

**计算机组成原理Project6实验报告**

Verilog – 支持50指令流水线CPU

{add,addu,and,div,divu,mult,multu,nor,or,sub,subu,xor,

addi,addiu,andi,lui,ori,xori,

beq,blez,bltz,bgez,bgtz,bne,

slt,sltu,slti,sltiu,sll,sllv,sra,srav,srl,srlv,

lw,lh,lhu,lb,lbu,sw,sh,sb,jal,jr,j,jalr,mthi,mtlo,mfhi,mflo}.

北京航空航天大学

计算机学院

陈麒先

16061160

二○一七年十二月

**郑重声明**

**关于诚实守信公约：**

**本实验报告由本人独立完成，全部内容均为本人通过查找互联网资料、翻阅课件、课堂笔记和教材后独立思考的结果。特此声明。**

**16061160**

**陈麒先**

**原创性声明**

**作业中出现的公式、图片、代码段以及图片的文字注释信息，均为作者原创。抄袭行为在任何情况下都被严格禁止(COPY is strictly prohibited under any circumstances)！转载或引用须征得作者本人同意，并注明出处！**

**16061160**

**陈麒先**

**目录**

[**第一章** **设计架构** 2](#_Toc498883720)



[第一节 流水线CPU顶层架构视图 2](#_Toc498883721)

[第二节 流水线CPU模块定义说明 2](#_Toc498883729)

[第三节 流水线CPU控制单元设计 8](#_Toc498883735)

[第四节 流水线CPU数据通路构造 11](#_Toc498883735)

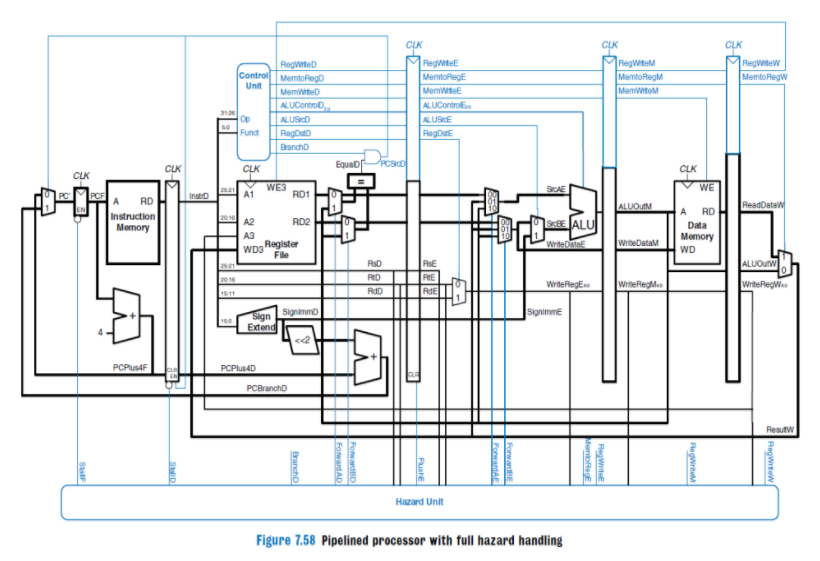
[第五节 流水线CPU转发暂停机制 13](#_Toc498883735)

[**第二章 测试验证** 13](#_Toc498883736)

[**第三章 思考题** 33](#_Toc498883736)

1. **设计架构**

第一节 流水线 CPU 顶层架构视图(部分)



第二节 流水线 CPU 模块定义说明

**1、IFU**

**（1）基本要求**

•起始地址：0x00003000。

•在实现设计中，为了保持模块的独立性，将IFU模块分成IM、PC、NPC三个模块的组合。其中IM模块单独负责取指令，其输入为32位PC，输出为32位指令码；PC单独建模寄存器，用于在clk上升沿更新为NPC的值；NPC通过控制信号的输入，实现对NPC的计算。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| clk | In | 时钟信号 |
| Reset | In | 异步复位信号 |
| if\_beq | In | B类跳转使能信号 |
| if\_jr | In | jr跳转使能信号 |
| if\_j | In | j跳转使能信号 |
| If\_jal | In | jal跳转使能信号 |
| zero | In | 相等条件信号 |
| Offset | In | 地址偏移量 |
| Instr | Out | 输出32位指令码 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 同步复位 | Reset信号有效时，PC复位 |
| 2 | b类跳转 | b类信号和条件信号同时有效时，执行跳转 |
| 3 | j类跳转 | j类信号有效时，执行跳转 |
| 4 | jr类跳转 | jr信号有效时，执行跳转至jr\_PC |
| 5 | PC4 | PC决定所取指令的地址，每个周期PC+4 |

**2、GPR**

**（1）基本要求**

•用具有写使能的寄存器实现，寄存器总数为 32 个。

•0 号寄存器的值保持为 0。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| clk | In | 时钟信号 |
| reset | In | 异步复位信号 |
| RA | [4:0] In | 读总线A地址 |
| RB | [4:0] In | 读总线B地址 |
| RW | [4:0] In | 写寄存器地址 |
| WD | [31:0] In | 写数据输入 |
| WE | In | 写使能 |
| BusA | [31:0] Out | 总线A输出 |
| BusB | [31:0] Out | 总线B输出 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 同步复位 | Reset信号有效时，GRF复位 |
| 2 | 读寄存器 | 根据RA，RB所指示的地址，读出对应寄存器的值 |
| 3 | 写寄存器 | 根据RW所指示的地址，将WD的数据写入对应寄存器 |
| 4 | 0号寄存器 | 0号寄存器不连接数据写入端口，输出接地 |

