存档资料

成绩：

# 武汉大学计算机学院

# 实 验 报 告

课程名称 计算机系统综合

专业班级 16 级弘毅计算机

学　　号 2016300030006

学生姓名 关焕康

指导教师 刘芹

2018 年 7 月 7 日

# 设计任务书

**主要内容**

用硬件描述语言（Verilog）设计 MIPS 流水线 CPU，单周期支持如下指令集：{addu，subu，ori，lw，sw，beq，jal，and，nor} ，多周期和流水线新增 29 条指令实现了 Mips-C2 指令集{LB、LBU、LH、LHU、LW、SB、SH、SW、ADD、ADDU、SUB、SUBU、SLL、SRL、SRA、SLLV、SRLV、SRAV、AND、OR、XOR、NOR、SLT、SLTU、ADDI、ADDIU、ANDI、ORI、XORI、LUI、SLTI、SLTIU、BEQ、BNE、BLEZ、BGTZ、BLTZ、BGEZ、J、JAL、JALR、JR}。支持中断异常的流水线在 Mips-C2 指令集基础上新增 4 条指令{syscall,mfc0,mtc0,eret}。

用仿真软件 Modelsim 对有数据冒险和指令冒险的汇编程序进行仿真。最终在 FPGA 上完成包括但不限于跑马灯，矩形变换，文本显示，PC 检查等测试。

**基本要求**

1. 熟悉硬件描述语言（Verilog）和仿真软件 Modelsim；
2. 用硬件描述语言（Verilog）设计如下主要模块：
   (1) 程序计数器模块（PC）；
   (2) 寄存器堆模块（RF）；
   (3) 主控制器模块（MCU）
   (4) 指令存储器模块（IM）
   (5) 分支决策模块（Branch）
   (6) 零扩展模块（ZE）
   (7) 符号扩展模块（SE）
   (8) 左移 2 位模块（SL）
   (9) 左移 226 模块（SL226）
   (10) 清洗流水线模块（Erasure）
   (11) 数据冒险检测模块（HDU）
   (12) 数据存储器模块（DM）
   (13) 算术运算模块（ALU）
   (14) 算术运算控制器模块（ALUCU）

(15) 协处理器寄存器堆模块（c0rf）

(16) 中断异常决策模块（intmodule）

(17) 数据冒险桥接模块（FU）

(18) 取指与解码之间寄存器（IFID）

(19) 解码与执行之间寄存器（IDEX）

(20) 执行与访存之间寄存器（EXMEM）

(21) 访存与写回之间寄存器（MEMWB）

(22) 二路选择模块（Mux1bits）

(23) 四路选择模块（Mux2bits）

(24) 八路选择模块（Mux3bits）

3. 完成汇编程序的仿真调试。

4. 完成在 FPGA 上的测试。

**参考资料**

【1】计算机原理与设计：Verilog HDL 版，李亚民著。

目录

代码上传至：

Github:git@github.com:guanhuankang/Mips.git

# 第 1 章 需求分析

学习了计算机组成原理课程后需要融会贯通计算机组成与设计课程所教授的知识，通过对知识的综合应用，加深对 CPU 系统各模块的工作原理及相互联系。

通过课程设计来学习 Mips 架构的 CPU 的工作原理和基于 Verilog 的硬件描述语言来设计 CPU 的方法，掌握采用 ModelSim 仿真技术进行调试和仿真的技术，培养科学研究的独立工作能力和分析解决问题的能力，取得 CPU 设计与仿真的实

践和经验。

最后通过 ISE 软件编译下载到 FPGA 上进行简单测试，包括但不限于跑马灯，矩形变换，文本显示，PC 检查的测试。

# 第 2 章 设计环境

## 2.1 Verilog HDL 简介

Verilog HDL 是一种硬件描述语言（HDL:Hardware Description Language），以文本形式来描述数字系统硬件的结构和行为的语言，用它可以表示逻辑电路图、逻辑表达式，还可以表示数字逻辑系统所完成的逻辑功能。 Verilog HDL 和 VHDL 是世界上最流行的两种硬件描述语言，都是在 20 世纪 80 年代中期开发出来的。前者由 Gateway Design Automation 公司（该公司于 1989 年被 Cadence 公司收购）开发。两种 HDL 均为 IEEE 标准。

## 2.2 ModelSim 简介

Mentor 公司的 ModelSim 是业界最优秀的 HDL 语言仿真软件，它能提供友好的仿真环境，是业界唯一的单内核支持 VHDL 和 Verilog 混合仿真的仿真器。它

采用直接优化的编译技术、Tcl/Tk 技术、和单一内核仿真技术，编译仿真速度快，编译的代码与平台无关，便于保护 IP 核，个性化的图形界面和用户接口，为用户加快调错提供强有力的手段，是 FPGA/ASIC 设计的首选仿真软件。

本次实验使用 10.0 版本。

## 2.3 MARS 简介

一款基于 JAVA 的 Mips 汇编器。用该汇编器执行 Mips 指令，同时带有汇编功能，将 mips 指令转换成机器指令输出。附件中有该工具。

## 2.4 ISE 简介

ISE 是使用 XILINX 的 FPGA 的必备的设计工具。目前官方提供下载的最新版本是 14.7。它可以完成 FPGA 开发的全部流程，包括设计输入、仿真、综合、布局布线、生成 BIT 文件、配置以及在线调试等，功能非常强大。ISE 除了功能完整，使用方便外，它的设计性能也非常好，拿 ISE 9.x 来说，其设计性能比其他解决方案平均快 30%，它集成的时序收敛流程整合了增强性物理综合优化，提供最佳的时钟布局、更好的封装和时序收敛映射，从而获得更高的设计性能。先进的综合和实现算法将动态功耗降低了 10%。

# 第 3 章 概要设计

## 3.1 PC（程序计数器）

### 3.1.1 功能描述

PC 模块总是指向下一指令地址，对 PC 的控制从而可以改变程序控制流。

### 3.1.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Clk | Input | 时钟信号 |
| Pc_in | Input | PC 待写入值 |
| PCWrite | Input | PC 写信号 |
| Rst | Input | 复位信号 |
| Pc_out | Output | PC 输出值 |

## 3.2 RF（寄存器堆）

### 3.2.1 功能描述

RF 模块负责 32 个寄存器的读写操作。

### 3.2.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Clk | Input | 时钟信号 |
| Rst | Input | 复位信号 |
| RegWrite | Input | RF 写信号 |
| Raddr1 | Input | 读地址 1 |

| | | |
|---|---|---|
| Raddr2 | Input | 读地址 2 |
| Waddr | Input | 写地址 |
| Wdata | Input | 写数据 |
| Rdata1 | Output | 读数据 1 |
| Rdata2 | Output | 读数据 2 |

## 3.3 MCU（主控制器）

### 3.3.1 功能描述

MCU 主控制器负责产生一条指令需要的所有的信号。

### 3.3.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Instr | Input | 指令 |
| WB | Output | 写回阶段所需信号 |
| M | Output | 访存阶段所需信号 |
| EX | Output | 执行阶段所需信号 |
| EXCE | Output | 中断异常所需信号 |

## 3.4 IM（指令存储器）

### 3.4.1 功能描述

IM 指令存储器，存储执行的指令，不包含数据。(address, instr, clk);

### 3.4.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|

| | | |
|---|---|---|
| Address | Input | 地址输入 |
| Instr | Output | 指令输出 |
| Clk | Input | 时钟输入（未使用） |

## 3.5 Branch（分支决策）

### 3.5.1 功能描述

Branch 模块接受具体分支信号，判断是否跳转。

### 3.5.2 分支识别码

| 指令 | 识别码（3 位） |
|---|---|
| Beq | 100 |
| Bne | 101 |
| BLEZ,BGTZ,BLTZ,BGEZ | 110 |
| J,JAL,JALR,JR | 111 |
| 其他 | 000 |

### 3.5.3 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Bs | Input | 分支识别码 |
| Zero | Input | 零信号 |
| Alurs | Input | 运算模块输出结果 |
| Jrs | Output | 是否跳转 |

## 3.6 ZE（零扩展模块）

### 3.6.1 功能描述

ZE 输入低 16 位，0 补全高 16 位输出。

### 3.6.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| ze_in | Input | 16 位输入 |
| ze_out | Output | 32 位输出 |

## 3.7 SE（符号扩展模块）

### 3.7.1 功能描述

输入低 16 位，输出符号扩展 32 位结果。

### 3.7.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Se_in | Input | 16 位输入 |
| Se_out | Output | 32 位输出 |

## 3.8 SL（左移 2 位模块）

### 3.8.1 功能描述

SL 输入 32 位，逻辑左移 2 位输出。

### 3.8.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Sl_in | Input | 32 位输入 |

| | | |
|---|---|---|
| Sl_out | Output | 32 位输出 |

## 3.9 SL226（左移 226）

### 3.9.1 功能描述

SL226 接受一个输入 sl226_in 和 pc 输入，先将 sl226_in 左移两位后作为输出低 28 位，高四位输出 pc 高四位。

### 3.9.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Sl226_in | Input | 32 位输入 |
| Pcin | Input | Pc 输入 |
| Sl226_out | Output | 32 位输出 |

## 3.10 Erasure（清洗流水线模块）

### 3.10.1 功能描述

该模块位于第四级流水线上，当跳转指令，异常中断等发生时，清洗不应该在流水线上的指令。Erasure(isj, rst1, rst2, rst3, clk, rst);

### 3.10.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Isj | Input | 是否需要清洗 |
| Rst | Input | 全局重置信号 |
| Rst1 | Output | 重置 IFID 寄存器 |
| Rst2 | Output | 重置 IDEX 寄存器 |
| Rst3 | Output | 重置 EXMEM 寄存器 |

| | | |
|---|---|---|
| Clk | Input | 全局时钟信号（未使用） |

## 3.11 HDU（数据冒险检测模块）

### 3.11.1 功能描述

检查是否出现有关于 lw 的数据冒险，若出现某指令依赖 lw 数据时，输出产生气泡信号。

### 3.11.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Instr | Input | 指令输入 |
| Rs | Input | 上一条指令 rs |
| Rt | Input | 上一条指令 rt |
| Rd | Input | 当前指令目的寄存器地址 |
| MMR | Input | 读数据存储器 DM 信号 |
| Rst2 | Output | IDEX 寄存器重置信号 |
| PCWriteA | Output | 数据冒险输出 1，反之 0 |
| IFID | Output | IFID 寄存器写信号 |

## 3.12 DM（数据存储模块）

### 3.12.1 功能描述

DM 数据存储器，专门存放数据，实质上是一个 RAM。

### 3.12.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Address | Input | 地址输入 |
| Dm_in | Input | 写数据 |

| | | |
|---|---|---|
| MemWrite | Input | 写入信号 |
| MemRead | Input | 读出信号 |
| SBHW | Input | sw,sh,sb 标识码 |
| Clk | Input | 时钟信号 |
| Dm_out | Output | 数据输出 |

## 3.13 ALU（算术运算模块）

### 3.13.1 功能描述

ALU 负责完成指令的计算问题，如加减乘除，移位等操作，输出溢出信号，zero 信号，计算结果。

### 3.13.2 运算操作标识码

| 运算操作 | 标识码 |
|---|---|
| ALU_Invalid | 5'b00_000 |
| ALU_Add | 5'b00_001 |
| ALU_Addu | 5'b00_010 |
| ALU_Sub | 5'b00_011 |
| ALU_Subu | 5'b00_100 |
| ALU_Sll | 5'b00_101 |
| ALU_Srl | 5'b00_110 |
| ALU_Sra | 5'b00_111 |
| ALU_And | 5'b01_000 |
| ALU_Or | 5'b01_001 |
| ALU_Xor | 5'b01_010 |
| ALU_Nor | 5'b01_011 |
| ALU_Slt | 5'b01_100 |
| ALU_Sltu | 5'b01_101 |
| ALU_Lui | 5'b01_110 |

| | |
|---|---|
| ALU_Le | 5'b01_111 |
| ALU_Gt | 5'b10_000 |
| ALU_Ge | 5'b10_001 |
| ALU_Jalr | 5'b00_000 |
| ALU_Jr | 5'b00_000 |

### 3.13.3 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| SrcA | Input | 操作数 A |
| SrcB | Input | 操作数 B |
| ALUOPCtrl | Input | 运算操作标识码，指示执行何种运算 |
| Zero | Output | 零信号，运算结果为 0 输出 1 |
| Ovf | Output | 溢出信号，溢出输出 1 |
| Aluout | Output | 运算结果输出（32 位） |

