***2018***



**计算机组成原理 课程设计报告**

|  |  |
| --- | --- |
| 题 目： | 5段流水CPU设计 |
| 专 业： | 计算机科学与技术 |
| 班 级： | CS15\*\* |
| 学 号： |  |
| 姓 名： |  |
| 电 话： |  |
| 邮 件： |  |
| 完成日期： | 2018-04-06 |

目 录

[1 课程设计概述 1](#_Toc510771142)

[1.1 课设目的 1](#_Toc510771143)

[1.2 设计任务 1](#_Toc510771144)

[1.3 设计要求 1](#_Toc510771145)

[1.4 技术指标 2](#_Toc510771146)

[2 总体方案设计 4](#_Toc510771147)

[2.1 单周期CPU设计 4](#_Toc510771148)

[2.2 单周期多级中断机制设计 10](#_Toc510771149)

[2.3 流水CPU设计 12](#_Toc510771150)

[2.4 气泡式流水线设计 13](#_Toc510771151)

[2.5 数据转发流水线设计 14](#_Toc510771152)

[2.6 动态分支预测机制 15](#_Toc510771153)

[3 详细设计与实现 17](#_Toc510771154)

[3.1 单周期CPU 实现 17](#_Toc510771155)

[3.2 中断机制实现 22](#_Toc510771156)

[3.3 流水CPU实现 25](#_Toc510771157)

[3.4 气泡式流水线实现 27](#_Toc510771158)

[3.5 数据转发流水线实现 28](#_Toc510771159)

[3.6 动态分支预测机制实现 30](#_Toc510771160)

[4 实验过程与调试 34](#_Toc510771161)

[4.1 功能测试和性能分析 34](#_Toc510771162)

[4.2 主要故障与调试 39](#_Toc510771163)

[4.3 实验进度 44](#_Toc510771164)

[5 设计总结与心得 45](#_Toc510771165)

[5.1 课设总结 45](#_Toc510771166)

[5.2 课设心得 45](#_Toc510771167)

[参考文献 47](#_Toc510771168)

# 课程设计概述

## 课设目的

计算机组成原理是计算机专业的核心基础课。该课程力图以“培养学生现代计算机系统设计能力”为目标，贯彻“强调软/硬件关联与协同、以CPU设计为核心/层次化系统设计的组织思路，有效地增强对学生的计算机系统设计与实现能力的培养”。课程设计是完成该课程并进行了多个单元实验后，综合利用所学的理论知识，并结合在单元实验中所积累的计算机部件设计和调试方法，设计出一台具有一定规模的指令系统的简单计算机系统。所设计的系统能在LOGISIM仿真平台和FPGA实验平台上正确运行，通过检查程序结果的正确性来判断所设计计算机系统正确性。

课程设计属于设计型实验，不仅锻炼学生简单计算机系统的设计能力，而且通过进行中央处理器底层电路的实现、故障分析与定位、系统调试等环节的综合锻炼，进一步提高学生分析和解决问题的能力。

## 设计任务

本课程设计的总体目标是利用FPGA以及相关外围器件，在课程实验中完成的单周期cpu基础上，完成单周期cpu在FPGA开发板上的正确运行，并改造设计五段流水CPU，要求所设计的流水CPU系统能支持自动和单步运行方式，能正确地执行存放在主存中的程序的功能，对主要的数据流和控制流通过LED、数码管等适时的进行显示，方便监控和调试。对于五段流水，要求分别使用气泡、重定向、分支预测等方式处理数据冲突和控制冲突等，此外，还要求支持中断请求。尽可能利用EDA软件或仿真软件对模型机系统中各部件进行仿真分析和功能验证。在学有余力的前提下，可进一步扩展相关功能。

## 设计要求

1. 根据课程设计指导书的要求，制定出设计方案；
2. 分析指令系统格式，指令系统功能。
3. 根据指令系统构建基本功能部件，主要数据通路。
4. 根据功能部件及数据通路连接，分析所需要的控制信号以及这些控制信号的有效形式；
5. 设计出实现指令功能的硬布线控制器；
6. 调试、数据分析、验收检查；
7. 课程设计报告和总结。