**3、ALU**

**（1）基本要求**

•提供 32 位加、减、或运算及大小比较功能。

•可以不支持溢出（不检测溢出）。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| A | [31:0] In | ALU 第一个操作数 |
| B | [31:1] In | ALU 第二个操作数 |
| ALUOp | [1:0] In | ALU 控制信号 |
| ALUOut | [31:0] Out | ALU 计算结果 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 加 | ALUOp = 00,两个操作数相加 |
| 2 | 减 | ALUOp = 01,两个操作数相减 |
| 3 | 或 | ALUOp = 10,两个操作数按位或 |

**4、DM**

**（1）基本要求**

•容量为 32bit \* 1024。

•起始地址：0x00000000。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| clk | In | 时钟信号 |
| reset | In | 异步复位信号 |
| WE | In | 写使能信号 |
| WD | In | 写入数据 |
| address | In | 写入地址 |
| RD | Out | 读内存数据 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 同步复位 | Reset信号有效时，DM复位 |
| 2 | 读内存数据 | WE信号为0时，读内存中Address指示的地址数据 |
| 3 | 写内存数据 | WE信号为1时，向内存中Address指示的地址写数据 |

**5、EXT：**

**（1）基本要求**

•注意扩展方式

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| Imm16 | [15:0] In | 输入待扩展的16位立即数 |
| ExtOp | [1:0] In | 扩展信号 |
| Ext32 | [31:0] Out | 扩展结果输出 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 零扩展 | 当ExtOp==2’b00时，执行零扩展 |
| 2 | 符号扩展 | 当ExtOp==2’b01时，执行符号扩展 |
| 3 | 高位扩展 | 当ExtOp==2’b10时，执行高位扩展 |

**6、CMP：**

**（1）基本要求**

•作为跳转类指令的条件判断信号，需前移至F级。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| A | [31:0] In | 比较数A |
| B | [31:0] In | 比较数B |
| equal | Out | 相等比较结果 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 相等比较 | 当equal输出高电平时，说明两输入操作数相等 |

**7、MD：**

**（1）基本要求**

•要求实现乘除运算，并能够模拟硬件设备中的延迟。

•内置hi/lo寄存器，并能对其进行读写。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| A | [31:0] In | 第一个操作数 |
| B | [31:0] In | 第二个操作数 |
| WD | [31:0] In | 写寄存器数据 |
| clk | In | 时钟信号 |
| reset | In | 复位信号 |
| mthi | In | 写hi寄存器使能 |
| mtlo | In | 写lo寄存器使能 |
| start | In | 発動信号 |
| MDOp | [1:0] In | 运算操作控制信号 |
| busy | Out | 繁忙信号 |
| hi | [31:0] Out | 高32位寄存器 |
| lo | [31:0] Out | 低32位寄存器 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 异步复位 | Reset信号有效时，MD复位 |
| 2 | 有无符号乘除运算 | 根据MDOp的输入值选择进行有无符号乘除运算 |
| 3 | 写hi/lo寄存器 | 根据hi/lo寄存器写使能信号将WD写入 |
| 4 | 延迟处理 | 输出busy信号模拟乘除运算单元延迟 |

**8、DMEXT：**

**（1）基本要求**

•对DM读出的数据进行扩展。

**（2）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| A | [1:0] In | 输入DM的地址低2位 |
| Op | [2:0] In | 扩展操作控制符 |
| Din | [31:0] In | 输入扩展单元数据（从DM获得） |
| Dout | [31:0] Out | 输出扩展结果待写回 |

**（3）功能说明**

|  |  |  |
| --- | --- | --- |
| 序号 | 功能 | 描述 |
| 1 | 零扩展半字 | 根据选择的部分进行零扩展半字 |
| 2 | 符号扩展半字 | 根据选择的部分进行符号扩展半字 |
| 3 | 零扩展字节 | 根据选择的部分进行零扩展字节 |
| 4 | 符号扩展字节 | 根据选择的部分进行符号扩展字节 |

第三节 流水线 CPU 控制单元设计

**（1）端口定义**

|  |  |  |
| --- | --- | --- |
| 端口名 | 方向 | 描述 |
| OpCode | [5:0] In | 指令高六位 Ins [31:26] |
| Func | [5:1] In | 指令低六位 Ins [5:0] |
| RegDst | Out | 若为高电平，选择Rd，否则选择Rt作为GPR的RW输入 |
| ALUSrc | Out | 若为高电平，选择扩展数字，否则选择BusB作为ALU的第二位输入 |
| MemToReg | Out | 若为高电平，选择DM，否则选择ALUOut作为GPR的WD输入 |
| RegWrite | Out | 若为高电平，则对GPR进行写操作，否则写使能无效 |
| MemWrite | Out | 若为高电平，则对DM进行写操作，否则对DM进行读操作 |
| If\_beq | Out | 若为高电平且zero为高电平，则跳转 |
| ExtOp[0] | Out | 共同决定EXT的行为 |
| ExtOp[1] | Out |
| ALUOp[0] | Out | 共同决定ALU的行为 |
| ALUOp[1] | Out |