## 3.14 ALUCU（运算单元控制器）

### 3.14.1 功能描述

主控制器输出操作信号，该模块结合指令低六位功能码输出运算操作识别码，这样 ALU 才可以正确运算。

### 3.14.2 操作码

| 指令 | 操作码 |
|---|---|
| ALUOP_Invalid | 5'b00_000 |
| ALUOP_Load | 5'b00_010 |
| ALUOP_Store | 5'b00_010 |
| ALUOP_R_Type | 5'b10_000 |
| ALUOP_Addi | 5'b01_000 |

| | |
|---|---|
| ALUOP_Addiu | 5'b01_001 |
| ALUOP_Slti | 5'b01_010 |
| ALUOP_Sltiu | 5'b01_011 |
| ALUOP_Andi | 5'b01_100 |
| ALUOP_Ori | 5'b01_101 |
| ALUOP_Xori | 5'b01_110 |
| ALUOP_Lui | 5'b01_111 |
| ALUOP_Beq | 5'b00_100 |
| ALUOP_Bne | 5'b00_101 |
| ALUOP_Blez | 5'b00_110 |
| ALUOP_Bgtz | 5'b00_111 |
| ALUOP_Bltz | 5'b00_001 |
| ALUOP_Bgez | 5'b10_001 |
| ALUOP_J | 5'b00_000 |
| ALUOP_JAL | 5'b00_000 |
| ALUOP_JALR | 5'b10_000 |
| ALUOP_JR | 5'b10_000 |

### 3.14.3 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| ALUOP | Input | 主控制输出操作信号 |
| Funccode | Input | 指令功能码 |
| ALUOPCtrl | Output | 运算操作码 |
| Clk | Input | 时钟信号（未使用） |

## 3.15 C0rf（协处理器寄存器堆模块）

### 3.15.1 功能描述

C0rf 负责协处理器寄存器堆的读写。

### 3.15.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Rst | Input | 复位信号 |
| Clk | Input | 时钟信号 |
| INT | Input | 中断信号 |
| Wepc | Input | 写入 epc 值 |
| Wcause | Input | 写入中断号（实质是中断号乘 4） |
| Wstatus | Input | 写入 status |
| Raddr | Input | 读地址 |
| Waddr | Input | 写地址 |
| Wdata | Input | 写数据 |
| C0w | Input | 写信号 |
| Back | Input | 寄存器回退信号 |
| Rdata | Output | 读数据输出 |
| Status | Output | Status 值输出，即 12 号寄存器输出 |
| EPC | Output | EPC 值输出，即 14 号寄存器输出 |
| Cause | Output | Cause 值输出，即 13 号寄存器输出 |

## 3.16 Intmodule（中断异常决策模块）

### 3.16.1 功能描述

中断异常决策模块位于第四级流水线位置，第四级是所有指令检查中断异常的点，intmodule 接收所有中断异常信号挑选一个未被屏蔽的最高级中断输出，同时屏蔽该中断号。所以本决策决定屏蔽同级中断。

### 3.16.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| INT32 | Input | 32 个中断号，1 中断 |
| Status | Input | 当前指令对应屏蔽变量 |
| Epcin | Input | 当前指令地址 |
| Wcause | Output | 输出中断号（实质是中断号*4） |
| Wsattus | Output | 输出新 status 值，屏蔽被选择的中断 |
| INT | Output | 是否有中断被选择 |
| Clk | Input | 时钟信号（未使用） |

## 3.17 FU（数据冒险桥接模块）

### 3.17.1 功能描述

Forwarding 是解决流水线数据冒险的一种方案。FU(Forwarding Unit)解决了 MEM->EX,WB->EX 的数据冒险。

### 3.17.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Rs | Input | 执行阶段指令 rs 的地址 |
| Rt | Input | 执行阶段指令 rt 的地址 |
| Exrd | Input | 访存阶段指令目的地址 |

| | | |
|---|---|---|
| Exw | Input | 访存阶段指令写 RF 信号 |
| Mrd | Input | 写回阶段指令目的地址 |
| Mw | Input | 写回阶段指令写 RF 信号 |
| Fa | Output | ALU 选择操作数 A 的指示信号 |
| Fb | Output | ALU 选择操作数 B 的指示信号 |

## 3.18 IFID（取指与解码阶段之间寄存器）

### 3.18.1 功能描述

IFID 取指与解码阶段之间寄存器。

### 3.18.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Clk | Input | 时钟信号 |
| IFIDW | Input | 写信号 |
| Rst1 | Input | 复位信号 |
| Pcin | Input | PC 当前值 |
| Instr | Input | 读取的指令 |
| Opcin | Output | 输出 PC 值 |
| Oinstr | Output | 输出指令 |

## 3.19 IDEX（解码与执行阶段之间寄存器）

### 3.19.1 功能描述

IDEX 解码与执行阶段之间寄存器。

### 3.19.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|

| | | |
|---|---|---|
| Clk | Input | 时钟信号 |
| IDEXW | Input | 写信号 |
| Rst2 | Input | 复位信号 |
| WB | Input | 写回阶段信号集合 |
| M | Input | 访存阶段信号集合 |
| EX | Input | 执行阶段信号集合 |
| Pcnext | Input | Pc+4 往下传递 |
| Rd1 | Input | 读数据 1 |
| Rd2 | Input | 读数据 2 |
| Zer | Input | 零扩展 |
| Ser | Input | 符号扩展 |
| Rs | Input | 指令 rs 地址 |
| Rt | Input | 指令 rt 地址 |
| Rd | Input | 指令 rd 地址 |
| Low26 | Input | 指令低 26 位 |
| Instr | Input | 指令 |
| EXCEREG_in | Input | 中断异常相关信号 129 位信号{eret，INT32，status，EPC，epcin} |
| OWB | Output | 写回阶段信号集合-输出 |
| OM | Output | 访存阶段信号集合-输出 |
| OEX | Output | 执行阶段信号集合-输出 |
| OPcnext | Output | Pc+4 往下传递-输出 |
| ORd1 | Output | 读数据 1-输出 |
| ORd2 | Output | 读数据 2-输出 |
| OZer | Output | 零扩展-输出 |
| OSer | Output | 符号扩展-输出 |
| ORs | Output | 指令 rs 地址-输出 |

| | | |
|---|---|---|
| ORt | Output | 指令 rt 地址-输出 |
| ORd | Output | 指令 rd 地址-输出 |
| OLow26 | Output | 指令低 26 位-输出 |
| OInstr | Output | 指令-输出 |
| OEXCEREG | Output | 中断异常相关信号 129 位信号{eret，INT32，status，EPC，epcin}-输出 |

## 3.20 EXMEM（执行与访存阶段之间寄存器）

### 3.20.1 功能描述

EXMEM 执行与访存阶段之间寄存器。

### 3.20.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Clk | Input | 时钟信号 |
| IDEXW | Input | 写信号 |
| Rst3 | Input | 复位信号 |
| WB | Input | 写回阶段信号集合 |
| M | Input | 访存阶段信号集合 |
| Jaddr | Input | J 型跳转类指令目的地址 |
| Addres | Input | 非 J 型跳转类指令目的地址 |
| Rd1 | Input | 读数据 1 |
| Zero | Input | 运算结果为零标志 |
| Aluout | Input | 运算结果 |
| Rd2 | Input | 读数据 2 |
| Dst | Input | 写 RF 目的地址 |
| Pc | Input | Pc+4 |
| EXMEM_in | Input | 异常中断类相关指令，在上一级流水 |

| | | |
|---|---|---|
| | | 线寄存器基础上计入 ovf 和 INT 中断 |
| OWB | Output | 写回阶段信号集合-输出 |
| OM | Output | 访存阶段信号集合-输出 |
| OJaddr | Output | J 型跳转类指令目的地址-输出 |
| OAddres | Output | 非 J 型跳转类指令目的地址-输出 |
| ORd1 | Output | 读数据 1-输出 |
| OZero | Output | 运算结果为零标志-输出 |
| OAluout | Output | 运算结果-输出 |
| ORd2 | Output | 读数据 2-输出 |
| ODst | Output | 写 RF 目的地址-输出 |
| OPc | Output | Pc+4-输出 |
| OEXMEM | Output | 异常中断类相关指令，在上一级流水线寄存器基础上计入 ovf 和 INT 中断-输出 |

## 3.21 MEMWB（访存与写回阶段之间寄存器）

### 3.21.1 功能描述

MEMWB 访存与写回阶段之间寄存器。

### 3.21.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| Clk | Input | 时钟信号 |
| MEMWBW | Input | 写信号 |
| Rst4 | Input | 复位信号 |
| WB | Input | 写回阶段信号集合 |
| Rdfm | Input | Read Data From Memory 从数据存储器中读出来的数据 |
| Aluout | Input | 运算结果 |

| | | |
|---|---|---|
| Dst | Input | 写入 RF 的写信号 |
| OWB | Output | 写回阶段信号集合-输出 |
| ORdfm | Output | Read Data From Memory 从数据存储器中读出来的数据-输出 |
| OAluout | Output | 运算结果-输出 |
| ODst | Output | 写入 RF 的写信号-输出 |

## 3.22 Mux1Bits（二路选择）

### 3.22.1 功能描述

二路选择。

### 3.22.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| D1 | Input | 选择输入 0 |
| D2 | Input | 选择输入 1 |
| S | Input | 选择信号 1 位 |
| Dout | Output | 选择输出 |

## 3.23 Mux2Bits（四路选择）

### 3.23.1 功能描述

四路选择。

### 3.23.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| M0 | Input | 选择输入 0 |
| M1 | Input | 选择输入 1 |

| | | |
|---|---|---|
| M2 | Input | 选择输入 2 |
| M3 | Input | 选择输入 3 |
| Ctrl | Input | 选择信号 2 位 |
| Mout | Output | 选择输出 |

## 3.24 Mux3Bits（八路选择）

### 3.24.1 功能描述

八路选择。

### 3.24.2 模块接口

| 信号名 | 方向 | 描述 |
|---|---|---|
| M0 | Input | 选择输入 0 |
| M1 | Input | 选择输入 1 |
| M2 | Input | 选择输入 2 |
| M3 | Input | 选择输入 3 |
| M4 | Input | 选择输入 4 |
| M5 | Input | 选择输入 5 |
| M6 | Input | 选择输入 6 |
| M7 | Input | 选择输入 7 |
| Ctrl | Input | 选择信号 2 位 |
| Mout | Output | 选择输出 |

# 第 4 章 详细设计

## 4.1 单周期 CPU 总体结构

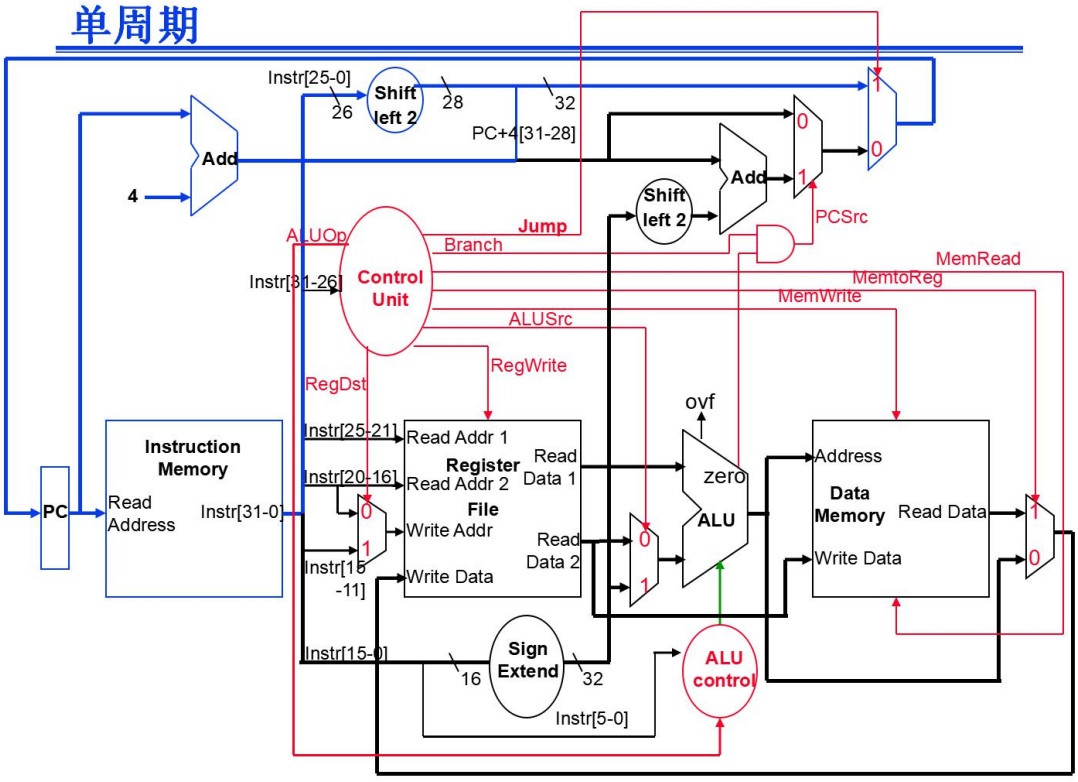


图 1 单周期 CPU 总体结构

单周期的 CPU 总体结构如上图 1 所示，其中包括程序计数器（PC)、指令存储器（IM）、寄存器组（RF）、运算器（ALU）、符号扩展单元（SE）、数据存储器（DM）和主控制器（MCU）。