## 技术指标

1. 支持表 1.1的27条基本32位MIPS指令；
2. 支持教师指定的4条扩展指令，即表1.1中的最后四条指令，分别任blez、xori、sltiu和lbu；
3. 支持多级嵌套中断，利用中断触发扩展指令集测试程序；
4. 支持5段流水机制，可处理数据冒险，结构冒险，分支冒险；
5. 能运行由自己所设计的指令系统构成的一段测试程序，测试程序应能涵盖所有指令，程序执行功能正确。
6. 能运行教师提供的标准测试程序，并自动统计执行周期数。
7. 对于基本指令，测试程序一次性运行，对于扩展指令，要求单步手动执行，一次执行一段。
8. 能自动统计各类分支指令数目，如不同种类指令的条数、冒险冲突次数、插入气泡数目、load-use冲突次数、动态分支预测流水线能自动统计预测成功与失败次数。

表 1.1 指令集

| **#** | **指令助记符** | **简单功能描述** | **备注** |
| --- | --- | --- | --- |
| 1 | ADD | 加法 | 指令格式参考MIPS32指令集，最终功能以MARS模拟器为准。 |
| 2 | ADDI | 立即数加 |
| 3 | ADDIU | 无符号立即数加 |
| 4 | ADDU | 无符号数加 |
| 5 | AND | 与 |
| 6 | ANDI | 立即数与 |
| 7 | SLL | 逻辑左移 |
| 8 | SRA | 算数右移 |
| 9 | SRL | 逻辑右移 |
| 10 | SUb | 减 |
| 11 | OR | 或 |
| 12 | ORI | 立即数或 |
| 13 | NOR | 或非 |
| 14 | LW | 加载字 |
| 15 | SW | 存字 |
| 16 | BEQ | 相等跳转 |
| 17 | BNE | 不相等跳转 |
| 18 | SLT | 小于置数 |
| 19 | STI | 小于立即数置数 |
| 20 | SLTU | 小于无符号数置数 |
| 21 | J | 无条件转移 |
| 22 | JAL | 转移并链接 |
| 23 | JR | 转移到指定寄存器 | If $v0==10 halt(停机指令)  else 数码管显示$a0值 |
| 24 | SYSCALL | 系统调用 |
| 25 | MFC0 | 访问CP0 |  |
| 26 | MTC0 | 访问CP0 |  |
| 27 | ERET | 中断返回 |  |
| 28 | BLEZ | 小于或等于0跳转 |  |
| 29 | XORI | 异或立即数 |  |
| 30 | SLTIU | 小于立即数置1（无符号） |  |
| 31 | LBU | 加载字节（地址无符号） |  |

# 总体方案设计

## 单周期CPU设计

本次我们采用的方案是硬布线控制。将系统分成指令存储器IM、操作控制器、寄存器文件、运算器ALU、数据存储器DM、地址转移逻辑NPC等几大模块。从指令存储器中取出一条指令，操作控制器根据指令解析出各模块间的数据通路控制信号，寄存器文件的输出作为ALU的操作数，ALU的运算结构送到数据存储器或者寄存器文件，地址转移逻辑决定下一条指令的地址。以上所有操作都在一个周期内完成，不需要考虑各种数据冲突。实际上，取指令和取数据存储器的值都涉及到访存，可能会有冲突，为此需要另加控制信号，为了简化系统的实现，我们把指令存储器和数据存储器分开，用增加硬件的方式解决了这一冲突。

在logisim平台上模拟的单周期cpu已经在课程实验中完成了，现在需要用Verilog代码来描述该系统，使之能够在FPGA开发板上完成相同的功能，这个过程不需要修改系统的设计，只需要根据电路图划分模块，将各模块的输入输出作为Verilog模块的接口，即可完成转换。理论上，用Verilog生成的结构图与Logisim的结构图基本相同。总体结构图如图 2.1所示。



图 2.1 单周期总体结构图

### 主要功能部件

#### 程序计数器PC

程序计数器决定取指令的地址，接收地址转移逻辑的输出作为输入，在每个时钟上升沿到来时输出一个地址送到指令存储器，应该用一个寄存器来实现。该寄存器需要时钟信号、清零信号，使能端接停机信号，当停机信号到来时PC值不再改变，系统也就停止运转。输入输出引脚描述如表 2.1。