**（2）控制器真值表**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| cal-r | add | addu | and | nor | or | sub | subu | xor |
| OpCode | 000000 | | | | | | | |
| Func | 100000 | 100001 | 100100 | 100111 | 100101 | 100010 | 100011 | 100110 |
| ALUSrc | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| RegWrite | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| MemToReg | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| EXTOp[1] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| EXTOp[0] | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| RegDst | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| ALUOp[2] | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| ALUOp[1] | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| ALUOp[0] | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| cal-i | addi | addiu | andi | lui | ori | xori |
| OpCode | 001000 | 001001 | 001100 | 001111 | 001101 | 001110 |
| Func | N/A | | | | | |
| ALUSrc | 1 | 1 | 1 | 1 | 1 | 1 |
| MemWrite | 0 | 0 | 0 | 0 | 0 | 0 |
| RegWrite | 1 | 1 | 1 | 1 | 1 | 1 |
| MemToReg | 0 | 0 | 0 | 0 | 0 | 0 |
| EXTOp[1] | 0 | 0 | 0 | 1 | 0 | 0 |
| EXTOp[0] | 1 | 1 | 0 | 0 | 0 | 0 |
| RegDst | 0 | 0 | 0 | 0 | 0 | 0 |
| ALUOp[2] | 0 | 0 | 0 | 0 | 0 | 1 |
| ALUOp[1] | 0 | 0 | 1 | 0 | 1 | 0 |
| ALUOp[0] | 0 | 0 | 1 | 0 | 0 | 0 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| branch | beq | bgez | bgtz | blez | bltz | bne |
| OpCode | 000100 | 000001 | 000111 | 000110 | 000001 | 000101 |
| Func | N/A | | | | | |
| Func2 | N/A | 00001 | N/A | N/A | 00000 | N/A |
| EXTOp[1] | 0 | 0 | 0 | 0 | 0 | 0 |
| EXTOp[0] | 1 | 1 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| jump | j | jal | jalr | jr |
| OpCode | 000010 | 000011 | 000000 | 000000 |
| Func | N/A | | 001001 | 001000 |
| RegWrite | 0 | 1 | 1 | 0 |
| set | slt | slti | sltiu | sltu |
| OpCode | 000000 | 001010 | 001011 | 000000 |
| Func | 101010 | N/A | | 101011 |
| ALUSrc | 0 | 1 | 1 | 0 |
| MemWrite | 0 | 0 | 0 | 0 |
| RegWrite | 1 | 1 | 1 | 1 |
| MemToReg | 0 | 0 | 0 | 0 |
| EXTOp[1] | 0 | 0 | 0 | 0 |
| EXTOp[0] | 0 | 1 | 1 | 0 |
| RegDst | 1 | 0 | 0 | 1 |
| ALUOp[2] | 1 | 1 | 1 | 1 |
| ALUOp[1] | 1 | 1 | 1 | 1 |
| ALUOp[0] | 0 | 0 | 1 | 1 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| shift | sll | sllv | sra | srav | srl | srlv |
| OpCode | 000000 | 000000 | 000000 | 000000 | 000000 | 000000 |
| Func | 000000 | 000100 | 000011 | 000111 | 000010 | 000110 |
| RegWrite | 1 | 1 | 1 | 1 | 1 | 1 |
| RegDst | 1 | 1 | 1 | 1 | 1 | 1 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| load | lw | lh | lhu | lb | lbu |
| OpCode | 100011 | 100001 | 100101 | 100000 | 100100 |
| Func | N/A | | | | |
| ALUSrc | 1 | 1 | 1 | 1 | 1 |
| MemWrite | 0 | 0 | 0 | 0 | 0 |
| RegWrite | 1 | 1 | 1 | 1 | 1 |
| MemToReg | 1 | 1 | 1 | 1 | 1 |
| EXTOp[1] | 0 | 0 | 0 | 0 | 0 |
| EXTOp[0] | 1 | 1 | 1 | 1 | 1 |
| RegDst | 0 | 0 | 0 | 0 | 0 |
| ALUOp[2] | 0 | 0 | 0 | 0 | 0 |
| ALUOp[1] | 0 | 0 | 0 | 0 | 0 |
| ALUOp[0] | 0 | 0 | 0 | 0 | 0 |
| DMEXTOp[2] | 0 | 1 | 0 | 0 | 0 |
| DMEXTOP[1] | 0 | 0 | 1 | 1 | 0 |
| DMEXTOp[0] | 0 | 0 | 1 | 0 | 1 |

|  |  |  |  |
| --- | --- | --- | --- |
| store | sw | sh | sb |
| OpCode | 101011 | 101001 | 101000 |
| Func | N/A | | |
| ALUSrc | 1 | 1 | 1 |
| MemWrite | 1 | 1 | 1 |
| RegWrite | 0 | 0 | 0 |
| MemToReg | 0 | 0 | 0 |
| EXTOp[1] | 0 | 0 | 0 |
| EXTOp[0] | 1 | 1 | 1 |
| RegDst | 0 | 0 | 0 |