## 4.2 多周期 CPU 总体结构

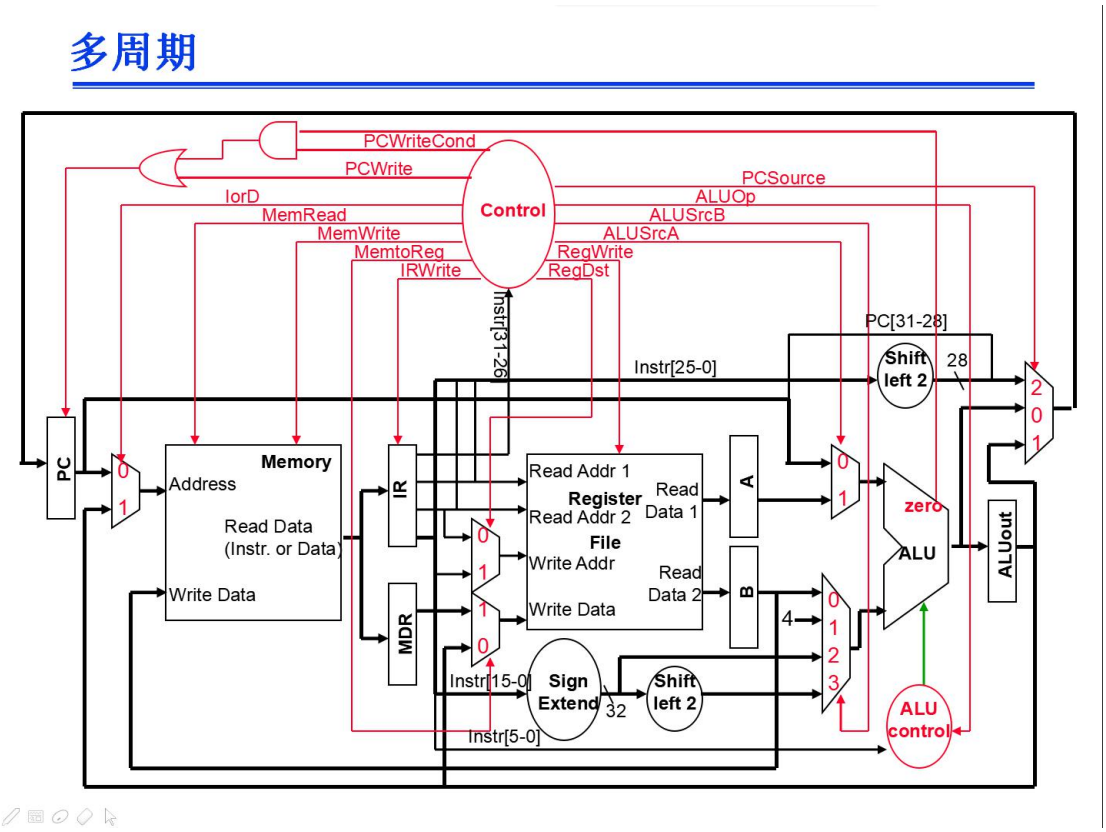


图 2 多周期 CPU 总体结构

多周期的 CPU 总体结构如上图 2 所示，其中包括程序计数器（PC)、指令数据公用存储器（Memory）、数据暂存器（MDR）、指令暂存器（IR）、寄存器组（RF）、运算器（ALU）、符号扩展单元（SE）、和主控制器（MCU）。

## 4.3 流水线 CPU 总体结构

流水线的 CPU 总体结构如下图 3 所示，其中包括程序计数器（PC)、指令存储器（IM）、寄存器组（RF）、运算器（ALU）、ForwardingUnit（FU）、数据存储器（DM）、数据冒险检测模块（HDU），各级流水线之间寄存器组（IFID、IDEX、EXMEM、MEMWB）和主控制器（MCU）。

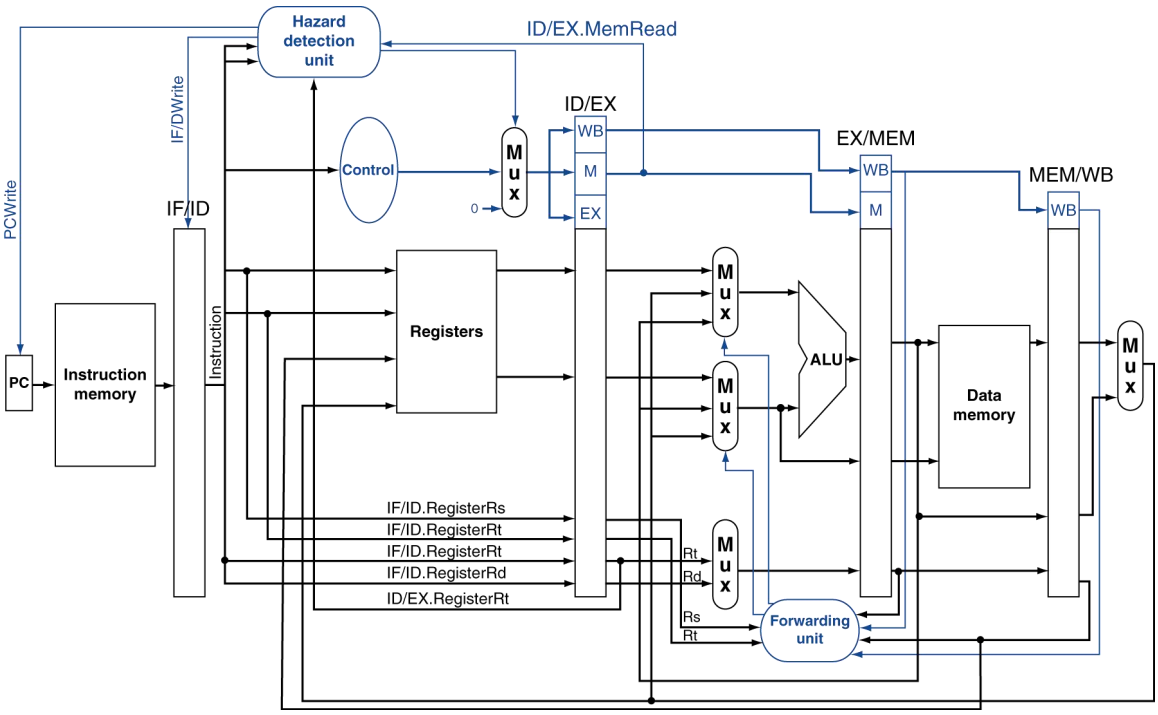


图 3 流水线 CPU 总体结构

## 4.4 流水线的进一步分析

为了进一步说明流水线工作过程，下面逐级剖析流水线工作原理。总共五级流水，第一级取指阶段（IF 阶段），完成取出指令；第二级解码阶段（ID 阶段），完成指令的解码，生成所需要的所有信号，同时从 RF 中读出 rs，rt 的值；第三阶段执行阶段（EX 阶段），完成转移地址计算，指令要求运算；第四级访存阶段（MEM 阶段）完成对 memory 的访问；第五级写回阶段（WB 阶段），完成选择目的值和根据写回信号，回写 RF，指令完成。