表 2.1 程序计数器引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| Clk | 输入 | 1 | 时钟 |
| Rst | 输入 | 1 | 清零 |
| En | 输入 | 1 | 使能端，为0时忽略时钟输入 |
| PC\_in | 输入 | 32 | 加载到寄存器输入端的值 |
| PC\_out | 输出 | 32 | 寄存器输出端的值 |

#### 指令存储器IM

为简化实现，将IM和DM分开实现。指令是只读的，应该用ROM存储器来存储；MIPS指令都为定长32位，所以数据位宽设为32位；程序计数器PC输出的是字节地址，一条指令有4个字节，所以指令存储器应该以字寻址，将PC截去低2位作为ROM的输入。考虑到我们实现的只是一个模拟的cpu，可以不用全部的30位字地址，取10位即可，也就是PC的第2到第11位。输入输出引脚描述如表 2.2。

表 2.2 指令存储器引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| PC | 输入 | 10 | 程序地址 |
| Instru | 输出 | 32 | 待解析指令 |

#### 数据存储器DM

数据存储器要求可读可写，用RAM实现。sw、lw指令要读写字，lbu指令则要求读取字节，所以要能够以不同的模式进行访问。通过两位的访问模式信号决定是字、字节或半字访问，写使能信号决定读数据或写数据，时钟上升沿控制写入，而读数据则是组合逻辑，随时可读。读写的地址有32位，与指令存储器同理，我们不需要使用那么大的空间，所以可以截取低12位作为读写地址。输入输出引脚描述如表 2.3。

表 2.3 数据存储器引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| Clk | 输入 | 1 | 时钟 |
| Rst | 输入 | 1 | 清零 |
| Mode | 输入 | 2 | 访问模式，00表示字访问，01字节访问 |
| We | 输入 | 1 | 写使能信号 |
| Din | 输入 | 32 | 待写入的数据 |
| addr | 输入 | 12 | 读写的字节地址 |
| Dout | 输出 | 32 | 读出的数据 |

#### 运算器

运算器接受两个32位的操作数，根据操作码决定对两个数进行何种运算，输出一个32位的结果，并输出一位数表示两个操作数是否相等。除此之外还有有符号溢出、无符号溢出等多种输出，由于在这里不需要使用，所以忽略这些输出。输入输出引脚描述如表 2.4。

表 2.4 ALU引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| aluX | 输入 | 32 | 操作数X |
| aluY | 输入 | 32 | 操作数Y |
| Aluop | 输入 | 4 | 操作码，决定运算方式 |
| Result | 输出 | 4 | 计算结果 |
| Equal | 输出 | 32 | 两操作数是否相等的标志 |

各操作码对应的运算方式如表 2.5。

表 2.5 操作码功能描述

| Aluop | 运算 |
| --- | --- |
| 0000 | 逻辑左移 |
| 0001 | 算术右移 |
| 0010 | 逻辑右移 |
| 0011 | 相乘 |
| 0100 | 相除 |
| 0101 | 相加 |
| 0110 | 相减 |
| 0111 | 按位与 |
| 1000 | 按位或 |
| 1001 | 按位异或 |
| 1010 | 按位或非 |
| 1011 | 符号比较 |
| 1100 | 无符号比较 |

#### 寄存器堆RF

寄存器文件包含MIPS中的32个32位寄存器，一次可以读取ra和rb两个寄存器；一次可写一个寄存器，用rw表示，用时钟控制，一位写使能信号用来标识当前周期是否需要写数据；写入和读出的数据都是32位的。输入输出引脚描述如表 2.6。

表 2.6 寄存器堆引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| Clk | 输入 | 1 | 时钟 |
| Rw | 输入 | 5 | 写寄存器号 |
| Ra | 输入 | 5 | 读寄存器号1 |
| Rb | 输入 | 5 | 读寄存器号2 |
| We | 输入 | 1 | 写使能 |
| W | 输入 | 32 | 写入数据 |
| A | 输出 | 32 | 读出数据1 |
| B | 输出 | 32 | 读出数据2 |

