本描述

一个寄存器，用于记录最近一次导致发生地址错例外的虚地址

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| write | I | 寄存器写使能 |
| address[31:0] | I | 错例外的虚地址输入 |
| address\_out[31:0] | O | 虚地址输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 保存地址 | 记录最近发生地址错例外的虚地址 |

### 21. Status模块

（1）基本描述

一个包含处理器状态信息的寄存器

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| write | I | 写使能信号 |
| status[31:0] | I | 状态输入 |
| ststus\_out[31:0] | O | 状态输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 保存状态 | 更新发生错例外后的处理器状态 |

### 22. Count模块

（1）基本描述

计时器寄存器

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| write | I | 写使能 |
| count\_in[31:0] | I | 计时器输入 |
| count[31:0] | O | 计时器输出 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 计时器 | 每两个时钟信号自增一次计数 |

### 23. epc模块

（1）基本描述

保存异常地址寄存器，用于恢复异常。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| write | I | 写使能信号 |
| pc[31:0] | I | 送入的异常地址 |
| pc\_out[31:0] | O | 恢复异常的地址 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 保存地址 | 保存发生异常时的指令地址 |

### 24. cause模块

（1）基本描述

保存异常原因寄存器，用于查询异常。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| write | I | 写使能信号 |
| reason[31:0] | I | 异常原因 |
| reason\_out[31:0] | O | 异常原因 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 保存异常原因 | 保存发生异常的原因等待查询 |

### 25. datapath模块

（1）基本描述

整个五级流水线处理器的数据通路，用来连接各个模块。

（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| reset | I | 复位信号  1：复位， 0：无效 |
| int | I | interruput |
| inst\_sram\_rdata[31:0] | I | im读出的数据 |
| inst\_sram\_en | I | im整体使能 |
| inst\_sram\_wen[3:0] | I | im写使能（常为0） |
| inst\_sram\_addr[31:0] | I | im指令地址 |
| inst\_sram\_wdata[31:0] | I | 写入im的数据 |
| data\_sram\_en | I | dm整体使能 |
| data\_sram\_wen[3:0] | I | dm写使能 |
| data\_sram\_addr[31:0] | I | dm数据地址 |
| data\_sram\_wdata[31:0] | I | 写入dm的数据 |
| data\_sram\_rdata[31:0] | I | dm读出的数据 |
| debug\_wb\_pc[31:0] | O | debug使用信号 |
| debug\_wb\_rf\_wen[3:0] | O | debug使用信号 |
| debug\_wb\_rf\_wnum[4:0] | O | debug使用信号 |
| debug\_wb\_rf\_wdata[31:0] | O | debug使用信号 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 数据通路 | 连接处理器的各个模块 |