下面以 lw 指令为例子。逐级流动图示。

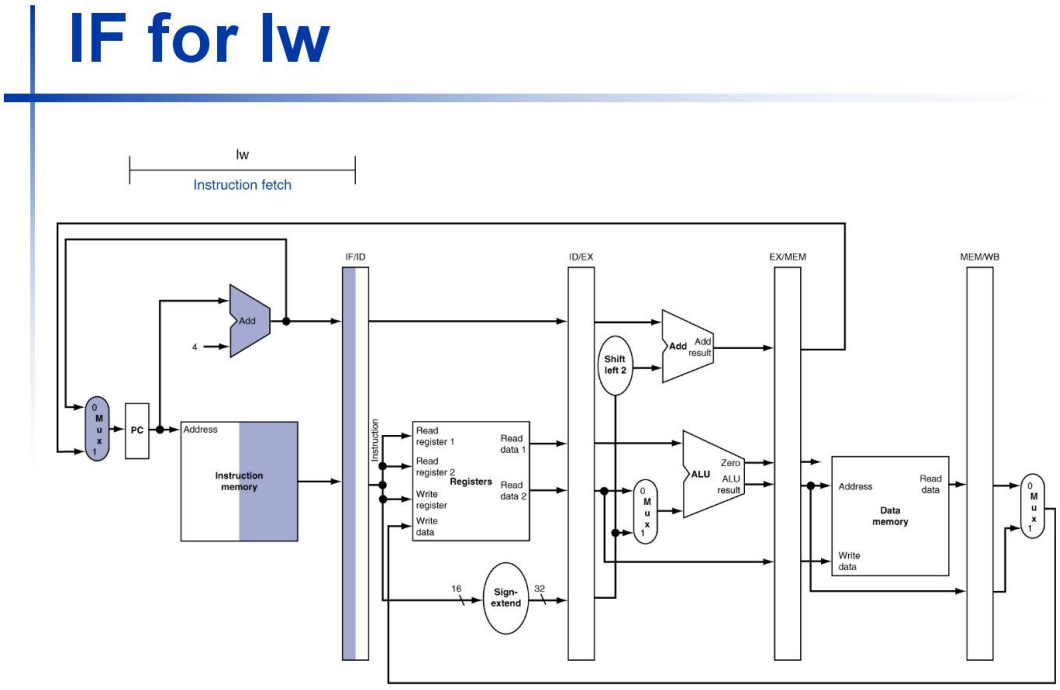


图 4　lw 第一阶段 取指阶段 该图片来自课件 PPT 截图

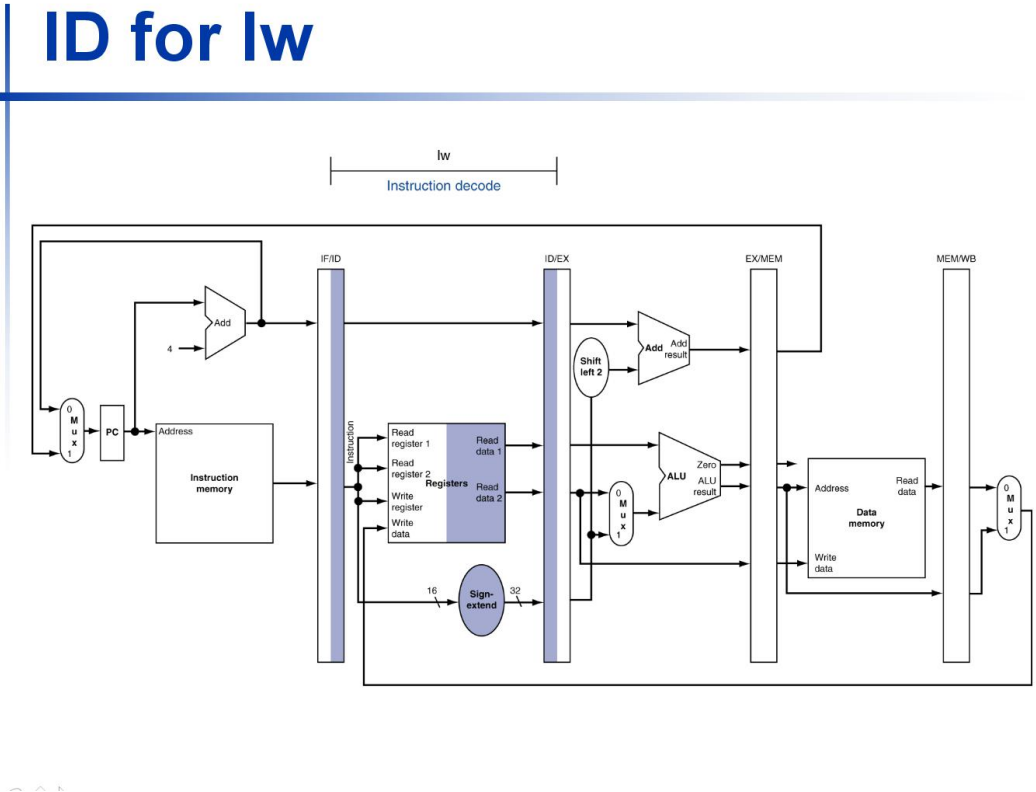


图 5　lw 的第二阶段 解码阶段 该图片来自课件 PPT 截图

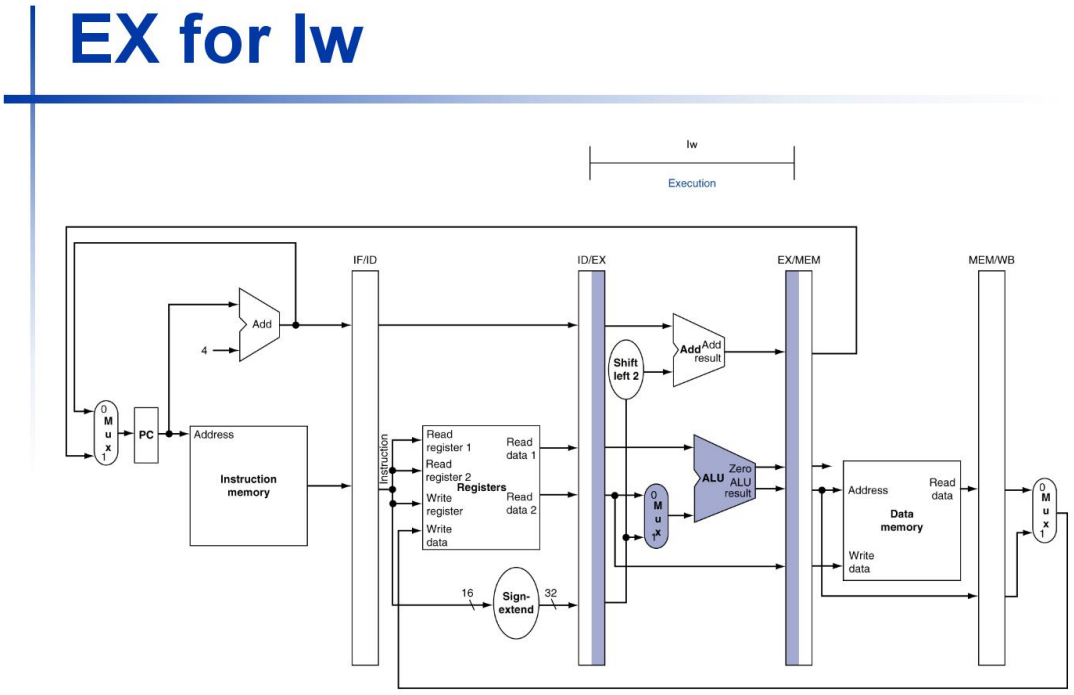


图 6　lw 的第三阶段 执行阶段 该图片来自课件 PPT 截图

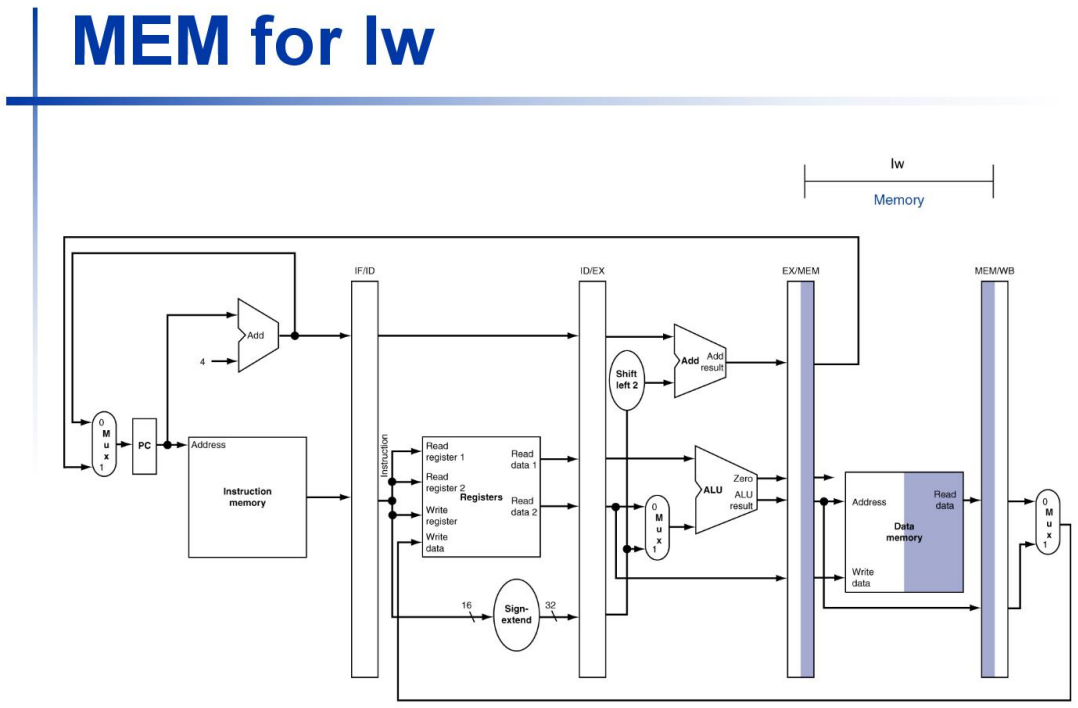


图 7　lw 的第四阶段　访存阶段 该图片来自课件 PPT 截图

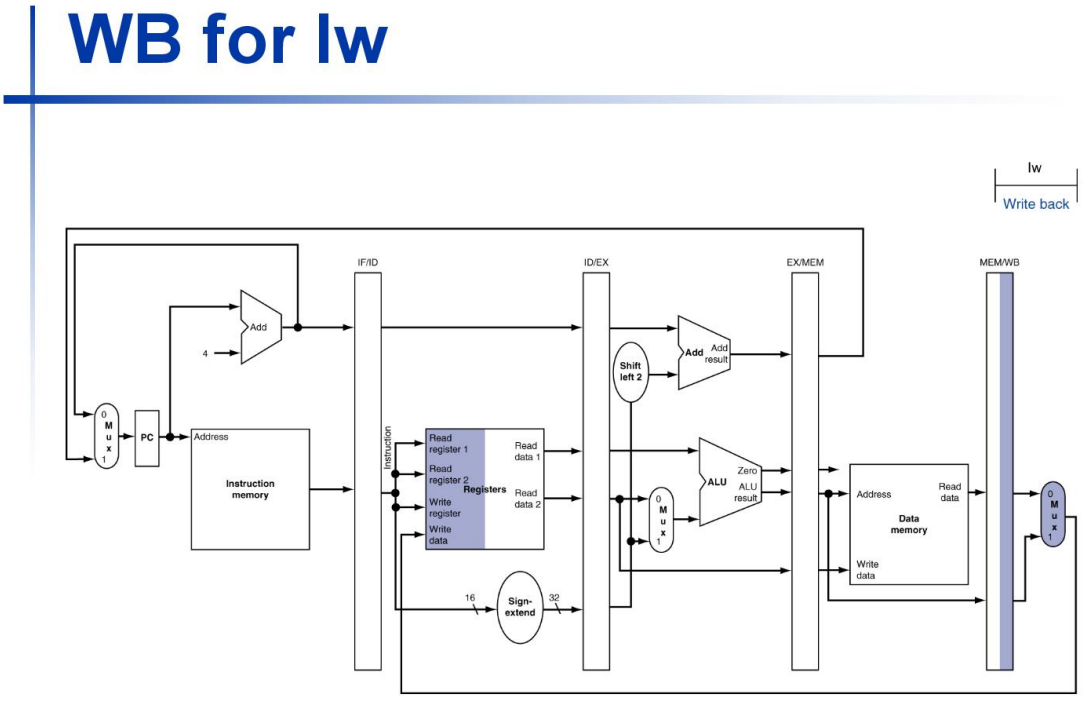


图 8　lw 第五阶段 写回阶段 该图片来自课件 PPT 截图

## 4.5 中断下的流水线

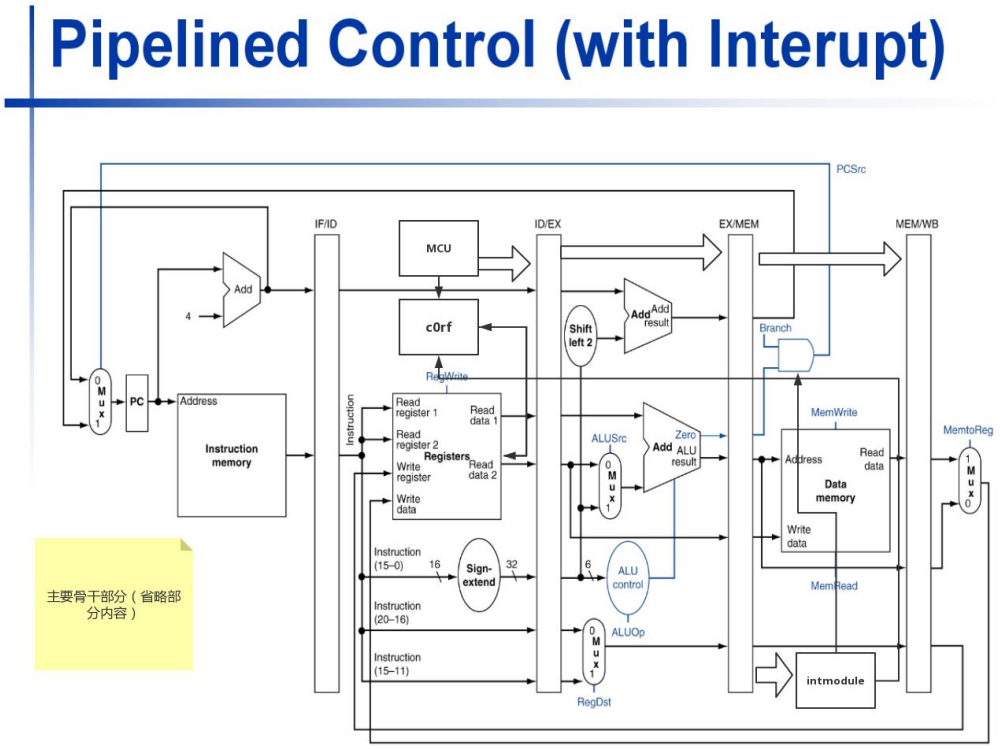


图 9　中断下的流水线总体结构

中断下的流水线总体结构如图 9 所示。在流水线基础上主要增加协处理器寄存器堆（c0rf）和中断异常决策模块（intmodule）。

## 4.6 模块设计

### 4.6.1 PC 模块

```
module PC(clk, pc_in, PCWrite, pc_out, rst); input clk,PCWrite,rst; input[31:0] pc_in;
output pc_out; wire[31:0] pc_out; reg[31:0] pc; always@(posedge clk , posedge rst)
begin if(rst==1) pc <= 32'h0000_3000; else if(PCWrite==1) pc <= pc_in; end
always@(negedge clk) begin $display("PC:%x",pc); end assign pc_out = pc; endmodule
```

4.6.2 RF 模块

```
module RF(raddr1, raddr2, waddr,rdata1, rdata2, wdata, RegWrite, clk,rst,testPC);
input[4:0] raddr1,raddr2,waddr;
input[31:0] testPC;
input[31:0] wdata;
input RegWrite,clk,rst;
output rdata1, rdata2;
wire[31:0] rdata1,rdata2;
reg[31:0] regfiles[31:0];
always @(posedge clk ,posedge rst, negedge clk)
if(rst==1)
begin
regfiles[0] <= 0;regfiles[1] <= 0;regfiles[2] <= 0;regfiles[3] <= 0;
regfiles[4] <= 0;regfiles[5] <= 0;regfiles[6] <= 0;regfiles[7] <= 0;
regfiles[8] <= 0;regfiles[9] <= 0;regfiles[10] <= 0;regfiles[11] <= 0;
regfiles[12] <= 0;regfiles[13] <= 0;regfiles[14] <= 0;regfiles[15] <= 0;
regfiles[16] <= 0;regfiles[17] <= 0;regfiles[18] <= 0;regfiles[19] <= 0;
regfiles[20] <= 0;regfiles[21] <= 0;regfiles[22] <= 0;regfiles[23] <= 0;
regfiles[24] <= 0;regfiles[25] <= 0;regfiles[26] <= 0;regfiles[27] <= 0;
regfiles[28] <= 0;regfiles[29] <= 0;regfiles[30] <= 0;regfiles[31] <= 0;
end
else if(RegWrite == 1 && waddr[4:0]!=0)
begin
regfiles[waddr[4:0]] <= wdata;
end
reg[31:0] rd1,rd2;
always@(*)
begin
  if(RegWrite==1 && waddr==raddr1 && waddr==raddr2){rd1,rd2}={wdata,wdata};
  else if(RegWrite==1 && waddr==raddr1 && waddr!=raddr2){rd1,rd2}={wdata,regfiles[raddr2[4:0]]};
  else if(RegWrite==1 && waddr!=raddr1 && waddr==raddr2){rd1,rd2}={regfiles[raddr1[4:0]],wdata};
  else {rd1,rd2}={regfiles[raddr1[4:0]],regfiles[raddr2[4:0]]};
end
assign rdata1 = rd1;
assign rdata2 = rd2;
//assign rdata1 = regfiles[raddr1[4:0]];
//assign rdata2 = regfiles[raddr2[4:0]];
//test
always @(negedge clk)
  begin
    if(rst!=1)
      begin
```

```
$display("0-%x 1-%x 2-%x 3-%x 4-%x 5-%x 6-%x 7-%x
",regfiles[0],regfiles[1],regfiles[2],regfiles[3],regfiles
[4],regfiles[5],regfiles[6],regfiles[7]);
$display("8-%x 9-%x 10-%x 11-%x 12-%x 13-%x 14-%x 15-%x
",regfiles[8],regfiles[9],regfiles[10],regfiles[11],regfil
es[12],regfiles[13],regfiles[14],regfiles[15]);
$display("16-%x 17-%x 18-%x 19-%x 20-%x 21-%x 22-%x 23-%x
",regfiles[16],regfiles[17],regfiles[18],regfiles[19],regf
iles[20],regfiles[21],regfiles[22],regfiles[23]);
$display("24-%x 25-%x 26-%x 27-%x 28-%x 29-%x 30-%x 31-%x
",regfiles[24],regfiles[25],regfiles[26],regfiles[27],regf
iles[28],regfiles[29],regfiles[30],regfiles[31]);
$display("------------------------------RF--%x------REGWr
ite-%x--data-%x--------PC:%x----------------------",waddr,
RegWrite,wdata,testPC);        end    end    endmodule
```