### 数据通路的设计

对于电路中的各大部件，记录其输入端数据来源，忽略控制类信号，仅保留数据类信号，得到数据通路框架如表 2.7。

表 2.7 指令系统数据通路框架



### 控制器的设计

首先对于控制信号进行统计，包括各个主要部件所需要输入的控制信号，以及数据通路合并表中所示的具有多输入的主要部件需要进行输入选择的控制信号，并且对各个统计信号的各种取值情况进行定义。统计得到的控制信号以及说明如表 2.8。

表 2.8 主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| Jal | 1 | 函数调用，写寄存器的编号选择31号 |
| 0 | 写寄存器编号选择指令解析器解析出的rd、rt |
| RegDst | 1 | 写寄存器编号为rt |
| 0 | 写寄存器编号为rd |
| Syscall | 1 | 读寄存器编号1为2，读寄存器编号2为4 |
| 0 | 读寄存器编号1为rs，读寄存器编号2为rt |
| Jump | 1 | 无条件跳转，npc取26位立即数 |
| 0 | 不跳转，npc取其他值 |
| Branch | 1 | 有条件跳转成功，npc选择16位立即数 |
| 0 | Npc为其他值 |
| Return | 1 | Jr指令，函数调用返回，npc为31号寄存器的值 |
| 0 | Npc为其他值 |
| Regdin | 1 | 写入寄存器的值为数据存储器的输出 |
| 0 | 写入寄存器的值为alu 的输出 |
| Shift | 1 | 移位标志，立即数扩展器输出5位移位值的32位扩展结果 |
| 0 | 立即数扩展器输出16位立即数的扩展结果 |
| Unsigned | 1 | 无符号扩展标记，16位立即数做无符号扩展 |
| 0 | 16位立即数做有符号扩展 |
| regW | 1 | 写寄存器使能信号，允许将寄存器的输入加载到输出 |
| 0 | 不允许写寄存器，寄存器输出值不变 |
| aluB | 1 | Alu操作数Y的选择，选择立即数扩展的结果 |
| 0 | 选择寄存器文件的第二个输出 |
| DMWrite | 1 | 数据存储器写使能信号，允许向DM写数据 |
| 0 | 从DM读数据 |
| Mode | 00 | 对DM进行字访问 |
| 01 | 对DM进行字节访问 |
| 10 | 对DM进行半字访问 |
| Aluop | 0000-110 | 功能过多，详见表 2.5 |

对照所有控制信号，依次分析各条指令，分析该指令执行过程中需要哪些控制信号。该控制信号表的框架如表 2.9所示，其中为空的表项表示取值为0或者取任意值均可。

表 2.9 主控制器控制信号框架

## 单周期多级中断机制设计

### 总体设计

中断响应的执行流程如图 2.2所示。

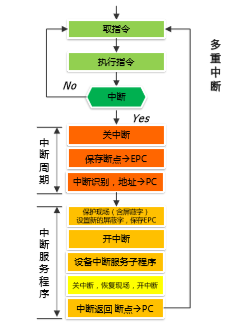


图 2.2 多级中断响应流程

用独立请求的方式响应中断，每个中断对应一个中断号。根据中断号确定执行哪段中断程序，用类似于中断向量表的方式存放中断程序入口，可以事先获取中断入口地址，以常量形式进行选择。

为了在中断执行后能够正常返回主程序，要将跳转前的地址保存下来，就像jal指令将返回地址保存在31号寄存器中一样。每级中断在执行过程中都需要动态地改变中断优先级，所以需要若干个中断屏蔽寄存器。在执行某些操作时，不允许任何外部事件中断，需要完全关中断，所以还需要添加中断使能寄存器IE。

对EPC、IE和中断屏蔽寄存器的操作需要增加指令才能实现，具体为MFC0和MTC0，以及中断返回指令ERET，所以还需要修改操作控制器和数据通路。

### 硬件设计

1. 中断屏蔽寄存器

用D触发器实现，数量与中断的数量相同，为3个。时钟上升沿触发，正常执行时写入0，不屏蔽任何中断；中断程序执行前向屏蔽寄存器写入1。寄存器输出取反，和中断请求信号进行与操作。