### 26. mycpu\_top模块

（1）基本描述

总CPU仿真控制模块，下属datapath顶层模块。

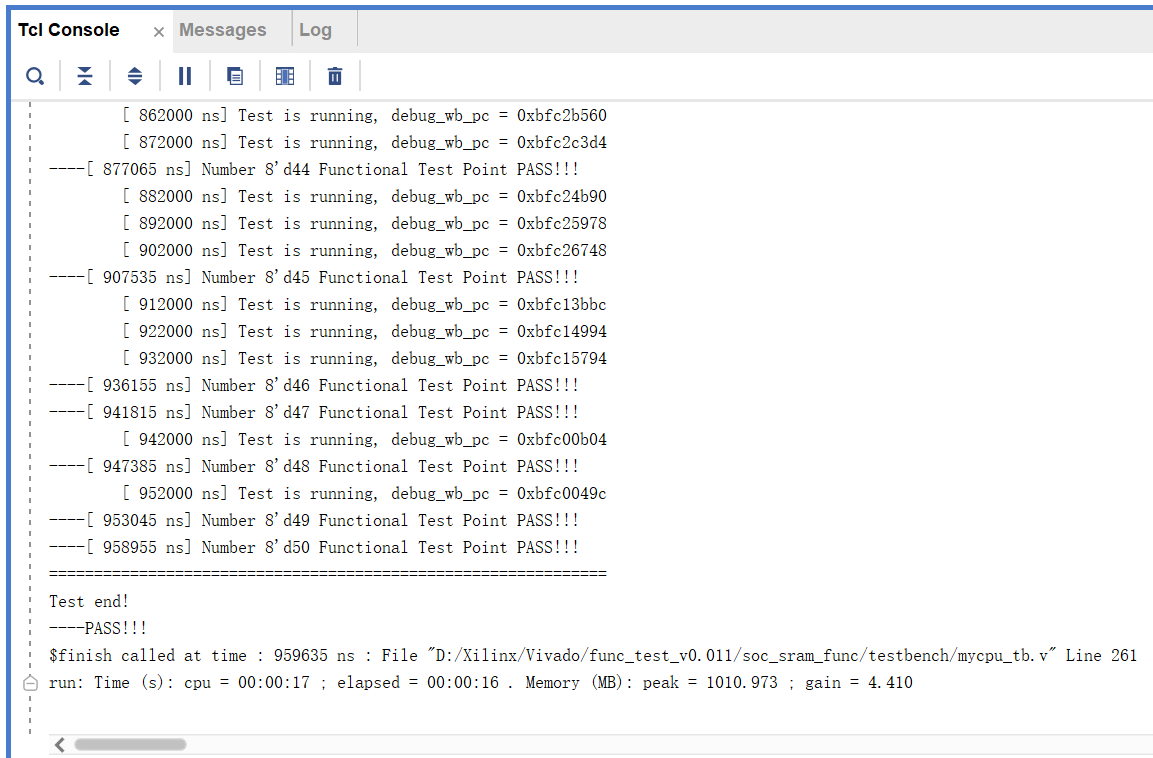
（2）模块接口

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| clk | I | 时钟信号 |
| resetn | I | 复位信号  0：复位， 1：无效 |
| int | I | interruput |
| inst\_sram\_rdata[31:0] | I | im读出的数据 |
| inst\_sram\_en | I | im整体使能 |
| inst\_sram\_wen[3:0] | I | im写使能（常为0） |
| inst\_sram\_addr[31:0] | I | im指令地址 |
| inst\_sram\_wdata[31:0] | I | 写入im的数据 |
| data\_sram\_en | I | dm整体使能 |
| data\_sram\_wen[3:0] | I | dm写使能 |
| data\_sram\_addr[31:0] | I | dm数据地址 |
| data\_sram\_wdata[31:0] | I | 写入dm的数据 |
| data\_sram\_rdata[31:0] | I | dm读出的数据 |
| debug\_wb\_pc[31:0] | O | debug使用信号 |
| debug\_wb\_rf\_wen[3:0] | O | debug使用信号 |
| debug\_wb\_rf\_wnum[4:0] | O | debug使用信号 |
| debug\_wb\_rf\_wdata[31:0] | O | debug使用信号 |

（3）功能定义

|  |  |  |
| --- | --- | --- |
| 序号 | 功能名称 | 功能描述 |
| 1 | 提供IM，DM接口 | 与IM，DM相连完成数据读写 |

# 四、测试代码及结果



# 五、实验时间安排

1.复习流水线知识点，阅读指令及调试相关资料——9h

2.将单周期处理器改为14条指令的初级流水线处理器（包括冒险），并进行测试——24h

3.分析中级流水线处理器新添加的指令，确定需新添加的模块，整理数据通路与控制指令——12h

4.熟悉trace调试，了解修改访存指令的处理方法——15h

5.确定最终的数据通路与控制指令，将模块进行重新编写——28h

6.debug以及进行调试——24h

7.书写实验报告——6h

共近118小时时间完成整个实验

# 六、心得体会

本次课设与之前的单周期处理器相比，进行了很大程度的拓展，对我们而言是非常大的考验。我们首先完成了初级的代码，测试通过后，接着往指令集中添加指令，最后利用trace对比机制进行仿真、调试。整个过程中，我们遇到了很多问题，不断地在修改代码，同时我们也学到了很多，分析和处理问题的能力越来越强。非常感谢我们小组的助教——郑好同学，不厌其烦地为我们解疑答惑。

以下是一些遇到的问题和相应的解决方法：

1.对于jal指令的处理方法，刚开始我们是再次调用regfile模块，将pc的值写入31号寄存器中，但是仿真中debug三个信号都不对，所以后来我们修改了jal指令产生的控制信号，使它能在wb段将pc的值写入31号寄存器。

2.dm的读写问题，我们添加了一个judge模块，放在了exe级，用来产生写使能信号和写入的数据，对于lb/lh/lw修改从内存读出数据的过程，我们放在了mem级。

3.一些移位指令用到了R型指令的shamt字段，因为我们的移位运算是在alu模块中进行的，所以我们在control模块中增加了一个控制信号，用来从rs的值和shamt的值中选一个送到alu的输入端。

4、对于冒险的处理和跳转操作我们都提前到了id段，解决了各种繁杂的数据矛盾，将其整合，保证各种情况下的数据冒险进行转发和停滞以及分支延迟槽都能够正常运行。同时对数据冒险的优先级进行了优化，保证不会相互冲突且转发的数据都时正确的。

5、在判断冒险和跳转时，进行了细小的延迟操作，并且使用下降沿周期触发，保证先判断load use数据冒险再判断跳转指令。同时又能够保证送来的信号能够正确判断，且不会因为判断结果而扰乱其他模块的正常进行。

6、在刚开始上trace对比机时，发现pc和指令总是岔开一个周期，经过多次调试发现要利用reset信号进行初始化，将各个模块以及im的读取进行限制，才能得到正确的运行流程。

7、在trace对比机上进行debug时，经历了非常多种多样的错误，提高了我们对vivado的使用能力，能够熟练的找到信号查看波形，对指令集进行对照查找后，判断错误出现的地方，解决了各种情况下的冒险和跳转以及内存的读写与判断问题。

经过反复修改与调试后，这些问题最终都得以解决，虽然过程充满坎坷，但看到结果的那一刻还是无比欣慰的。本次课设不仅锻炼了我们的学习能力，也提高了我们的耐心，获得了丰富的经验。希望未来我们能做的更好！