### 4.6.3 MCU 模块

```
// Mian Control Unit  `include "ALUOP_Def.v"
`include "Instr_Def.v"  module MCU(instr, WB, M, EX, EXCE);
    input[31:0] instr;  output[34:0] EXCE;  output
WB,M,EX;  wire[4:0] WB;  wire[9:0] M;  wire[9:0] EX;
        //WB  reg RegWrite;  reg MemToReg;  reg[2:0]
SLSE;  //M  reg[2:0] BranchSign;  reg[1:0] JMSign;
reg DVSign;  reg MemRead;  reg MemWrite;  reg[1:0]
SaveSign;  //EX  reg ALUSrcA;  reg[1:0] ALUSrcB;
reg[4:0] ALUOP;  reg[1:0] RegDst;  //EXCE
{mfc0,mtc0,eret,INT32} = {34,33,32,31:0}  wire
mfc0,mtc0,eret,syscall;  reg Unimplent;  wire[31:0]
INT32;  assign eret = instr==32'h42000018;  assign mfc0
= instr[31:21]==11'd512;  assign mtc0 =
instr[31:21]==11'd516;  assign syscall = instr==32'd12;
assign INT32 = {29'd0,Unimplent,syscall,1'd0};
wire[5:0] idcode = instr[31:26];  wire[5:0] funccode =
instr[5:0];  wire RTBit = instr[16];  reg[2:0]
branchTmp;  reg[5:0] jalrjr;//Note  //set WB,M,EX
assign WB = {RegWrite,MemToReg,SLSE};  assign M =
{jalrjr,MemRead,MemWrite,SaveSign};  assign EX =
{ALUSrcA,ALUSrcB,ALUOP,RegDst};  assign EXCE =
{mfc0,mtc0,eret,INT32};  always@(*)  begin
case(idcode)  `Instr_LB,  `Instr_LH,
`Instr_LW,  `Instr_LBU,  `Instr_LHU:
begin  RegWrite = 1;//WB  MemToReg =
1;//WB  SLSE = idcode[2:0];//WB
BranchSign = 3'b000;//M  JMSign = 2'b00;//M
DVSign = 0;//M  MemRead = 1;//M  MemWrite
= 0;//M  SaveSign = 2'b00;//M  ALUSrcA =
0;//EX  ALUSrcB = 2'b10;//EX  ALUOP =
`ALUOP_Load;//EX  RegDst = 2'b00;//EX
Unimplent = 1'd0;  end  `Instr_SB,
`Instr_SH,  `Instr_SW:  begin
RegWrite = 0;//WB  MemToReg = 0;//WB  SLSE
```

```
= idcode[2:0];//WB            BranchSign = 3'b000;//M
JMSign = 2'b00;//M            DVSign = 0;//M
MemRead = 0;//M            MemWrite = 1;//M
SaveSign = idcode[1:0];//M            ALUSrcA = 0;//EX
ALUSrcB = 2'b10;//EX            ALUOP = `ALUOP_Store;//EX
RegDst = 2'b00;//EX            Unimplent = 1'd0;            end
                `Instr_R_Type:            begin
RegWrite = 1;//WB            MemToReg = 0;//WB            SLSE
= idcode[2:0];//WB            BranchSign = 3'b000;//M
JMSign = 2'b00;//M            DVSign = 0;//M
MemRead = 0;//M            MemWrite = 0;//M
SaveSign = 2'b00;//M            ALUSrcA =
~(funccode[5]|funccode[4]|funccode[3]|funccode[2]);//EX
ALUSrcB = 2'b00;//EX            ALUOP = `ALUOP_R_Type;//EX
RegDst = 2'b01;//EX            Unimplent = 1'd0;            end
            `Instr_MFC0,        `Instr_MTC0,
`Instr_ERET:            begin            RegWrite =
instr[25:21]==5'd0;//WB            MemToReg = 0;//WB
SLSE = idcode[2:0];//WB            BranchSign = 3'b000;//M
JMSign = 2'b00;//M            DVSign = 0;//M
MemRead = 0;//M            MemWrite = 0;//M
SaveSign = 2'b00;//M            ALUSrcA =
~(funccode[5]|funccode[4]|funccode[3]|funccode[2]);//EX
ALUSrcB = 2'b00;//EX            ALUOP = `ALUOP_R_Type;//EX
RegDst = 2'b00;//EX            Unimplent = 1'd0;        end
            `Instr_ADDI,        `Instr_ADDUI,
`Instr_SLTI,        `Instr_SLTUI,        `Instr_ANDI,
`Instr_XORI,        `Instr_ORI,        `Instr_LUI:
begin            RegWrite = 1;//WB            MemToReg =
0;//WB            SLSE = idcode[2:0];//WB
BranchSign = 3'b000;//M            JMSign = 2'b00;//M
DVSign = 0;//M            MemRead = 0;//M            MemWrite
= 0;//M            SaveSign = 2'b00;//M            ALUSrcA =
0;//EX            ALUSrcB =
```

```
{1'b1,{1{~(idcode==6'd8||idcode==6'd10||idcode==6'd15||idc
ode==6'd9||idcode==6'd11)}}};//EX            ALUOP =
idcode[4:0];//EX            RegDst = 2'b00;//EX
Unimplent = 1'd0;          end                    `Instr_BEQ,
`Instr_BNE,          `Instr_BLEZ,          `Instr_BGTZ,
`Instr_BLTZ,          `Instr_BGEZ:          begin
RegWrite = 0;//WB            MemToReg = 0;//WB            SLSE
= idcode[2:0];//WB            BranchSign = branchTmp;//M
JMSign = 2'b00;//M            DVSign = 0;//M
MemRead = 0;//M            MemWrite = 0;//M
SaveSign = 2'b00;//M            ALUSrcA = 0;//EX
ALUSrcB = {1'b0,{1{~(idcode==6'd4 || idcode==6'd5)}}};//EX
          if(RTBit==1 && idcode==`Instr_BGEZ)//EX
ALUOP = `ALUOP_Bgez;            else            ALUOP =
idcode[4:0];            RegDst = 2'b00;//EX
Unimplent = 1'd0;          end                    `Instr_J:
begin            RegWrite = 0;//WB            MemToReg =
0;//WB            SLSE = idcode[2:0];//WB
BranchSign = branchTmp;//M            JMSign = 2'b01;//M
DVSign = 0;//M            MemRead = 0;//M            MemWrite
= 0;//M            SaveSign = 2'b00;//M            ALUSrcA =
0;//EX            ALUSrcB = 0;//EX            ALUOP =
`ALUOP_J;//EX            RegDst = 2'b00;//EX
Unimplent = 1'd0;          end                    `Instr_JAL:
          begin            RegWrite = 1;//WB
MemToReg = 0;//WB            SLSE = idcode[2:0];//WB
BranchSign = branchTmp;//M            JMSign = 2'b01;//M
DVSign = 1;//M            MemRead = 0;//M            MemWrite
= 0;//M            SaveSign = 2'b00;//M            ALUSrcA =
0;//EX            ALUSrcB = 0;//EX            ALUOP =
`ALUOP_JAL;//EX            RegDst = 2'b10;//EX
Unimplent = 1'd0;          end                default:
begin            RegWrite = 0;//WB            MemToReg =
0;//WB            SLSE = idcode[2:0];//WB
```

```
BranchSign = 3'b000;//M JMSign = 2'b00;//M
DVSign = 0;//M MemRead = 0;//M MemWrite
= 0;//M SaveSign = 2'b00;//M ALUSrcA =
~(funccode[5]|funccode[4]|funccode[3]|funccode[2]);//EX
ALUSrcB = 2'b00;//EX ALUOP = `ALUOP_R_Type;//EX
RegDst = 2'b01;//EX Unimplent = 1'd1; end
endcase end always@(*) begin case(idcode)
`Instr_BEQ: branchTmp = 3'b100;
`Instr_BNE: branchTmp =3'b101; `Instr_BLEZ,
`Instr_BGTZ, `Instr_BLTZ,
`Instr_BGEZ: branchTmp = 3'b110; `Instr_J,
`Instr_JAL, `Instr_JALR, `Instr_JR:
branchTmp = 3'b111; default:
branchTmp = 3'b000; endcase end always@(*)
begin //include Instr_JALR if(idcode==`Instr_JALR
&& (funccode==6'd8||funccode==6'd9)) //Branch JM DV
3.2.1 begin jalrjr = 6'b111_10_1;
//$display("JALR-%x",jalrjr); end else
begin jalrjr = {BranchSign,JMSign,DVSign};
//$display("JALR-%x",jalrjr); end end
endmodule
```

### 4.6.4 IM 模块

```
module IM(address, instr, clk); input[31:0] address; input clk; output instr; wire[31:0] instr; reg[7:0]
instrM[16383:0]; //debug initial begin
{instrM[12291],instrM[12290],instrM[12289],instrM[12288]}=32'h08000c06;
{instrM[12295],instrM[12294],instrM[12293],instrM[12292]}=32'h00000000;
{instrM[12299],instrM[12298],instrM[12297],instrM[12296]}=32'h08000c1e;
{instrM[12303],instrM[12302],instrM[12301],instrM[12300]}=32'h08000c0e;
{instrM[12307],instrM[12306],instrM[12305],instrM[12304]}=32'h00000000;
{instrM[12311],instrM[12310],instrM[12309],instrM[12308]}=32'h00000000;
{instrM[12315],instrM[12314],instrM[12313],instrM[12312]}=32'h34140056; end assign instr =
{instrM[address+3],instrM[address+2],instrM[address+1],instrM[address]}; endmodule
```

### 4.6.5 Branch 模块

```
module Branch(bs, zero, alurs, jrs); input zero,alurs; input[2:0] bs; output jrs; wire jrs; wire b = bs[1]; wire s = bs[0]; assign jrs = ((zero&(~b)&(~s))|((~zero)&s)|(b&s)|(b&alurs))&bs[2]; endmodule
```

### 4.6.6 ZE 模块

```
module ZE(ze_in, ze_out); input[15:0] ze_in; output ze_out; wire[31:0] ze_out; assign ze_out = {16'd0,ze_in}; endmodule
```

### 4.6.7 SE 模块

```
module SE(se_in, se_out); input[15:0] se_in; output se_out; wire[31:0] se_out; assign se_out = {{16{se_in[15]}},se_in}; endmodule
```

### 4.6.8 SL2 模块

```
module SL2(sl_in, sl_out); input[31:0] sl_in; output sl_out; wire[31:0] sl_out; assign sl_out = {sl_in[29:0],2'b00}; endmodule
```

### 4.6.9 SL226 模块

```
//shift left 2,plus pc high 4 bits = 32; module SL226(sl226_in, pcin, sl226_out); input[25:0] sl226_in; wire[25:0] sl226_in; input[31:0] pcin; output sl226_out; wire[31:0] sl226_out; assign sl226_out = {pcin[31:28],sl226_in,{2'b00}}; endmodule
```