1. 中断识别

三个中断按钮都可能被按下，但是一次只能响应一个，应该有限响应优先级高的，应该用优先编码器对中断请求信号进行编码。

1. 中断使能寄存器

用D触发器实现，输出1时允许中断，输出0时屏蔽所有中断。

1. EPC寄存器

用32位寄存器实现，上升沿触发，当有中断请求时，使能端置1，写入当前PC+4的值作为返回地址。中断执行过程中，使能端一直为0，不许写入新的值。

1. 操作控制器修改

为了支持新的三条指令，增加三个比较器，根据操作码判断是否为eret、mfc0或者mtc0指令。Mfc0和mtc0需要读写通用寄存器和IE、EPC寄存器，所以需要将mfc和mtc指令作为寄存器的使能信号。

1. 硬件关中断

在中断请求发出的一个周期，中断信号为1，系统响应中断，但是一个周期之后应立马关中断，防止后续中断继续响应，这应该由硬件实现，可以将中断信号反馈到D触发器的清零端。

### 软件设计

依靠硬件只能实现单级中断，不能在中断执行过程中将其打断，因为新的EPC值会覆盖原有的值，为此要用软件方法实现。

在中断程序的最开始，需要将通用寄存器都压栈保存。用数据存储器DM作为堆栈，用sw指令完成即可。然后依次用mfc0指令读取IE、EPC和中断屏蔽寄存器的值，存入通用寄存器。再将新的中断屏蔽字写回屏蔽寄存器，并将IE写入1，实现软件开中断，在此之前实现的操作都是不可打断的，在此之后中断可以被嵌套打断。

执行完中断处理程序后，按照原来入栈顺序的倒序依次出栈各寄存器的值，恢复IE、EPC和中断屏蔽字。

## 流水CPU设计

### 总体设计

在指令的实际执行过程中，取指、译码、执行等操作都是有先后顺序的，有的操作必须使用另一操作的结果，所以很多部件需要等待其他部件运算完毕才能接着计算，系统效率极低。为了充分利用各部件，需要设计五段流水线。一条指令的操作可以细分为五个阶段，如图 2.3所示。

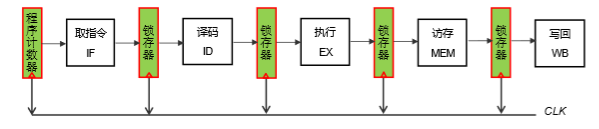


图 2.3 指令流水线逻辑架构

### 流水接口部件设计

流水接口部件的作用是将上一段中的数据和控制信号锁存一个周期，供下一段使用，应该用寄存器实现。时钟上升沿触发；使能端接停机信号；将清零信号作为多路选择器的选择信号，正常情况下选择输入信号，清零时选择常数0。各段的流水接口结构大体相同，只是传输的数据不同。

### 理想流水线设计

将单周期cpu中的各部件依次划分到五个段中。

1. 取指：程序计数器PC，指令存储器IM。
2. 译码：指令解析器，有符号、无符号扩展器，操作控制器，寄存器堆。
3. 执行：运算器ALU。
4. 访存：数据存储器DM。
5. 写回：与译码段一样使用寄存器堆，不过这里是写寄存器，为了防止数据冲突，需用下降沿写入，保证下一个上升沿到来时，译码阶段能取到最新的寄存器值。

## 气泡式流水线设计

### 总体设计

理想流水线只能顺序执行，无法解决跳转问题；此外，相邻指令间还存在着读写数据的冲突，读寄存器指令读取的寄存器值还未被及时写回，会导致读取错误。

分支跳转是否成功要等到执行段才能判断出来，此时修改PC的值为分支目标地址，取指段和译码段已经取出了两条指令，这两条指令是不应该被执行的，应该清空，使CPU空转两个周期，这就是控制冲突的解决方案。

数据冲突有多种，其中对电路功能有影响的只有先写后读。第一条或第二条指令要写寄存器，而且寄存器的编号与第三条指令要读的寄存器编号相同时，就会有数据冲突。因为第三条指令在译码段读寄存器，此时前两条指令还未执行到写回段，第一条指令必须等待两个周期才能取到正确的值，可以在两者之间插入空转周期实现等待。

