CPU设计文档

目录：

[一、工程化方法与流水线设计 1](#_Toc532340381)

[1.指导性原则 1](#_Toc532340382)

[2.设计说明 1](#_Toc532340383)

[3.设计要求 1](#_Toc532340384)

[二、模块设计（沿用单周期模块） 2](#_Toc532340385)

[1. IFU（取指令单元） 2](#_Toc532340386)

[2.NPC（计算下一PC） 3](#_Toc532340387)

[3.CMP（beq比较前移） 3](#_Toc532340388)

[4. GRF（寄存器堆） 3](#_Toc532340389)

[5. ALU（算术逻辑单元） 4](#_Toc532340390)

[6. DM（数据存储器） 4](#_Toc532340391)

[7. EXT（位扩展器） 5](#_Toc532340392)

[三、数据通路设计 6](#_Toc532340393)

[1.基础数据通路设计 6](#_Toc532340394)

[2.暂停机制控制 6](#_Toc532340395)

[3.转发机制控制 6](#_Toc532340396)

[4.设计数据通路 7](#_Toc532340397)

[四、控制器设计 9](#_Toc532340398)

[1.主控制器 9](#_Toc532340399)

[2.AT编码器 11](#_Toc532340400)

[3.冲突检测器 11](#_Toc532340401)

[4.转发控制器 12](#_Toc532340402)

[五、测试程序 13](#_Toc532340403)

[1.无分支/跳转测试 13](#_Toc532340404)

[2. 分支/跳转测试 13](#_Toc532340405)

[3. 暂停/转发机制测试 14](#_Toc532340406)

[六、思考题 16](#_Toc532340407)

一、工程化方法与流水线设计

1.指导性原则

（1）规范化命名：对于数据信号，明确指出信号名、阶段名、方向。

（2）模块化结构：把每一个流水级作为一个独立模块，每一级流水线寄存器也作为一个独立模块，数据只能在相邻模块之间传递。

（3）分布式译码**：**只把指令在流水级之间传递，然后在不同流水级分别实例化一个同样的控制模块进行译码。

2.设计说明

在上次实验的基础上，使用Verilog语言设计一个流水线处理器。

（1）支持MIPS-lite2指令集：{addu, subu, ori, lw, sw, beq, lui, j, jal, jr, nop}。

（2）addu,subu不支持溢出。

（3）处理器为流水线设计。

（4）考虑延迟槽。

3.设计要求

（1）顶层视图中将控制器与冲突单元分离，定义独立 controller 模块和 hazard 模块。控制器的设计与单周期没有差别。

（2）流水线的设计以追求性能为第一目标，尽最大可能支持转发以解决数据冒险。

（3）对于b类和j类指令，流水线设计必须支持延迟槽，因此设计需要注意使用 PC+8。

（4）为了解决数据冒险而设计的转发数据来源必须是某级流水线寄存器，不允许对功能部件的输出直接进行转发。

二、模块设计（沿用单周期模块）

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 |
| branch | I | 判断当前指令是需要跳转 |
| Br[1:0] | I | 判断分支/跳转指令的PC计算方式 |
| PC[31:0] | O | 当前PC的值 |
| PC4[31:0] | O | 当前PC+4的值 |
| PC8[31:0] | O | 当前PC+8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 |
| 3 | 计算PC4 | 计算PC+4的值并输出 |
| 4 | 计算PC8 | 计算PC+8的值并输出 |
| 5 | 更新PC | 通过branch确定下一条指令的PC |

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 |
| Instr[31:0] | I | 当前PC所指的指令 |
| 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 |
| 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 | 分支指令beq | 跳转，NPC = PC + 4 + {14{ Instr [15]}，Instr [15:0]，0，0} |
| 2 | 跳转指令jal | 跳转，NPC = {PC[31:28] ，Instr [25:0]，0，0} |
| 3 | 跳转指令jr | 跳转，NPC = rs |

3.CMP（beq比较前移）

比较两输入，相等时输出，不相等时输出0。

表2.3.1 CMP模块端口定义

|  |  |  |
| --- | --- | --- |
| 信号名 | 方向 | 描述 |
| A[31:0] | I | 第一个待比较值 |
| B[31:0] | I | 第二个待比较值 |
| C | O | 比较结果 |