### 4.6.10 Erasure 清洗流水线模块

```
//Erasure when predict wrong module Erasure(isj, rst1, rst2, rst3, clk, rst); input isj,clk,rst; output rst1,rst2,rst3; wire rst1,rst2,rst3; assign {rst1,rst2,rst3} = {3{isj|rst}}; endmodule
```

### 4.6.11 HDU 模块

```
`include "Instr_Def.v" //Hazard detect unit module HDU(instr,rs, rt, rd, MMR, rst2, PCWriteA, IFIDW); input[4:0] rs,rt,rd; input MMR; input[5:0] instr; output rst2,PCWriteA,IFIDW; wire rst2,PCWriteA,IFIDW; reg[2:0] res; always@(*) begin if(MMR==1 && (rs==rd || rt==rd) && instr==`Instr_LW) begin res = 3'b110; end else begin res = 3'b001; end end assign {rst2,PCWriteA,IFIDW} = res; endmodule
```

### 4.6.12 DM 模块

```
module DM(address, dm_in, dm_out, MemWrite, MemRead, SBHW, clk);
    input[31:0] address,dm_in;    input[1:0] SBHW;    input
MemWrite, MemRead, clk;    output dm_out;        wire[31:0]
dm_out;        reg[7:0] datamemory[2047:0];        always
@(posedge clk)    begin     if(MemWrite==1 && SBHW==2'b11)
{datamemory[address[31:0]+3],datamemory[address[31:0]+2],datam
emory[address[31:0]+1],datamemory[address[31:0]]} <= dm_in;
if(MemWrite==1 && SBHW==2'b01)
{datamemory[address[31:0]+1],datamemory[address[31:0]]} <=
dm_in[15:0];     if(MemWrite==1 && SBHW==2'b00)
datamemory[address[31:0]] <= dm_in[7:0];     end
assign dm_out =
{32{MemRead}}&{datamemory[address[31:0]+3],datamemory[address[
31:0]+2],datamemory[address[31:0]+1],datamemory[address[31:0]]}
;       endmodule
```

### 4.6.13 ALU 模块

```
`include "ALUOPCtrl_Def.v" module ALU(srcA, srcB, ALUOPCtrl, zero, ovf, aluout); input[31:0] srcA,srcB; input[4:0] ALUOPCtrl; output zero,ovf,aluout; wire zero,ovf; wire[31:0] aluout; //zero,ovf,32bitsOfAluout total 34bits reg[33:0] arithmetic; reg[33:0] shift; reg[33:0] logic; reg[33:0] lui; reg[33:0] comparison; reg[33:0] slt; reg[33:0] result; ////////////////////////////////////////////// always@(*) begin case(ALUOPCtrl) `ALU_Add, `ALU_Addu, `ALU_Sub, `ALU_Subu: result = arithmetic; `ALU_Sll, `ALU_Srl, `ALU_Sra: result = shift; `ALU_And, `ALU_Or, `ALU_Xor, `ALU_Nor: result = logic; `ALU_Slt, `ALU_Sltu: result = slt; `ALU_Lui: result = lui; `ALU_Le, `ALU_Ge, `ALU_Gt: result = comparison; default: result = 34'd0; endcase end assign {zero,ovf,aluout} = result; ////////////////////////////////////////////// //arithmetic wire imsign = srcA[31]^srcB[31]; wire[32:0] sum = srcA+srcB; wire[32:0] minus = {1'b0,srcA}+{1'b0,~srcB}+{31'd1}; wire tmp = sum[31]^srcA[31]; wire tmp1 = minus[31]^srcA[31]; wire NzeroS = (|sum[31:0]); wire NzeroM = (|minus[31:0]); wire ovf0 = (~imsign)&tmp; wire ovf1 = imsign&tmp1; wire ovf2 = (|srcB)&(~minus[32]); wire[33:0] myadd = {~(ovf0|NzeroS),ovf0,sum[31:0]}; wire[33:0] myaddu = {~(NzeroS|sum[32]),sum[32],sum[31:0]}; wire[33:0] mysub = {~(ovf1|NzeroM),ovf1,minus[31:0]}; wire[33:0] mysubu = {~(ovf2|NzeroM),ovf2,minus[31:0]}; always@(*) begin case(ALUOPCtrl) `ALU_Add:
```

```
arithmetic = myadd;          `ALU_Addu:          arithmetic =
myaddu;          `ALU_Sub:          arithmetic = mysub;
`ALU_Subu:          arithmetic = mysubu;          default:
arithmetic = 32'd0;       endcase     end       //shift
wire[31:0] mysll = srcB<<srcA[4:0];     wire[31:0] mysrl =
srcB>>srcA[4:0];     wire[31:0] myshift =
~({32{srcB[31]}}>>(srcA[4:0]));     wire[31:0] mysra =
({32{~srcB[31]}}&mysrl)|({32{srcB[31]}}&(mysrl|myshift)) ;
    always@(*)     begin       case(ALUOPCtrl)
`ALU_Sll:          shift[31:0] = mysll;          `ALU_Srl:
shift[31:0] = mysrl;          `ALU_Sra:          shift[31:0] =
mysra;          default:          shift[31:0] = 32'd0;
endcase       shift[33:32] = {~(|shift[31:0]),1'b0};     end
        //logic     always@(*)     begin       case(ALUOPCtrl)
        `ALU_And:          logic[31:0] = srcA & srcB;
`ALU_Or:          logic[31:0] = srcA | srcB;          `ALU_Xor:
          logic[31:0] = srcA ^ srcB;          `ALU_Nor:
logic[31:0] = ~(srcA | srcB);          default:
logic[31:0] = 32'd0;       endcase       logic[33:32] =
{~(|logic[31:0]),1'b0};     end        //slt     wire[31:0]
tmpll = {32{((~imsign)&srcA[31])}};     wire[31:0] A =
(tmpll&(~srcA)) | ((~tmpll) & srcA);     wire[31:0] B =
(tmpll&(~srcB)) | ((~tmpll) & srcB);     wire C = A<B?1'd1:1'd0;
    wire myslttmp = (imsign&srcA[31])|((~imsign)&C);
wire[31:0] myslt = {31'd0,myslttmp};          always@(*)
begin       case(ALUOPCtrl)          `ALU_Slt:
slt[31:0] = myslt;          `ALU_Sltu:          slt[31:0] =
srcA < srcB;          default:          slt[31:0] = 32'd0;
endcase       slt[33:32] = 2'b0;     end        //lui
always@(*)     begin       case(ALUOPCtrl)          `ALU_Lui:
          lui[31:0] = srcB<<16;          default:
lui[31:0] = 32'd0;       endcase       lui[33:32] =
{~(|lui[31:0]),1'b0};     end
//////////////////////////////////     //comparison
```

```
wire compar = myslt[0]|mysub[33];    always@(*)    begin
case(ALUOPCtrl)          `ALU_Le:          comparison =
{31'd0,compar};          `ALU_Gt:          comparison =
{31'd0,~compar};          `ALU_Ge:          comparison =
{31'd0,(~myslt[0])};          default:
comparison[31:0] = 32'd0;       endcase
comparison[33:32] = 2'b0;    end      endmodule
```

### 4.6.14 ALUCU 模块

```
`include "ALUOP_Def.v"  `include "ALUOPCtrl_Def.v"  `include "funccode_Def.v"  module ALUCU(clk,ALUOP,funccode,ALUOPCtrl);  input clk;  input[4:0] ALUOP;  input[5:0] funccode;  output ALUOPCtrl;  wire[4:0] ALUOPCtrl;  reg[4:0] tmp,Rtmp;  always@(*)  begin  case(funccode)  `Func_Add:  Rtmp = `ALU_Add;  `Func_Addu:  Rtmp = `ALU_Addu;  `Func_Sub:  Rtmp = `ALU_Sub;  `Func_Subu:  Rtmp = `ALU_Subu;  `Func_Sll:  Rtmp = `ALU_Sll;  `Func_Srl:  Rtmp = `ALU_Srl;  `Func_Sra:  Rtmp = `ALU_Sra;  `Func_Sllv:  Rtmp = `ALU_Sll;  `Func_Srlv:  Rtmp = `ALU_Srl;  `Func_Srav:  Rtmp = `ALU_Sra;  `Func_And:  Rtmp = `ALU_And;  `Func_Or:  Rtmp = `ALU_Or;  `Func_Xor:  Rtmp = `ALU_Xor;  `Func_Nor:  Rtmp = `ALU_Nor;  `Func_Slt:  Rtmp = `ALU_Slt;  `Func_Sltu:  Rtmp = `ALU_Sltu;  `Func_Jalr:  Rtmp = `ALU_Jalr;  `Func_Jr:  Rtmp = `ALU_Jr;  `Func_Else:  Rtmp = `ALU_Invalid;  default:  Rtmp = 6'b0;  endcase  end  always@(*)  begin  case(ALUOP)  `ALUOP_Addi,  `ALUOP_Load,  `ALUOP_Store:  tmp = `ALU_Add;  `ALUOP_Addiu:  tmp = `ALU_Addu;  `ALUOP_Slti:  tmp = `ALU_Slt;  `ALUOP_Sltiu:  tmp = `ALU_Sltu;  `ALUOP_Andi:  tmp = `ALU_And;  `ALUOP_Ori:  tmp = `ALU_Or;  `ALUOP_Xori:  tmp = `ALU_Xor;  `ALUOP_Lui:  tmp = `ALU_Lui;  `ALUOP_Beq,  `ALUOP_Bne:  tmp = `ALU_Subu;  `ALUOP_Blez:  tmp = `ALU_Le;  `ALUOP_Bgtz:  tmp = `ALU_Gt;  `ALUOP_Bltz:  tmp = `ALU_Slt;  `ALUOP_Bgez:  tmp = `ALU_Ge;  `ALUOP_R_Type:  tmp = Rtmp;  `ALUOP_Invalid:  tmp =
```

```
`ALU_Invalid;          default:          tmp = `ALU_Invalid;
        endcase        end                assign ALUOPCtrl =
tmp;          endmodule
```

### 4.6.15 C0rf 模块

```
module c0rf(input rst,  input clk,
          input INT,  input[31:0] wepc,
    input[31:0] wcause,  input[31:0] wstatus,
          input[4:0]  raddr,  input[4:0]  waddr,
          input[31:0] wdata,  input  c0w,
          input  back,  output[31:0] rdata,
          output[31:0] Status,  output[31:0] EPC,
          output[31:0] Cause);  reg[31:0]
regfiles[31:0];  reg[37:0] b1;  reg[37:0] b2;
always @(posedge clk ,posedge rst)  begin  if(rst==1)
       begin  regfiles[0]  <= 0;regfiles[1]  <=
0;regfiles[2] <= 0;regfiles[3]  <= 0;  regfiles[4] <=
0;regfiles[5] <= 0;regfiles[6]  <= 0;regfiles[7]  <= 0;
regfiles[8]  <= 0;regfiles[9]  <= 0;regfiles[10] <=
0;regfiles[11]  <= 0;  regfiles[12]  <=
0;regfiles[13]  <= 0;regfiles[14] <= 0;regfiles[15] <= 0;
       regfiles[16] <= 0;regfiles[17] <= 0;regfiles[18] <=
0;regfiles[19]  <= 0;  regfiles[20]  <=
0;regfiles[21]  <= 0;regfiles[22] <= 0;regfiles[23] <= 0;
       regfiles[24] <= 0;regfiles[25] <= 0;regfiles[26] <=
0;regfiles[27]  <= 0;  regfiles[28]  <=
0;regfiles[29]  <= 0;regfiles[30] <= 0;regfiles[31] <= 0;
    b1 <= 0; b2 <= 0;  end  else  begin
   if(INT==1)  begin  $display("wepc:%x
wcause:%x",wepc,wcause);  regfiles[14] <= wepc;
        regfiles[13] <= wcause;  regfiles[12] <=
wstatus;  //back  if(b1[0]==1 && b2[0]!=1)
         regfiles[b1[5:1]] <= b1[37:6];  else
if(b1[0]!=1 && b2[0]==1)  regfiles[b2[5:1]] <=
b2[37:6];  else if(b1[0]==1 && b2[0]==1)
   begin
if(b1[5:1]==b2[5:1])regfiles[b2[5:1]] <= b2[37:6];
    else  begin  regfiles[b1[5:1]]
<= b1[37:6];  regfiles[b2[5:1]] <= b2[37:6];
```

```
            end             end             else            ;
        //back          end         else if(back==1)
    begin           //back              if(b1[0]==1 && b2[0]!=1)
            regfiles[b1[5:1]] <= b1[37:6];              else
if(b1[0]!=1 && b2[0]==1)            regfiles[b2[5:1]] <=
b2[37:6];               else if(b1[0]==1 && b2[0]==1)
    begin
if(b1[5:1]==b2[5:1])regfiles[b2[5:1]] <= b2[37:6];
      else              begin             regfiles[b1[5:1]]
<= b1[37:6];                regfiles[b2[5:1]] <= b2[37:6];
            end             end             else            ;
        //back          end         else        begin
b2 = b1;          b1 = {regfiles[waddr],waddr,c0w};
if(c0w==1)              regfiles[waddr] <= wdata;       end
      end     end             assign Status = regfiles[12];
assign Cause = regfiles[13];     assign EPC = regfiles[14];
    assign rdata = regfiles[raddr];          //test     always
@(negedge clk)     begin       if(rst!=1)          begin
$display("--***************----back:%x----************--",
back);          $display("0-%x 1-%x 2-%x 3-%x 4-%x 5-%x 6-%x
7-%x
",regfiles[0],regfiles[1],regfiles[2],regfiles[3],regfiles
[4],regfiles[5],regfiles[6],regfiles[7]);
$display("8-%x 9-%x 10-%x 11-%x 12-%x 13-%x 14-%x 15-%x
",regfiles[8],regfiles[9],regfiles[10],regfiles[11],regfil
es[12],regfiles[13],regfiles[14],regfiles[15]);
$display("16-%x 17-%x 18-%x 19-%x 20-%x 21-%x 22-%x 23-%x
",regfiles[16],regfiles[17],regfiles[18],regfiles[19],regf
iles[20],regfiles[21],regfiles[22],regfiles[23]);
$display("24-%x 25-%x 26-%x 27-%x 28-%x 29-%x 30-%x 31-%x
",regfiles[24],regfiles[25],regfiles[26],regfiles[27],regf
iles[28],regfiles[29],regfiles[30],regfiles[31]);
    $display("b1:%x  b2:%x",b1,b2);
$display("---**********************----RF--%x------REGWrite
```

```
-%x--data-%x--*********-----",waddr,c0w,wdata);          end
    end    endmodule
```