### 控制冲突处理设计

无条件跳转在译码段就能得到目标地址；有条件跳转要等到执行段利用alu的运算结果才能确定。为统一管理，将j型指令的控制信号也向后传送一个周期，与b指令同等处理。在执行段判断当前指令是否为j指令，或者为b指令且成功跳转，满足条件时将目标地址送到pc寄存器，完成跳转，同时还要将IF-ID和ID-EX流水部件完成误取指令的清除。

### 数据冲突处理设计

要判断一条指令与其前面两条指令的关系，应该在译码、执行、访存段判断，将执行段和访存段的写寄存器编号和写使能送回译码段，与译码段要读的寄存器编号相比较，假如两者相同则冲突，在ID-EX流水部件中插入一个气泡。读写0号寄存器不算冲突，所以要排除这种情况。生成的插气泡信号接到ID-EX的清零端清零实现插入气泡，此外还要接到IF-ID和PC寄存器的使能端，在插入气泡的同时将前两段锁住。

## 数据转发流水线设计

### 总体设计

数据冲突中要读的寄存器值没有及时写回寄存器，它已经存在于访存段和写回段的流水部件中，重定向的思路就是把这些数据送到需要寄存器最新值的地方。重定向检测放在执行段，判断执行段读寄存器与访存段、写回段写寄存器之间是否冲突。

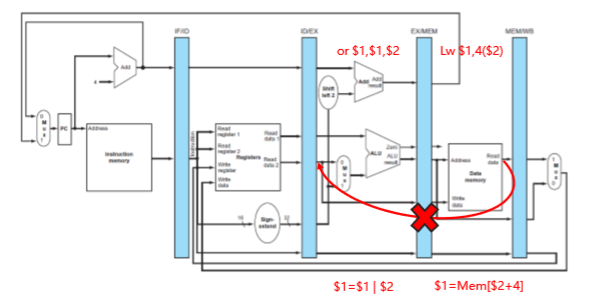


图 2.4 Load-Use相关

若相邻两条指令数据相关，且前一条是访存指令，不能采用重定向。如图 2.4，EX段读寄存器与MEM段目的寄存器相同，此时寄存器的值没有锁存在EX/MEM中，等DM读操作完成后才会出现在Readdata 上，若将其直接重定向到EX段，会使得EX段的延迟变成了访存延迟+运算器延迟。所以应在ID段就检出，并插入一个气泡。

### Load-Use相关处理设计

Load-Use相关要在ID段检出，在ID段增加检测逻辑，当ID段为读寄存器指令、EX段为lw或lbu指令、ID段读寄存器编号与EX段目的寄存器编号相同条件均满足时，即为load-use相关，需要生成气泡插入信号。

### 重定向设计

重定向是从访存段和写回段定向到alu的两个输入端，两段的数据均可能定向到两个输入端，所以共有四种定向可能，需要四个控制信号。

四个控制信号的生成逻辑基本相同，只是控制的数据来源和去向不同。重定向要满足三个条件：执行段读寄存器、执行段读寄存器编号与访存段/写回段写的寄存器编号相同、访存段/写回段的寄存器写使能信号为1，表示该段需要写寄存器。上述条件中包含了Load-Use相关，不过Load-Use相关已经在ID段检测出并且插入了气泡，所以在执行段不会再检测出。理论上还要求读写寄存器的编号不为0，但是没有哪一条指令会写零号寄存器，所以这里也省去了这一逻辑。

## 动态分支预测机制

### 总体设计

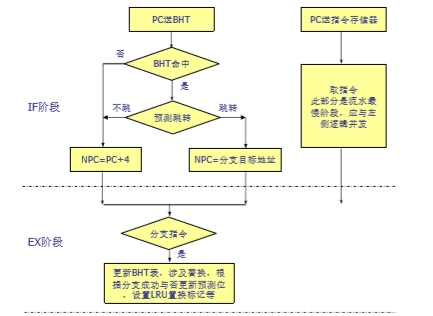


图 2.5 动态分支预测流程

重定向流水线相比气泡流水线已经有了极大的优化，但是仍然因为跳转清空时误取指令而浪费了很多时钟周期，为了继续优化，我们要能够尽早判断出跳转，使误取深度将为0，这就是动态分支预测需要完成的工作。