第四节 数据通路构造表(部分)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | | | addu | subu | ori | lui | lw | sw |
| F级 | pc | | pc4 | pc4 | pc4 | pc4 | pc4 | pc4 |
| im | | pc | pc | pc | pc | pc | pc |
| pc4 | | pc | pc | pc | pc | pc | pc |
| F-D | D-Reg | IR@D | im | im | im | im | im | im |
| PC4@D | pc4 | pc4 | pc4 | pc4 | pc4 | pc4 |
| D级 | GPR | RA | IR@D[Rs] | IR@D[Rs] | IR@D[Rs] | IR@D[Rs] | IR@D[Rs] | IR@D[Rs] |
| RB | IR@D[Rt] | IR@D[Rt] |  |  |  | IR@D[Rt] |
| EXT | |  |  | IR@D[IMM] | IR@D[IMM] | IR@D[IMM] | IR@D[IMM] |
| NPC | pc4 |  |  |  |  |  |  |
| jr\_pc |  |  |  |  |  |  |
| imm |  |  |  |  |  |  |
| IR |  |  |  |  |  |  |
| CMP | D1 |  |  |  |  |  |  |
| D2 |  |  |  |  |  |  |
| pc8 | |  |  |  |  |  |  |
| D-E | E-Reg | IR@E | IR@D | IR@D | IR@D | IR@D | IR@D | IR@D |
| PC8@E |  |  |  |  |  |  |
| RA@E | GPR.RA | GPR.RA | GPR.RA | GPR.RA | GPR.RA | GPR.RA |
| RB@E | GPR.RB | GPR.RB |  |  |  | GPR.RB |
| ext@E |  |  | EXT | EXT | EXT | EXT |
| E级 | ALU | A | RA@E | RA@E | RA@E | RA@E | RA@E | RA@E |
| B | RB@E | RB@E | ext@E | ext@E | ext@E | ext@E |
| N/A | | | | | |
| E-M | M-Reg | IR@M | IR@E | IR@E | IR@E | IR@E | IR@E | IR@E |
| PC8@M |  |  |  |  |  |  |
| AO@M | ALU | ALU | ALU | ALU | ALU | ALU |
| WM@M |  |  |  |  |  | RB@E |
| M级 | DM | add |  |  |  |  | AO@M | AO@M |
| WM |  |  |  |  |  | WM@M |
| M-W | W-Reg | IR@W | IR@E | IR@E | IR@E | IR@E | IR@E | IR@E |
| PC8@W |  |  |  |  |  |  |
| AO@W | AO@E | AO@E | AO@E | AO@E |  |  |
| DO@W |  |  |  |  | DM |  |
| W级 | GPR | RW | IR@W[Rd] | IR@W[Rd] | IR@W[Rt] | IR@W[Rt] | IR@W[Rt] |  |
| WD | AO@W | AO@W | AO@W | AO@W | DO@W |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 指令 | beq | jal | jr | j | MUX | 0 | 1 | 2 |
| F级 | pc4|npc | npc | npc | npc | pc\_in | pc4@F | NPC |  |
| pc | pc | pc | pc | pc |  |  |  |
| pc | pc | pc | pc | pc |  |  |  |
| F-D | im | im | im | im | im |  |  |  |
| pc4 | pc4 | pc4 | pc4 | pc4 |  |  |  |
| D级 | IR@D[Rs] |  | IR@D[Rs] |  | IR@D[Rs] |  |  |  |
| IR@D[Rt] |  |  |  | IR@D[Rt] |  |  |  |
| IR@D[IMM] |  |  |  | IR@D[IMM] |  |  |  |
| PC4@D | PC4@D |  | PC4@D | PC4@D |  |  |  |
|  |  | GPR.RA |  | MFAD | GPR.RA | F1 | F2 |
| EXT |  |  |  | EXT |  |  |  |
|  | IR@D |  | IR@D | IR@D |  |  |  |
| IR@D[Rs] |  |  |  | MFAD |  |  |  |
| IR@D[Rt] |  |  |  | MFBD |  |  |  |
|  | PC4@D |  |  | PC4@D |  |  |  |
| D-E | IR@D | IR@D | IR@D | IR@D | IR@D |  |  |  |
|  | pc8 |  |  | pc8 |  |  |  |
|  |  |  |  | GPR.RA |  |  |  |
|  |  |  |  | GPR.RB |  |  |  |
|  |  |  |  | EXT |  |  |  |
| E级 |  |  |  |  | MFAE | RA@E | WD@W | AO@M |
|  |  |  |  | ALUB | MFBE | ext@E |  |
| N/A |  |  |  | MFBE | RB@E | WD@W | AO@M |
| E-M | IR@E | IR@E | IR@E | IR@E | IR@E |  |  |  |
|  | PC8@E |  |  | PC8@E |  |  |  |
|  |  |  |  | ALU |  |  |  |
|  |  |  |  | RB@E |  |  |  |
| M级 |  |  |  |  | AO@M |  |  |  |
|  |  |  |  | MFBM | WM@M | WD@W |  |
| M-W | IR@E | IR@E | IR@E | IR@E | IR@E |  |  |  |
|  | PC8@M |  |  | PC8@M |  |  |  |
|  |  |  |  | AO@E |  |  |  |
|  |  |  |  | DM |  |  |  |
| W级 |  | 31 |  |  | regwrite | IR@W[Rd] | IR@W[Rt] | 31 |
|  | PC8@W |  |  | writeback | AO@W | DO@W | PC8@W |