### 4.6.16 Intmodule 模块

```
module intmodule(input[31:0] INT32,
                 input[31:0] Status,
input[31:0] epcin,                output[31:0] wepc,
           output[31:0] wcause,                output[31:0]
wstatus,              output      INT,                input
clk);          wire[31:0] valids = (~Status)&INT32;
    wire[31:0] tmp = valids&(~valids+32'd1);
    always@(negedge clk)     begin
    $display("helo:%x",tmp);     end     assign INT = |tmp;
    assign wepc = epcin;     assign wstatus = Status|tmp;
        reg[31:0] cau;     always@(*)     begin
    if(tmp==32'd1)cau=0;     else if(tmp==32'd2)cau=1;
    else if(tmp==32'd4)cau=2;     else if(tmp==32'd8)cau=3;
    else if(tmp==32'd16)cau=4;       else
if(tmp==32'd32)cau=5;     else if(tmp==32'd64)cau=6;
    else if(tmp==32'd128)cau=7;      else
if(tmp==32'd256)cau=8;    else if(tmp==32'd512)cau=9;
    else if(tmp==32'd1024)cau=10;    else
if(tmp==32'h800)cau=11;       else if(tmp==32'h1000)cau=12;
    else if(tmp==32'h2000)cau=13;    else
if(tmp==32'h4000)cau=14;      else if(tmp==32'h8000)cau=15;
    else if(tmp==32'h10000)cau=16;       else
if(tmp==32'h20000)cau=17;     else if(tmp==32'h40000)cau=18;
    else if(tmp==32'h80000)cau=19;       else
if(tmp==32'h100000)cau=20;       else
if(tmp==32'h200000)cau=21;       else
if(tmp==32'h400000)cau=22;       else
if(tmp==32'h800000)cau=23;       else
if(tmp==32'h1000000)cau=24;      else
if(tmp==32'h2000000)cau=25;      else
if(tmp==32'h4000000)cau=26;      else
if(tmp==32'h8000000)cau=27;      else
if(tmp==32'h10000000)cau=28;     else
if(tmp==32'h20000000)cau=29;     else
```

```
if(tmp==32'h40000000)cau=30;    else
if(tmp==32'h80000000)cau=31;    // There is too long....blur
blur blur     else cau=tmp;     end    assign wcause =
cau<<2;    endmodule
```

### 4.6.17 FU 模块

```
//forwarding unit module FU(rs,rt,exrd,exw,mrd,mw,fa,fb); input[4:0]
rs,rt,exrd,mrd; input exw,mw; output fa,fb; wire[1:0] fa,fb; reg[1:0] ffa,ffb; always@(*) begin
if(exw==1 && exrd==rs && rs!=5'd0) ffa = 2'b01; else if(mw==1 && mrd==rs && rs!=5'd0) ffa = 2'b10;
else ffa = 2'b00; if(exw==1 && exrd==rt && rt!=5'd0) ffb = 2'b01; else if(mw==1 && mrd==rt && rt!=5'd0)
ffb = 2'b10; else ffb = 2'b00; end assign fa = ffa; assign fb = ffb; endmodule
```

### 4.6.18 IFID 模块

```
module IFID(clk,IFIDW,rst1,pcin,instr,Opcin,Oinstr); input clk,IFIDW,rst1;
input[31:0] pcin,instr; output Opcin,Oinstr; wire[31:0] Opcin,Oinstr; reg[31:0] reg_pc,reg_instr;
always@(posedge clk) begin if(rst1==1) {reg_pc,reg_instr} <= 64'd0; if(rst1==0 && IFIDW==1)
{reg_pc,reg_instr} <= {pcin,instr}; end assign {Opcin,Oinstr} = {reg_pc,reg_instr}; endmodule
```

### 4.6.19 IDEX 模块 module

```
IDEX(clk,IDEXW,rst2,WB,M,EX,pcnext,rd1,rd2,zer,ser,rs,rt,r
d,low26,instr,EXCEREG_in,
OWB,OM,OEX,Opcnext,Ord1,Ord2,Ozer,Oser,Ors,Ort,Ord,Olow26,
Oinstr,OEXCEREG);    input clk,IDEXW,rst2;    input
WB,M,EX,pcnext,rd1,rd2,zer,ser,rs,rt,rd,low26,instr;
input EXCEREG_in;    wire[4:0] WB;    wire[9:0] M;
wire[9:0] EX;    wire[31:0] pcnext;    wire[31:0] rd1;
wire[31:0] rd2;    wire[31:0] zer;    wire[31:0] ser;
wire[4:0] rs;    wire[4:0] rt;    wire[4:0] rd;    wire[25:0]
low26;    wire[5:0] instr;    wire[128:0] EXCEREG_in;
output
OWB,OM,OEX,Opcnext,Ord1,Ord2,Ozer,Oser,Ors,Ort,Ord,Olow26,
Oinstr,OEXCEREG;    wire[4:0] OWB;    wire[9:0] OM;
wire[9:0] OEX;    wire[31:0] Opcnext;    wire[31:0] Ord1;
wire[31:0] Ord2;    wire[31:0] Ozer;    wire[31:0] Oser;
wire[4:0] Ors;    wire[4:0] Ort;    wire[4:0] Ord;
wire[25:0] Olow26;    wire[5:0] Oinstr;    wire[128:0]
OEXCEREG;        reg[24:0] reg_sign;    reg[31:0] reg_pc;
reg[31:0] reg_rd1,reg_rd2;    reg[31:0] reg_zer;
reg[31:0] reg_ser;    reg[31:0] reg_instr;    reg[128:0]
reg_excereg;        always@(posedge clk)    begin
if(rst2==1)
{reg_sign,reg_pc,reg_rd1,reg_rd2,reg_zer,reg_ser,reg_instr,
reg_excereg}        <=
{25'd0,32'd0,32'd0,32'd0,32'd0,32'd0,32'd0,EXCEREG_in};
if(rst2==0 && IDEXW==1)
//WB,M,EX,pcnext,rd1,rd2,zer,ser,rs,rt,rd,low26,
{reg_sign,reg_pc,reg_rd1,reg_rd2,reg_zer,reg_ser,reg_instr,
reg_excereg}        <=
{WB,M,EX,pcnext,rd1,rd2,zer,ser,instr,low26,EXCEREG_in};
end        assign
{OWB,OM,OEX,Opcnext,Ord1,Ord2,Ozer,Oser,Ors,Ort,Ord,Olow26,
Oinstr,OEXCEREG}        =
```

```
{reg_sign,reg_pc,reg_rd1,reg_rd2,reg_zer,reg_ser,reg_instr
[25:21],
reg_instr[20:16],reg_instr[15:11],reg_instr[25:0],reg_inst
r[31:26],reg_excereg};                    endmodule
```

### 4.6.20 EXMEM 模块 module

```
EXMEM(clk,EXMEMW,rst3,WB,M,Jaddr,addres,rd1,zero,aluout,rd
2,dst,pc,EXMEM_in,
OWB,OM,OJaddr,Oaddres,Ord1,Ozero,Oaluout,Ord2,Odst,Opc,OEX
MEM);      input clk,EXMEMW,rst3;      input
WB,M,Jaddr,addres,rd1,zero,aluout,rd2,dst,pc;     input
EXMEM_in;      wire[4:0] WB;      wire[9:0] M;
wire[31:0] Jaddr;      wire[31:0] addres;      wire[31:0]
rd1;      wire zero;      wire[31:0] aluout;      wire[31:0]
rd2;      wire[4:0] dst;      wire[31:0] pc;
   wire[128:0] EXMEM_in;      output
OWB,OM,OJaddr,Oaddres,Ord1,Ozero,Oaluout,Ord2,Odst,Opc,OEX
MEM;      wire[4:0] OWB;      wire[9:0] OM;      wire[31:0]
OJaddr;      wire[31:0] Oaddres;      wire[31:0] Ord1;
wire Ozero;      wire[31:0] Oaluout;      wire[31:0] Ord2;
     wire[4:0] Odst;      wire[31:0] Opc;      wire[128:0]
OEXMEM;          reg[212:0] reg_reg;      reg[128:0]
reg_EXMEM;          always@(posedge clk)      begin
if(rst3==1)          {reg_reg,reg_EXMEM} <= {213'd0,129'd0};
       if(rst3==0 && EXMEMW==1)
{reg_reg,reg_EXMEM} <=
{WB,M,Jaddr,addres,rd1,zero,aluout,rd2,dst,pc,EXMEM_in};
end          assign
{OWB,OM,OJaddr,Oaddres,Ord1,Ozero,Oaluout,Ord2,Odst,Opc,OE
XMEM} = {reg_reg,reg_EXMEM};          endmodule
```

### 4.6.21 MEMEWB 模块 module

MEMWB(clk,MEMWBW,rst4,WB,Rdfm,aluout,dst,OWB,ORdfm,Oaluout,Odst); input clk,rst4,MEMWBW; input WB,Rdfm,aluout,dst; wire[4:0] WB; wire[31:0] Rdfm; wire[31:0] aluout; wire[4:0] dst; output OWB,ORdfm,Oaluout,Odst; wire[4:0] OWB; wire[31:0] ORdfm; wire[31:0] Oaluout; wire[4:0] Odst; reg[73:0] reg_reg; always@(posedge clk) begin if(rst4==1) reg_reg <= 74'd0; if(rst4==0 && MEMWBW==1) reg_reg <= {WB,Rdfm,aluout,dst}; end assign {OWB,ORdfm,Oaluout,Odst} = reg_reg; endmodule

### 4.6.22 Mux1Bits 模块 module Mux1Bits(d1, d2, dout, s); parameter width=32; input[width:1] d1,d2; input s; output dout; wire[width:1] dout; wire[width:1] exs = {width{s}}; assign dout = ((~exs)&d1)|(exs&d2); endmodule

### 4.6.23 Mux2Bits 模块 module Mux2Bits(m0,m1,m2,m3,mout,ctrl); parameter width=32; input[width:1] m0,m1,m2,m3; input[1:0] ctrl; output mout; wire[width:1] mout; wire[width:1] tmp1,tmp2; assign tmp1 = (m0&{width{~ctrl[0]}}) | (m1&({width{ctrl[0]}})); assign tmp2 = (m2&{width{~ctrl[0]}}) | (m3&({width{ctrl[0]}})); assign mout = (tmp1&{width{~ctrl[1]}}) | (tmp2&({width{ctrl[1]}})); endmodule