分支预测最核心的部件是分支历史表BHT，原理类似于广泛使用的cache，将最近出现过的数据进行缓存，下次使用时如果命中，就能够直接使用。动态分支预测流程如图 2.5所示。

### BHT设计

BHT的所有表项均用寄存器实现，设置8个表项，增大命中概率。BHT表主要包含有效位、指令地址、分支目标地址、预测历史和LRU。

有效位标记该表项是否被使用，初值为0，一旦存入数据则永远为1；指令地址是跳转指令在指令存储器中的地址；分支目标地址是指令成功跳转时的目标地址；预测历史用双预测位，高位为1时预测跳转，否则预测不跳转，状态转移图如图 2.6，转移条件为实际跳转情况，而不是预测跳转情况。

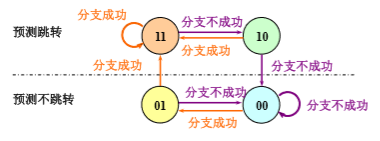


图 2.6 采用双预测位的BHT状态转换

在8个表项还未被填满时，用优先编码器获取当前为空的表项的编号，将该表项使能端置1，写入分支地址值，将该项的有效位置1，表示该表项已经被使用。若8个表项都被填满了，而当前又有以前没有出现过的跳转情况出现，则需要涉及到表项的替换。替换算法采用LRU算法，要从8个表项中选取lru值最大的一项，用比较器两两比较各表项的值，两者中较大者继续比较，最终选出最大者，将该表项的使能端置1，写入新的分支指令地址和目标地址，并将lru值置0，表示该项刚使用过。

# 详细设计与实现

## 单周期CPU 实现

### 主要功能部件实现

1. 程序计数器（PC）
2. Logism实现：

用一个32位寄存器PC，上升沿触发，输入来自于地址转移逻辑的输出，输出送到指令存储器。Halt为停机信号，取反后接在寄存器的使能端。如图 3.1所示。

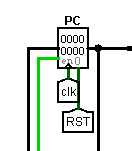


图 3.1 程序计数器（PC）

1. FPGA实现：

程序计数器PC的Verilog代码如下：

always@(posedge clk) begin

if(0==rst) pc <= 0;

else if(halt) pc <= pcout;

else pc <= pc;

end

1. 指令存储器（IM）
2. Logism实现：

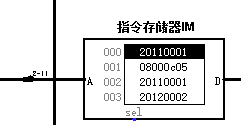


图 3.2 指令存储器（IM）

用一个只读存储器ROM实现。设置地址位宽为10位，数据位宽为32位。因为PC中存储的指令地址有32位，而ROM地址线宽度有限，仅为10位，故使用分线器只取32位指令地址的2-11位作为指令存储器的输入地址。如图 3.2所示。

1. 数据存储器（DM）
2. Logism实现：

使用4个数据位宽为8的随机存储器RAM实现，从32位地址中截取低12位作为DM的字节地址，字节地址中的高10位为字地址，低两位为字节片选信号。具体选择电路很复杂，这里只展示RAM的部分结构，如图 3.3所示。

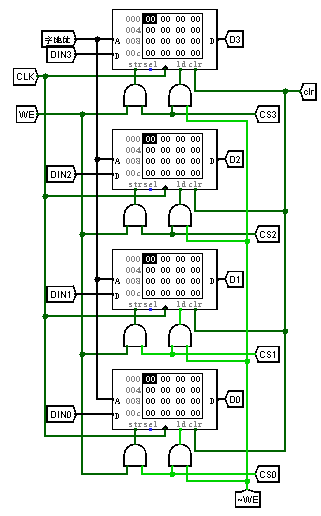


图 3.3 数据存储器（DM）

1. FPGA实现：

写数据用用always行为描述；读数据用assign赋值。DM的Verilog代码如下：

always @(posedge clk) begin

if(DMwrite) memory[addr1] <= Din;

else memory[addr1] <= memory[addr1];

end

assign Dout = (mode==0) ? data :

(mode==1) ? {24'h000000, bytes} :