4. 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2.4.1 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2.4.2 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 |

5. 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2.5.1 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2.5.2 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 |

6. 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2.6.1 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2.6.2 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 |

7. EXT（位扩展器）

表2.7.1 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2.7.2 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} |

三、数据通路设计

1.基础数据通路设计

（1）构造表头：表头的内容，主要包括各个阶段包含的部件、各个流水级寄存器以及它们输入的数据，依据它们来构造数据通路。在表头中，我们写出每一个功能模块以及这些功能模块的每个数据输入。对于RF我们把读功能和写功能分别写在D级和W级。

（2）分析指令，并填写表格：基础数据通路中暂且忽略流水线带来的各种冒险，只考虑数据从哪个部件流动到哪个部件，最后结果去向何处，最终写在表格中。由于数据不能跨越流水级寄存器进行传递，所以某个端口如果在这条指令的执行过程中不需要使用则留空。

（3）构造多选器：有些端口有着多个可能的数据来源，我们需要对这些输入端口前加入一个多选器，来控制数据来源的选择。

2.暂停机制控制

当判定需要暂停的时候，我们需要执行三个操作：

（1）冻结PC，不让PC的值改变——IFU增加PC写使能信号(!stall)。

（2）让ID/EX流水级寄存器清零——E级流水寄存器增加clr信号(stall)。

（3）冻结IF/ID流水级寄存器——D级寄存器增加写使能信号(!stall)。

3.转发机制控制

分析转发情况，构造转发多选器：每一个数据需求，它的正确数据，可能是之前从寄存器堆中读取来的值，也有可能是其他流水级尚未来得及写入寄存器堆的结果。

因此构造转发表格，分析冲突情况，构造转发多选器，把转发多选器加入数据通路中，控制信号由控制器输入。

表3.3.1 转发多选器

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **MUX** | **控制信号** | **输入来源** | | | | |
| MF\_RD1\_D | RD1\_D | RF.RD1 | PC8@M | AO@M | PC8@E |  |
| MF\_RD2\_D | RD2\_D | RF.RD2 | PC8@M | AO@M | PC8@E |  |
| MF\_RD1\_E | RD1\_E | RD1@E | WD |  | PC8@M | AO@M |
| MF\_RD2\_E | RD2\_E | RD2@E | WD |  | PC8@M | AO@M |
| MF\_RD2\_M | RD2\_M | RD2@M | WD |  |  |  |

当W级需要向D级转发时，由于GRF同时具有读写功能，可以在GRF模块内判断是否需要转发并实现转发，不通过datapath中的转发多选器实现。

4.设计数据通路

表3.4.1 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 |
| branch | I | 是否为分支/跳转指令的判断信号 |
| Br[1:0] | I | 分支/跳转地址计算方法的判断信号 |
| 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 |
| RES[2:0] | I | AT编码器计算出的Tnew信号 |
| stall | I | 冲突检测器传入暂停信号 |
| MRD1\_D[2:0] | I | 转发控制器传入MF\_RD1\_D的选择信号 |
| MRD2\_D[2:0] | I | 转发控制器传入MF\_RD2\_D的选择信号 |
| MRD1\_E[2:0] | I | 转发控制器传入MF\_RD1\_E的选择信号 |
| MRD2\_E[2:0] | I | 转发控制器传入MF\_RD2\_E的选择信号 |
| MRD2\_M[2:0] | I | 转发控制器传入MF\_RD2\_M的选择信号 |
| IR\_D[31:0] | O | D级指令 |
| IR\_E[31:0] | O | E级指令 |
| IR\_M[31:0] | O | M级指令 |
| IR\_W[31:0] | O | W级指令 |
| RES\_E[2:0] | O | E级RES |
| RES\_M2:0] | O | M级RES |
| RES\_W[2:0] | O | W级RES |
| A1\_E[4:0] | O | E级A1 |
| A2\_E[4:0] | O | E级A2 |
| A2\_M[4:0] | O | M级A2 |
| A3\_E[4:0] | O | E级A3 |
| A3\_M[4:0] | O | M级A3 |
| A3\_W[4:0] | O | W级A3 |