### 4.6.24 Mux3Bits 模块 module

```
Mux3Bits(m0,m1,m2,m3,m4,m5,m6,m7,mout,ctrl);    parameter
width=32;    input[width:1] m0,m1,m2,m3,m4,m5,m6,m7;
input[2:0] ctrl;    output mout;        wire[width:1] mout;
    wire[width:1] tmp1,tmp2,tmp3,tmp4;    wire[width:1]
tmp5,tmp6;        assign tmp1 = (m0&{width{~ctrl[0]}}) |
(m1&({width{ctrl[0]}}));    assign tmp2 =
(m2&{width{~ctrl[0]}}) | (m3&({width{ctrl[0]}}));    assign
tmp3 = (m4&{width{~ctrl[0]}}) | (m5&({width{ctrl[0]}}));
assign tmp4 = (m6&{width{~ctrl[0]}}) |
(m7&({width{ctrl[0]}}));    assign tmp5 =
(tmp1&{width{~ctrl[1]}}) | (tmp2&({width{ctrl[1]}}));
assign tmp6 = (tmp3&{width{~ctrl[1]}}) |
(tmp4&({width{ctrl[1]}}));    assign mout =
(tmp5&{width{~ctrl[2]}}) | (tmp6&({width{ctrl[2]}}));
endmodule
```

# 第 5 章 测试和结果分析

## 5.1 测试文件 testfinal.asm

```
1.          j   start
2.          add $zero,$zero,$zero
3.          add $zero,$zero,$zero
4.          j   sysproc
5.          add $zero,$zero,$zero
6.          add $zero,$zero,$zero
7.          add $zero,$zero,$zero
8.          add $zero,$zero,$zero
9.    start:  nor $1,$0,$0    #r1=0xFFFFFFFF
10.         add $3,$1,$1    #r3=0xFFFFFFFE
11.         add $3,$3,$3    #r3=0xFFFFFFFC
12.         add $3,$3,$3    #r3=0xFFFFFFF8
13.         add $3,$3,$3    #r3=0xFFFFFFF0
14.         add $3,$3,$3    #r3=0xFFFFFFE0
15.         add $3,$3,$3    #r3=0xFFFFFFC0
16.         nor $20,$3,$0   #r20=0x0000003F
17.         add $3,$3,$3    #r3=0xFFFFFF80
18.         add $3,$3,$3    #r3=0xFFFFFF00
19.         add $3,$3,$3    #r3=0xFFFFFE00
20.         add $3,$3,$3    #r3=0xFFFFFC00
21.         add $3,$3,$3    #r3=0xFFFFF800
22.         add $3,$3,$3    #r3=0xFFFFF000
23.         add $3,$3,$3    #r3=0xFFFFE000
24.         add $3,$3,$3    #r3=0xFFFFC000
25.         add $3,$3,$3    #r3=0xFFFF8000
26.         add $3,$3,$3    #r3=0xFFFF0000
27.         add $3,$3,$3    #r3=0xFFFE0000
28.         add $3,$3,$3    #r3=0xFFFC0000
29.         add $3,$3,$3    #r3=0xFFF80000
30.         add $3,$3,$3    #r3=0xFFF00000
31.         add $3,$3,$3    #r3=0xFFE00000
32.         add $3,$3,$3    #r3=0xFFC00000
33.         add $3,$3,$3    #r3=0xFF800000
34.         add $3,$3,$3    #r3=0xFF000000
35.         add $3,$3,$3    #r3=0xFE000000
36.         add $3,$3,$3    #r3=0xFC000000
37.         add $6,$3,$3    #r6=0xF8000000
```

```
        add $3,$6,$6    #r3=0xF0000000
        add $4,$3,$3    #r4=0xE0000000
        add $13,$4,$4   #r13=0xC0000000
        add $8,$13,$13  #r8=0x80000000
        ori $27,$0,0x0
        sltu    $2,$0,$1    #r2=0x00000001 unsigned slt
        add $14,$2,$2   #r14=0x2
        add $14,$14,$14 #r14=0x4
        nor $10,$0,$0   #r10=0xFFFFFFFF
        add $10,$10,$10 #r10=0xFFFFFFFE
        sw  $6,4($3)    #counter port:f0000004,r6=0xF8000000
        lw  $5,0($3)    #{counter0_out,counter1_out,counter2_out,led_out[12:0], SW};
        add $5,$5,$5
        add $5,$5,$5
        sw  $5,0($3)    #{GPIOf0[13:0],LED,counter_set}, port:f0000000
        add $9,$9,$2    #r9 uninitilized, r9 = 1
        sw  $9,0($4)    #r9ËÍr4=0xE0000000ÆβÍÎÂëÍË¿Ú
        lw  $13,0x14($0)    #r13=0xFFF7000
loop:   lw  $5,0($3)    #{counter0_out,counter1_out,counter2_out,led_out[12:0], SW}
        add $5,$5,$5
        add $5,$5,$5
        sw  $5,0($3)    #{GPIOf0[13:0],LED,counter_set}, port:f0000000
        lw  $5,0($3)    #{counter0_out,counter1_out,counter2_out,led_out[12:0], SW}
        and $11,$5,$8   #È¡r5×î¸ßÎ»
        add $13,$13,$2  #r13=0xFFF7001
        beq $13,$0,next
Disp:   lw  $5,0($3)
        add $30,$14,$14 #r18=0x8
        add $18,$14,$14
        add $22,$18,$18 #r22=0x10
        add $18,$18,$22 #r18=0x18
        and $11,$5,$18
        or  $11,$11,$27
        syscall
        beq $11,$0,L00  #SW[4:3]=0x00,ÒÆÎ»
        beq $11,$18,L11 #SW[4:3]=0x11£¬ïÔÊ¾ÆβÍÍ¼ÐÎ
        beq $11,$30,L01 #SW[4:3]=0x01,ïÔÊ¾7ÍÎÔ¤ÖÃÊý×Ö
        sw  $9,0($4)    #SW[4:3]=0x10£¬ïÔÊ¾r9
        j   loop
L00:    beq $10,$1,L4   #r1=0xFFFFFFFF
        j   L3
L4:     nor $10,$0,$0   #r10=0xFFFFFFFF
```

```
        add $10,$10,$10 #r10=0xFFFFFFFE
L3:     sw  $10,0($4)   #7¶ÎÍ¼ÐÎÏÔÊ¾r10
        j   loop
L11:    lw  $9,0x60($17)
        sw  $9,0($4)    #7¶ÎÍ¼ÐÎÏÔÊ¾$9
        j   loop
L01:    lw  $9,0x20($17)
        sw  $9,0($4)    #7¶ÎÎÄ±¾ÏÔÊ¾S9
        j   loop
next:   lw  $13,0x14($0)    #r13=0xFFF7000
        add $10, $10, $10
        or  $10, $10, $2
        add $17, $17, $14   #·Ã´æµØÖ·¼Ó4
        and $17, $17, $20   #r20=0x0000003F
        add $9, $9, $2  #r9=r9+1
        beq $9, $1, L2  #r1=0xFFFFFFFF
        j   L5
L2:     add $9, $0, $14 #r9=0x4
        add $9, $9, $2  #r9=0x5
L5:     lw  $5, 0($3)   #{counter0_out,counter1_out,counter2_out,led_out[12:0], SW}
        add $11, $5, $5
        add $11, $11, $11
        sw  $11, 0($3)  #{GPIOf0[13:0],LED,counter_set}, port:f0000000
        sw  $6, 4($3)   #counter port:f0000004,r6=0xF8000000
        lw  $5, 0($3)   #{counter0_out,counter1_out,counter2_out,led_out[12:0], SW}
        and $11, $5, $8 #È¡r5×îµßÎ»
        j   Disp
sysproc:mfc0 $30,$14
        addi $30,$30,0x4
        xor $27,$27,$18
        add $0,$0,$0
        add $0,$0,$0
        add $0,$0,$0
        add $0,$0,$0
        mtc0 $30,$14
        eret
```

## 5.2 测试机器码

```
1.  08000c08
2.  00000020
3.  00000020
4.  08000c6a
5.  00000020
6.  00000020
7.  00000020
8.  00000020
9.  00000827
10. 00211820
11. 00631820
12. 00631820
13. 00631820
14. 00631820
15. 00631820
16. 0060a027
17. 00631820
18. 00631820
19. 00631820
20. 00631820
21. 00631820
22. 00631820
23. 00631820
24. 00631820
25. 00631820
26. 00631820
27. 00631820
28. 00631820
29. 00631820
30. 00631820
31. 00631820
32. 00631820
33. 00631820
34. 00631820
35. 00631820
36. 00631820
37. 00633020
38. 00c61820
39. 00632020
40. 00846820
```

```
01ad4020
341b0000
0001102b
00427020
01ce7020
00005027
014a5020
ac660004
8c650000
00a52820
00a52820
ac650000
01224820
ac890000
8c0d0014
8c650000
00a52820
00a52820
ac650000
8c650000
00a85824
01a26820
11a00019
8c650000
01cef020
01ce9020
0252b020
02569020
00b25824
017b5825
0000000c
11600004
11720009
117e000b
ac890000
08000c37
11410001
08000c50
00005027
014a5020
ac8a0000
08000c37
```

```
8e290060
ac890000
08000c37
8e290020
ac890000
08000c37
8c0d0014
014a5020
01425025
022e8820
02348824
01224820
11210001
08000c62
000e4820
01224820
8c650000
00a55820
016b5820
ac6b0000
ac660004
8c650000
00a85824
08000c3f
401e7000
23de0004
0372d826
00000020
00000020
00000020
00000020
409e7000
42000018
```

## 5.3 测试结果分析

程序编译下载到 FPGA 上，数码管交替变换跑马灯和矩形变换，满足预期结果。测试程序中考虑了除 mtc0 外的所能想到的所有数据相关，并且嵌入 syscall，所以可以认为设计的 CPU 可以成功运行整个 Mips-C2 指令集。并且实现了异常。

因为时间关系，中断和 mtc0 的数据已经通过 modelsim 上的仿真但是都没有在 FPGA 上完成相关测试。

# 第 6 章 课程设计总结

## 6.1 实验经历

计算机组成与设计这个课程的实验持续了一个学期，但是实质上我们从上学期开始就开始在 modelsim 上编写调试代码，下学期正式把代码下载到 FPGA 上进行综合测试。

实验得从上学期开始，我们先是写 7 条指令的单周期，然后实现包含 42 条指令的 Mips-C2 指令集。寒假自主完成流水线的设计调试。下学期，在上学期的代码上进行修改，尝试把代码下载到 FPGA 上测试。最后，实现具备中断异常处理能力的五级流水线 CPU。

## 6.2 遇到过的问题

因为课程有一定难度，bug 总是会有的，我做实验的过程中遇到过许多问题。

| 序号 | 问题描述 | 原因 | 解决方案 |
|---|---|---|---|
| 1 | 单周期跑马灯不动 | 单周期实现 sltu 但是测试代码使用了 slt | 原始代码中把 slt 改成 sltu，重新生成 coe。 |
| 2 | 流水线矩形变换乱码 | Lw 数据相关中气泡实质未能如愿产生 | 修改 HDU 模块重新在 modelsim 上测试 |
| 3 | SW=01 矩形变换；SW=11 乱码 | WB 写回阶段下降边沿写回一次，但是 FPGA 上貌似未能这样做 | 增加判断是否数据相关电路，实现 WB 到 ID 阶段的 Forwarding。 |
| 4 | 异常产生后不能进入下一条指令 | EPC 没有加 4 | 增加测试代码实现 EPC 加 4 后写入 EPC 即可 |
| 5 | 中断测试时 PC 一直卡在 0x2-0x3 | INT 中断接受后未进行屏蔽，导致持续中断产生却不能进入中断处理程序 | 在硬件设计层面自动屏蔽已接受的中断，在退出中断处理程序时再由代码方法打开 |

## 6.3 感想

每个同学都付出了很多时间，学习的过程中我们相互学习，相互帮助，让我们收获知识的同时页收获了珍贵的友情。调试不出来的时候，会有同学陪我一起检查分析。我也会主动地去帮助其他同学一起分析错误的原因。实验的过程让我们感受到了来自同学之间的热情和温暖。

QQ 群里面一度出现经典对白。

A:明天谁去实验室

B:我晚上去，求钥匙+1

C:钥匙在***，可以来拿

D:我现在在实验室，钥匙在我手上。

我们拿到了实验室的钥匙，经常利用课后时间去做实验，也是很认真的。

这个课程已经结束，似乎折磨我们的实验也跟着远去，但是挺不甘心，付出了很多时间，却未能把所有实验都完成。

当然，我们学到了很多，对计算机组成有着更加深刻的认识和理解。我还深深认识到版本管理的重要性。在流水线的实验中，我找不到 bug，不断修改程序并进行测试，因为版本控制不好，导致浪费了很多时间。

最后感谢老师的耐心指导和同学们的帮助。

# 第 7 章 参考文献

“Computer Organization and Design　T H E H A R D W A R E / S O F T W A R E I N T E R F A C E” David A. Patterson (University of California, Berkeley ) John L. Hennessy (Stanford University)