**第五节 转发暂停控制机制**

1.概述

流水线各流水级间冲突主要体现在寄存器的占用冲突，因此结合分布式译码特性，我们容易获知各流水级所执行指令对寄存器的存取特征。根据这些特征，我们可以分析出转发暂停的构造机制。

2.转发暂停构造表

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | E | | | | M | | | | W | | | |
| RegWrite | | MemToReg | | RegWrite | | MemToReg | | RegWrite | | MemToReg | |
| Rs | Rt | Rs | Rt | Rs | Rt | Rs | Rt | Rs | Rt | Rs | Rt |
| D | Rs | S | - | S | - | F | - | S | - | F | - | F | - |
| Rt | - | S | - | S | - | F | - | S | - | F | - | F |
| E | Rs | - | - | - | - | F | - | S | - | F | - | F |  |
| Rt | - | - | - | - | - | F | - | S | - | F | - | F |
| M | Rt | - | - | - | - | - | - | - | - | - | F | - | F |

**第二章 测试验证**

流水线覆盖性分析测试用例构造表（部分）

1. addu测试样例表

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 用例编号 | | 测试类型 | 前序指令 | 冲突位置 | 冲突寄存器 | 测试样例 |
| 1 | R-M-RS | | subu | MEM | rs | subu $1,$2,$3  addu $4,$1,$2 |
| 2 | R-M-RT | | subu | MEM | rt | subu $1,$2,$3  addu $4,$2,$1 |
| 3 | R-W-RS | | subu | WB | rs | subu $1,$2,$3  nop  addu $4,$1,$2 |
| 4 | R-W-RT | | subu | WB | rt | subu $1,$2,$3  nop  addu $4,$2,$1 |
| 5 | R-W-RS | | subu | WB | rs | subu $1,$2,$3  nop  nop  addu $4,$2,$1 |
| 6 | R-W-RT | | subu | WB | rt | subu $1,$2,$3  nop  nop  addu $4,$2,$1 |
| 7 | I-M-RS | | ori | MEM | rs | ori $1,20  addu $4,$1,$2 |
| 8 | I-M-RT | | ori | MEM | rt | ori $1,20  addu $4,$2,$1 |
| 9 | I-W-RS | | ori | WB | rs | ori $1,20  nop  addu $4,$1,$2 |
| 10 | I-W-RT | | ori | WB | rt | ori $1,20  nop  addu $4,$2,$1 |
| 11 | I-W-RS | | ori | WB | rs | ori $1,20  nop  nop  addu $4,$1,$2 |
| 12 | I-W-RT | | ori | WB | rt | ori $1,20  nop  nop  addu $4,$2,$1 |
| 13 | LW-M-RS | | lw | MEM | rs | lw $1,0($0)  addu $4,$1,$2 |
| 14 | LW-M-RT | | lw | MEM | rt | lw $1,0($0)  addu $4,$2,$1 |
| 15 | LW-WB-RS | | lw | WB | rs | lw $1,0($0)  nop  addu $4,$1,$2 |
| 16 | LW-WB-RT | | lw | WB | rt | lw $1,0($0)  nop  addu $4,$2,$1 |
| 17 | LW-W-RS | | lw | WB | rs | lw $1,0($0)  nop  nop  addu $4,$1,$2 |
| 18 | LW-W-RT | | lw | WB | rt | lw $1,0($0)  nop  nop  addu $4,$2,$1 |
| 19 | J-M-RS | | jal | MEM | rs | jal loop  addu $1,$31,$2  loop: |
| 20 | J-M-RT | | jal | MEM | rt | jal loop  addu $1,$2,$31  loop: |
| 21 | J-W-RS | | jal | WB | rs | jal loop  ori $4,1  loop:  addu $1,$31,$2 |
| 22 | J-W-RS | | jal | WB | rt | jal loop  ori $4,1  loop:  addu $1,$2,$31 |