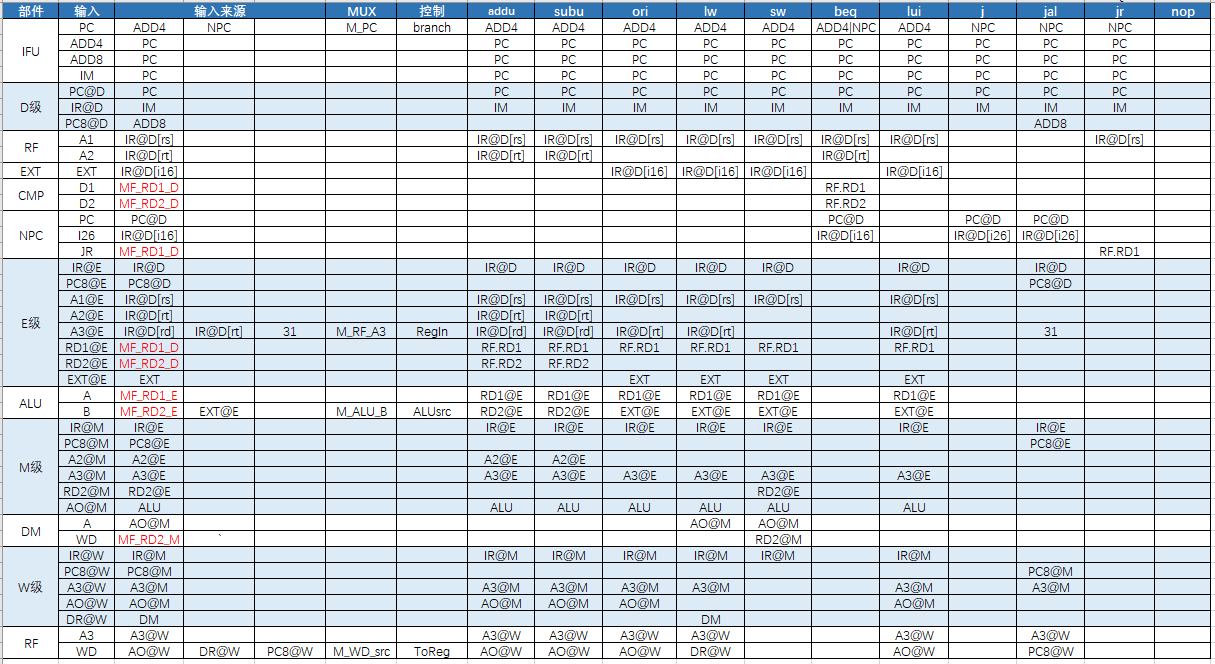


图3.1数据通路设计表格

四、控制器设计

1.主控制器

主控制器由指令生成相应控制信号，为精简流水寄存器，在每一级流水处设置一个主控制器，由本级指令产生本级所需控制信号。

表4.1.1 非转发暂停控制信号

|  |  |
| --- | --- |
| 控制信号 | 功能名称 |
| branch | branch = 0时，PC端输入为IFU.PC4；  branch = 1时，PC端输入为NPC。 |
| Br | Br = 00时，该指令为分支指令beq，NPC = PC + 4 + sign\_ext(Imm||00)。  Br = 01时，该指令为跳转指令jal或j，NPC = {PC[31:28] ，Jal，0，0}。  Br = 10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}；。 |
| ALUSrc | ALUSrc = 0时，ALU的第二个运算数来自转发复选器MF\_RD2\_E；  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； |
| 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4.1.2非转发暂停控制信号产生的真值表

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | opcode | funct | branch | Br | RegIn | RegWr | EXTOp | ALUSrc | ALUOp | MemWr | ToReg |
| addu | 000000 | 100001 | 0 | X | 01 | 1 | X | 0 | 00 | 0 | 00 |
| subu | 000000 | 100011 | 0 | X | 01 | 1 | X | 0 | 01 | 0 | 00 |
| ori | 001101 |  | 0 | X | 00 | 1 | 00 | 1 | 10 | 0 | 00 |
| lw | 100011 |  | 0 | X | 00 | 1 | 10 | 1 | 00 | 0 | 01 |
| sw | 101011 |  | 0 | X | X | 0 | 10 | 1 | 00 | 1 | X |
| beq | 000100 |  | 1 | 00 | X | 0 | X | 0 | X | 0 | X |
| lui | 001111 |  | 0 | X | 00 | 1 | 01 | 1 | 00 | 0 | 00 |
| nop | 000000 |  | 0 | X | 00 | 0 | 00 | 0 | 00 | 0 | 00 |
| j | 000010 |  | 1 | 01 | X | 0 | X | X | X | 0 | X |
| jal | 000011 |  | 1 | 01 | 10 | 1 | X | X | X | 0 | 10 |
| jr | 000000 | 001000 | 1 | 10 | 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 |
| j | 000010 instr\_index | PC = PC[31:28] || instr\_index || 00 |
| jr | 000000 rs 10{0} 5{0} 001000 | PC = GPR[rs] |
| jal | 000011 instr\_index | PC = PC[31:28] || instr\_index || 00  GPR[31] = PC + 4 |
| nop | 0x00000000 |  |

2.AT编码器

根据指令变量与，产生Tuse变量和Res编码值（Res是Tnew的变形，用于指定产生新值的功能部件）。

此处将指令分为Cal\_r类(addu, subu)、Cal\_i类(ori)、Beq类(beq)、Load类(lw)、Save类(sw)分类处理，J类的三条指令(j, jal, jr)分开处理。

表4.2.1 指令集的Tuse 表4.2.2 指令集的Tnew

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **指令** | **Tuse** | |  | **指令** | **功能部件** | **Tnew** | | |
| **rs** | **rt** |  | **E** | **M** | **W** |
| Cal\_r类 | 1 | 1 |  | Cal\_r类 | ALU | 1 | 0 | 0 |
| Cal\_i类 | 1 |  |  | Cal\_i类 | ALU | 1 | 0 | 0 |
| load类 | 1 |  |  | load类 | DM | 2 | 1 | 0 |
| store类 | 1 | 2 |  | store类 |  |  |  |  |
| beq | 0 | 0 |  | beq |  |  |  |  |
| lui |  |  |  | lui | ALU | 1 | 0 | 0 |
| nop |  |  |  | nop |  |  |  |  |
| j |  |  |  | j |  |  |  |  |
| jal |  |  |  | jal | PC | 0 | 0 | 0 |
| jr | 0 |  |  | jr |  |  |  |  |

3.冲突检测器

根据策略矩阵计算分类指令的暂停条件，根据条件按照逻辑取与和或运算。

表4.3.1 rs策略矩阵

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tnew**  **Tuse** | **E** | | | **M** | | | **W** | | |
| **ALU** | **DM** | **PC** | **ALU** | **DM** | **PC** | **ALU** | **DM** | **PC** |
| 1 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | **S** | **S** | F | F | **S** | F | F | F | F |
| 1 | F | **S** | F | F | F | F | F | F | F |

表4.3.2 rt策略矩阵

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Tnew**  **Tuse** | **E** | | | **M** | | | **W** | | |
| **ALU** | **DM** | **PC** | **ALU** | **DM** | **PC** | **ALU** | **DM** | **PC** |
| 1 | 2 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | **S** | **S** | F | F | **S** | F | F | F | F |
| 1 | F | **S** | F | F | F | F | F | F | F |
| 2 | F | F | F | F | F | F | F | F | F |