2.subu测试样例表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 用例编号 | 测试类型 | 前序指令 | 冲突位置 | 冲突寄存器 | 测试样例 |
| 1 | R-M-RS | addu | MEM | rs | addu $1,$2,$3  subu $4,$1,$2 |
| 2 | R-M-RT | subu | MEM | rt | subu $1,$2,$3  subu $4,$2,$1 |
| 3 | R-W-RS | addu | WB | rs | addu $1,$2,$3  nop  addu $4,$1,$2 |
| 4 | R-W-RT | subu | WB | rt | subu $1,$2,$3  nop  subu $4,$2,$1 |
| 5 | R-W-RS | addu | WB | rs | addu $1,$2,$3  nop  nop  addu $4,$1,$2 |
| 6 | R-W-RT | subu | WB | rt | subu $1,$2,$3  nop  nop  subu $4,$2,$1 |
| 7 | I-M-RS | ori | MEM | rs | ori $1,10  subu $4,$1,$2 |
| 8 | I-M-RT | ori | MEM | rt | ori $1,2  subu $4,$2,$1 |
| 9 | I-W-RS | ori | WB | rs | ori $1,10  nop  subu $4,$1,$2 |
| 10 | I-W-RT | ori | WB | rt | ori $1,2  nop  subu $4,$2,$1 |
| 11 | I-W-RS | ori | WB | rs | ori $1,10  nop  nop  subu $4,$1,$2 |
| 12 | I-W-RT | ori | WB | rt | ori $1,2  nop  nop  subu $4,$2,$1 |
| 13 | LW-M-RS | lw | MEM | rs | lw $1,0($0)  subu $4,$1,$2 |
| 14 | LW-M-RT | lw | MEM | rt | lw $1,0($0)  subu $4,$2,$1 |
| 15 | LW-W-RS | lw | WB | rs | lw $1,0($0)  nop  subu $4,$1,$2 |
| 16 | LW-W-RT | lw | WB | rt | lw $1,0($0)  nop  subu $4,$2,$1 |
| 17 | LW-W-RS | lw | WB | rs | lw $1,0($0)  nop  nop  subu $4,$1,$2 |
| 18 | LW-W-RT | lw | WB | rt | lw $1,0($0)  nop  nop  subu $4,$2,$1 |
| 19 | J-M-RS | jal | MEM | rs | jal loop  subu $1,$31,$2  loop: |
| 20 | J-M-RT | jal | MEM | rt | jal loop  subu $1,$2,$31  loop: |
| 21 | J-W-RS | jal | WB | rs | jal loop  ori $4,1  loop:  subu $1,$31,$2 |
| 22 | J-W-RT | jal | WB | rt | jal loop  ori $4,1  loop:  subu $1,$2,$31 |

3.ori测试样例表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 用例编号 | 测试类型 | 前序指令 | 冲突位置 | 冲突寄存器 | 测试样例 |
| 1 | R-M-RS | addu | MEM | rs | addu $1,$2,$3  ori $4,$1,7 |
| 3 | R-W-RS | addu | WB | rs | addu $1,$2,$3  nop  ori $4,$1,7 |
| 5 | I-M-RS | ori | MEM | rs | ori $1,$2,2  ori $3,$1,8 |
| 6 | I-W-RS | ori | WB | rs | ori $1,$2,2  nop  ori $3,$1,8 |
| 7 | LW-M-RS | lw | MEM | rs | lw $1,0($0)  ori $3,$1,2 |
| 8 | LW-W-RS | lw | WB | rs | lw $1,0($0)  nop  ori $3,$1,2 |
| 9 | LW-W-RS | lw | WB | rs | lw $1,0($0)  nop  nop  ori $3,$1,2 |
| 10 | J-M-RS | jal | MEM | rs | jal loop  ori $31,$31,1  loop: |
| 11 | J-W-RS | jal | WB | rs | jal loop  ori $1,$1,1  loop:  ori $31,$31,1 |

4.j测试样例表

|  |  |  |
| --- | --- | --- |
| 用例编号 | 测试类型 | 测试样例 |
| 1 | J | ori $2,5  ori $1,1  addu $1,$1,$2  j exit  ori $3,1  exit: |

5.lw测试样例表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 用例编号 | 测试类型 | 前序指令 | 冲突位置 | 冲突寄存器 | 测试样例 |
| 1 | R-M-RS | addu | MEM | rs | addu $1,$2,$3  lw $4,0($1) |
| 2 | R-W-RS | addu | WB | rs | addu $1,$2,$3  nop  lw $4,0($1) |
| 3 | R-W-RS | addu | WB | rs | addu $1,$2,$3  nop  nop  lw $4,0($1) |
| 4 | I-M-RS | ori | MEM | rs | ori $1,8  lw $4,0($1) |
| 5 | I-W-RS | ori | WB | rs | ori $1,8  nop  lw $4,0($1) |
| 6 | I-W-RS | ori | WB | rs | ori $1,8  nop  nop  lw $4,0($1) |
| 7 | LW-M-RS | lw | MEM | rs | lw $1,4($0)  lw $4,0($1) |
| 8 | LW-W-RS | lw | WB | rs | lw $1,4($0)  nop  lw $4,0($1) |
| 9 | LW-W-RS | lw | WB | rs | lw $1,4($0)  nop  nop  lw $4,0($1) |
| 10 | J-M-RS | jal | MEM | rs | jal loop  lw $1,0($31)  loop: |
| 11 | J-M-RT | jal | WB | rs | jal loop  loop:  lw $1,0($31) |

6.sw测试样例表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 用例编号 | 测试类型 | 前序指令 | 冲突位置 | 冲突寄存器 | 测试样例 |
| 1 | R-M-RS | addu | MEM | rs | addu $1,$2,$3  sw $2,0($1) |
| 2 | R-W-RS | addu | WB | rs | addu $1,$2,$3  nop  sw $2,0($1) |
| 3 | R-W-RS | addu | WB | rs | addu $1,$2,$3  nop  nop  sw $2,0($1) |
| 4 | I-M-RS | ori | MEM | rs | ori $1,8  sw $2,0($1) |
| 5 | I-W-RS | ori | WB | rs | ori $1,8  nop  sw $2,0($1) |
| 6 | I-W-RS | ori | WB | rs | ori $1,8  nop  nop  sw $2,0($1) |
| 7 | LW-M-RS | lw | MEM | rs | lw $1,4($0)  sw $2,0($1) |
| 8 | LW-W-RS | lw | WB | rs | lw $1,4($0)  nop  sw $2,0($1) |
| 9 | LW-W-RS | lw | WB | rs | lw $1,4($0)  nop  nop  sw $2,0($1) |
| 10 | J-M-RS | jal | MEM | rs | jal loop  sw $1,0($31)  loop: |
| 11 | J-M-RT | jal | WB | rs | jal loop  loop:  sw $1,0($31) |

7.beq测试样例表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 用例编号 | 测试类型 | 前序指令 | 冲突位置 | 冲突寄存器 | 测试样例 |
| 1 | R-M-RS | addu | MEM | rs | addu $1,$3,$0  beq $1,$3,exit  ori $4,1  ori $4,2  exit: |
| 2 | R-M-RT | addu | MEM | rt | addu $1,$3,$0  beq $3,$1,exit  ori $4,1  ori $4,2  exit: |
| 3 | R-W-RS | addu | WB | rs | addu $1,$3,$0  nop  beq $1,$3,exit  ori $4,1  ori $4,2  exit: |
| 4 | R-W-RT | addu | WB | rt | addu $1,$3,$0  nop  beq $3,$1,exit  ori $4,1  ori $4,2  exit: |
| 5 | R-W-RS | addu | WB | rs | addu $1,$3,$0  nop  nop  beq $1,$3,exit  ori $4,1  ori $4,2  exit: |
| 6 | R-W-RT | addu | WB | rt | addu $1,$3,$0  nop  nop  beq $3,$1,exit  ori $4,1  ori $4,2  exit: |
| 7 | I-M-RS | ori | MEM | rs | ori $1,3  beq $1,$3,exit  ori $4,1  ori $4,2  exit: |
| 8 | I-M-RT | ori | MEM | rt | ori $1,3  beq $3,$1,exit  ori $4,1  ori $4,2  exit: |
| 9 | I-W-RS | ori | WB | rs | ori $1,3  nop  beq $1,$3,exit  ori $4,1  ori $4,2  exit: |
| 10 | I-W-RT | ori | WB | rt | ori $1,3  nop  beq $3,$1,exit  ori $4,1  ori $4,2  exit: |
| 11 | I-W-RS | ori | WB | rs | ori $1,3  nop  nop  beq $1,$3,exit  ori $4,1  ori $4,2  exit: |
| 12 | I-W-RT | ori | WB | rt | ori $1,3  nop  nop  beq $3,$1,exit  ori $4,1  ori $4,2  exit: |
| 13 | LW-M-RS | lw | MEM | rs | addu $4,$2,$3  lw $1,8($0)  beq $1,$4,exit  ori $5,1  ori $5,2  exit: |
| 14 | LW-M-RT | lw | MEM | rt | addu $4,$2,$3  lw $1,8($0)  beq $4,$1,exit  ori $5,1  ori $5,2  exit: |
| 15 | LW-WB-RS | lw | WB | rs | addu $4,$2,$3  lw $1,8($0)  nop  beq $1,$4,exit  ori $5,1  ori $5,2  exit: |
| 16 | LW-WB-RT | lw | WB | rt | addu $4,$2,$3  lw $1,8($0)  nop  beq $4,$1,exit  ori $5,1  ori $5,2  exit: |
| 17 | LW-W-RS | lw | WB | rs | addu $4,$2,$3  lw $1,8($0)  nop  nop  beq $1,$4,exit  ori $5,1  ori $5,2  exit: |
| 18 | LW-W-RT | lw | WB | rt | addu $4,$2,$3  lw $1,8($0)  nop  nop  beq $4,$1,exit  ori $5,1  ori $5,2  exit: |