4.转发控制器

根据转发多选器需求和原则产生转发多选器的控制信号，控制信号的优先级由顺序决定，通过三目运算符的比较赋值顺序实现。

转发条件：转发接受级读寄存器编号与被转发级写寄存器编号相同不为0，转发级寄存器写使能信号为1。

转发来源：RES所标记的被转发级指令对应的新值产生部件储存在流水寄存器中的值。

表4.4.1 转发控制信号

|  |  |  |
| --- | --- | --- |
| **控制信号** | **取值** | **功能说明** |
| MF\_RD1\_D | MF\_PC8\_E | 向D级需要RD1值的端口转发PC8\_E |
| MF\_PC8\_M | 向D级需要RD1值的端口转发PC8\_M |
| MF\_AO\_M | 向D级需要RD1值的端口转发AO\_M |
| MF\_RD | D级需要RD1值的端口无需转发 |
| MF\_RD2\_D | MF\_PC8\_E | 向D级需要RD2值的端口转发PC8\_E |
| MF\_PC8\_M | 向D级需要RD2值的端口转发PC8\_M |
| MF\_AO\_M | 向D级需要RD2值的端口转发AO\_M |
| MF\_RD | D级需要RD2值的端口无需转发 |
| MF\_RD1\_E | MF\_PC8\_M | 向E级需要RD1值的端口转发PC8\_M |
| MF\_AO\_M | 向E级需要RD1值的端口转发AO\_M |
| MF\_WD\_W | 向E级需要RD1值的端口转发W级的WD值 |
| MF\_RD | E级需要RD1值的端口无需转发 |
| MF\_RD2\_E | MF\_PC8\_M | 向E级需要RD2值的端口转发PC8\_M |
| MF\_AO\_M | 向E级需要RD2值的端口转发AO\_M |
| MF\_WD\_W | 向E级需要RD2值的端口转发W级的WD值 |
| MF\_RD | E级需要RD2值的端口无需转发 |
| MF\_RD2\_M | MF\_WD\_W | 向M级需要RD2值的端口转发W级的WD值 |
| MF\_RD | M级需要RD2值的端口无需转发 |

五、测试程序

1.无分支/跳转测试

表5.1 CPU测试程序与期望输出（1）

|  |  |
| --- | --- |
| **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 ori $t4,4 | 00003010: $12 <= 00000004 |
| 6 sw $t3,0($t4) | 00003014: \*00000004 <= fffe0001 |
| 7 lw $t5,0($t4) | 00003018: $13 <= fffe0001 |

2. 分支/跳转测试

表5.2 CPU测试程序与期望输出（2）

|  |  |
| --- | --- |
| **测试程序** | **期望输出或现象** |
| 1 ori $t0,$t0,0xffff | 00003000: $ 8 <= 0000ffff |
| 2 beq $zero,$zero,yes | 跳转至第6行 |
| 3 nop | nop |
| 4 lui $t1,0xffff | 不执行 |
| 5 *yes：* |  |
| 6 beq $t0,$zero,noo | 不跳转 |
| 7 nop | nop |
| 8 ori $t1,$t1,0x1 | 00003018: $ 9 <= 00000001 |
| 9 *noo：* |  |
| 10 ori $t2,$t1,0xffff | 0000301c: $10 <= 0000ffff |

表5.3 CPU测试程序与期望输出（3）

|  |  |
| --- | --- |
| **测试程序** | **期望输出或现象** |
| 1 ori $t0,$t0,0xffff | 00003000: $ 8 <= 0000ffff |
| 2 jal jjal | 00003004: $31 <= 0000300c，跳转至8行 |
| 3 nop | nop |
| 4 lui $t1,0xffff | 0000300c: $ 9 <= ffff0000 |
| 5 j jjj | 跳转至13行 |
| 6 nop | nop |
| 7 *jjal:* |  |
| 8 addu $t2,$t0,$t1 | 00003018: $10 <= 0000ffff |
| 9 subu $t3,$t1,$t0 | 0000301c: $11 <= ffff0001 |
| 10 jr $ra | 跳转至第4行 |
| 11 nop | nop |
| 12 *jjj:* |  |
| 13 ori $t4,4 | 00003028: $12 <= 00000004 |