8.jr测试样例表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 用例编号 | 测试类型 | 前序指令 | 冲突位置 | 冲突寄存器 | 测试样例 |
| 1 | R-M-RS | addu | MEM | rs | ori $1,4  jal loop  addu $31,$31,$1  ori $2,1  ori $3,1  j exit  ori $8,1  loop:  jr $31  ori $4,1  exit: |
| 2 | R-W-RS | addu | WB | rs | ori $1,4  jal loop  addu $31,$31,$1  ori $2,1  ori $3,1  j exit  ori $8,1  loop:  ori $5,1  jr $31  ori $4,1  exit: |
| 3 | R-W-RS | subu | WB | rs | ori $1,4  jal loop  addu $31,$31,$1  ori $2,1  ori $3,1  j exit  ori $8,1  loop:  ori $5,1  ori $6,1  jr $31  ori $4,1  exit: |
| 4 | I-M-RS | ori | MEM | rs | ori $1,4  ori $1,4  ori $1,4  jal loop  ori $31,8  ori $2,1  ori $3,1  j exit  ori $8,1  loop:  jr $31  ori $4,1  exit: |
| 5 | I-W-RS | ori | WB | rs | ori $1,4  ori $1,4  ori $1,4  jal loop  ori $31,8  ori $2,1  ori $3,1  j exit  ori $8,1  loop:  ori $5,1  jr $31  ori $4,1  exit: |
| 6 | I-W-RS | ori | WB | rs | ori $1,4  ori $1,4  ori $1,4  jal loop  ori $31,8  ori $2,1  ori $3,1  j exit  ori $8,1  loop:  ori $5,1  ori $6,1  jr $31  ori $4,1  exit: |
| 7 | LW-M-RS | lw | MEM | rs | ori $1,4  ori $2,0x301c  sw $2,0($0)  jal loop  lw $31,0($0)  ori $2,1  ori $3,1  j exit  ori $8,1  loop:  jr $31  ori $4,1  exit: |
| 8 | LW-WB-RS | lw | WB | rs | ori $1,4  ori $2,0x301c  sw $2,0($0)  jal loop  lw $31,0($0)  ori $2,1  ori $3,1  j exit  ori $8,1  loop:  ori $5,1  jr $31  ori $4,1  exit: |
| 9 | LW-W-RS | lw | WB | rs | ori $1,4  ori $2,0x301c  sw $2,0($0)  jal loop  lw $31,0($0)  ori $2,1  ori $3,1  j exit  ori $8,1  loop:  ori $5,1  ori $6,1  jr $31  ori $4,1  exit: |
| 10 | J-M-RS | jal | WB | rs | ori $1,4  jal loop  addu $31,$31,$1  ori $2,1  ori $3,1  j exit  ori $8,1  loop:  jr $31  ori $4,1  exit: |
| 11 | J-W-RS | jal | WB | rs | ori $1,4  jal loop  addu $31,$31,$1  ori $2,1  ori $3,1  j exit  ori $8,1 |

**第三章 思考题**

1.为什么需要有单独的乘除法部件而不是整合进ALU？为何需要有独立的HI、LO寄存器？

答：在真实的硬件设备中，乘除法的运算需要调用更多的资源，消耗更多的时间，所以乘除法运算需要单独的部件，来控制指令流。独立的HI/LO寄存器是出于处理器的需要，根据指令集规定的行为，乘除法的运算结果并不直接写回寄存器，而是先保存在HI/LO寄存器中，所以必须在处理器中设置这两个寄存器。

2.参照你对延迟槽的理解，试解释“乘除槽”。

答：延迟槽即跳转类指令（包括分支类跳转和绝对跳转）后的那条指令，该条指令在不影响程序功能的情况下，可以不受跳转指令的控制而强制执行，从而提高了流水线的工作效率。

乘除槽的概念是仿照延迟槽的概念进行定义的，在乘除法运算后，根据模拟硬件行为的要求，需要对下一条的读写HI/LO寄存器或乘除运算的指令进行暂停。但若能进行合理的编译优化，则可将有意义且逻辑正确的指令装载到乘除槽中，从而提高流水线的工作效率。

3.为何上文文末提到的lb等指令使用的数据扩展模块应在 MEM/WB 之后，而不能在 DM 之后?

答：因为对DM的访问导致M级的运行时间最长，决定了流水线的运行周期。若数据扩展模块位于该级，将延长流水线的运行周期，导致流水线的资源浪费，和性能下降。因而应将扩展模块置于MEM/WB之后的W级，减少对流水线整体性能的影响。

4.举例说明并分析何时按字节访问内存相对于按字访问内存性能上更有优势。（Hint： 考虑C语言中字符串的情况）

答：对于访问内存地址不对齐的情况，按字节访问更有效，因为按字访存，只能访问首地址为4的倍数且内存连续的内容。但是按字节访存则更为灵活，可以应对不同类型的指令，如lb,lwl,lwr等指令。

5.如何概括你所设计的CPU的设计风格？为了对抗复杂性你采取了哪些抽象和规范手段？你对流水线CPU设计风格有何见解？

答：在阅读指导书完毕后，我将自己设计的CPU设计风格归结为探索者风格。我在指令集的每次扩增的过程中，对新的指令进行寄存器占用分析，对冒险的转发暂停进行动态的设计，这种设计风格更偏向于“探索者”。

为了对抗呈指数级增长的复杂性，我将流水线的暂停规范约束在E级流水段，以E级作为分析的基准，对于每条新指令，若该指令需要访问寄存器的值的位置发生在E级之前，或该指令更新寄存器的值行为发生在E级之后，则设计相应的暂停机制。

对于转发机制的设计，则完全依赖于寄存器的占用关系，前后级流水段的指令对同一寄存器进行占用（读、写操作），则可能需要转发数据，而具体是否发生转发，或转发的优先级如何，则依赖于具体的指令判断。这对于设计来说有一个工作量上的大大简化，因为对于占用寄存器的行为分析只需进行一次，译码也可交付控制器统一进行，分类项目也更为简化。综上所述，这种分析模式，有利于设计和处理器功能的扩展。

唯一的缺点，就是这种设计模式不利于显示的表达，一旦发生错误，将陷入调试的深渊。