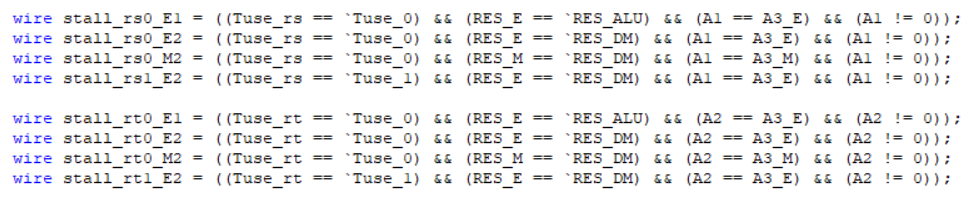
3. 暂停/转发机制测试

表5.4 CPU测试程序与期望输出（4）

|  |  |  |  |
| --- | --- | --- | --- |
| **序号** | **测试目的** | **测试程序** | **期望输出或现象** |
| 1 | rs0\_E1  rt0\_E1  MF\_RD1\_D  MF\_RD2\_D | ori $t0,0xffff  beq $t0,$t0,hhh  ……  *hhh:* | beq进入D后暂停1次，跳转  从M向D转发A0\_M给CMP\_A  从M向D转发A0\_M给CMP\_B， |
| 2 | rs0\_E2  rs0\_M2  rt0\_E2  rt0\_M2  MF\_RD1\_E  MF\_RD2\_M | ori $t0,0xffff  sw $t0,4($zero)  lw $t1,4($zero)  beq $t1,$t1,hhh  ……  *hhh：* | beq进入D后暂停2次，不跳转  从M向E转发WD给ALU\_B  从W向M转发WD给DM\_A |
| 3 | rs1\_E2  rt1\_E2  MF\_RD1\_E  MF\_RD2\_E | ori $t0,0xffff  sw $t0,4($zero)  lw $t1,4($zero)  addu $t2,$t1,$t1 | addu进入D后暂停1次  从W向E转发WD给ALU\_A  从W向E转发WD给ALU\_B |
| 4 | MF\_RD1\_D  MF\_RD2\_D | jal hhh  nop  *hhh:*  beq $ra,$ra,emm | 从M向D转发PC8\_M给CMP\_A  从M向D转发PC8\_M给CMP\_B |
| 5 | MF\_RD1\_E  MF\_RD2\_E  GRF内部转发W-D | ori $t0,0xffff  addu $t1,$t0,$t0  nop  addu $t2,$t0,$t0 | 从M向E转发AO\_M给ALU\_A和ALU\_B，GRF内部转发输出$t0 |
| 6 | MF\_RD1\_D  MF\_RD2\_D  MF\_RD1\_E  MF\_RD2\_E | jal hhh  addu $t0,$ra,$ra  ……  *hhh:* | 从E向D转发PC8\_E给RD1  从E向D转发PC8\_E给RD2  从M向E转发PC8\_M给ALU\_A  从M向E转发PC8\_M给ALU\_B |

六、思考题

Q： 在本实验中你遇到了哪些不同指令组合产生的冲突？你又是如何解决的？相应的测试样例是什么样的？请有条理的罗列出来。(非常重要)

A：按照Tuse和RES划分指令，分别有暂停情况8种、转发情况20种：

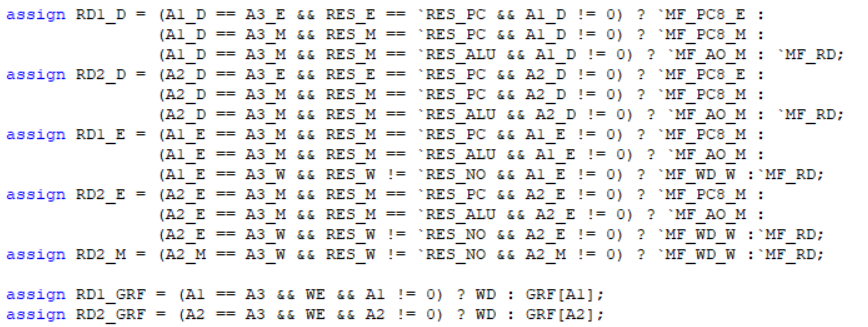
图6.1暂停情况

图6.2转发情况

测试样例见第五部分测试程序与期望输出，由于一些冲突会同时发正在一条指令中，因此实际测试样例较为精简。