(mode==2) ? {16'h0000, half} : 0;

1. 寄存器文件
2. Logism实现：

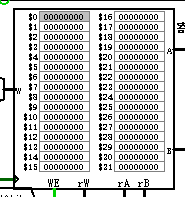


图 3.4 寄存器文件

使用CS3410库中提供的寄存器堆实现寄存器文件。如图 3.4所示。

1. FPGA实现：

用reg类型实现寄存器，用always行为描述，时钟上升沿时加载输入，0号寄存器永远位0。输出为组合逻辑，随时读出，不需要时钟。Verilog代码如下：

reg [31:0]memory[31:0];;

always @(posedge clk) begin

memory[0]=0;

if(WE==1) memory[rw]=w;

end

assign A=memory[rA];

assign B=memory[rB];

1. RTL电路图：Vivado生成的RTL电路图如图 3.5所示。

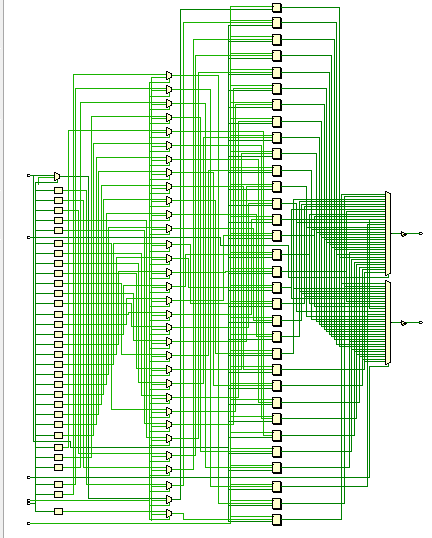


图 3.5 寄存器文件RTL图

### 数据通路的实现

采用工程化设计模式，一次构建所有数据通路。将每条指令改成RTL，仅保留数据类信号，记录各部件输入端数据来源，数据通路表已在2.1.2中给出，不再重复。

根据数据通路表进行多指令数据通路的合并，将各个主要功能部件进行连接，根据数据通路合并表的最终结果，对于所有的多输入部件使用多路选择器进行输入选择。最终便可以完成数据通路的搭建。数据通路如图 3.6所示。

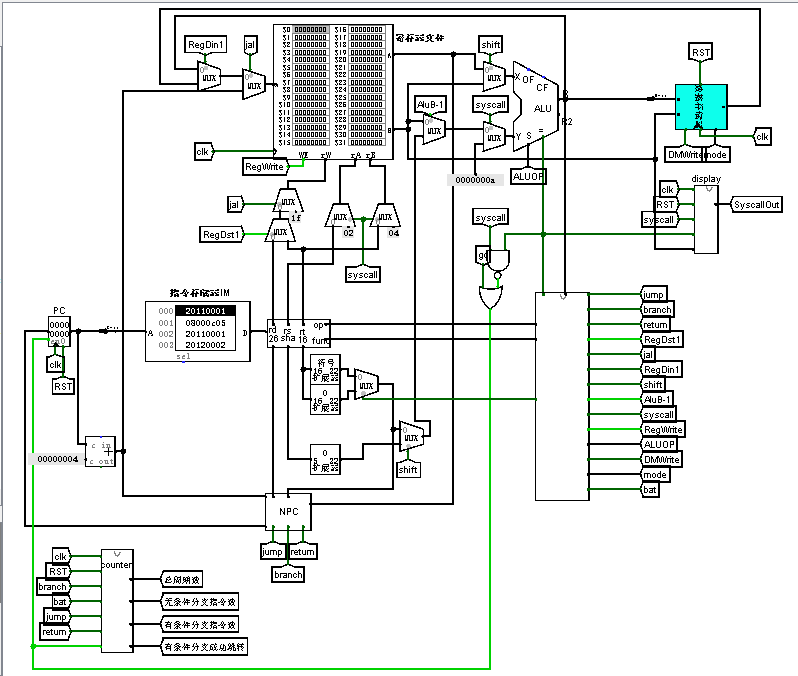


图 3.6 单周期CPU数据通路（Logism）

在Vivado中使用Verilog语言搭建的数据通路的原理图如图 3.7所示。

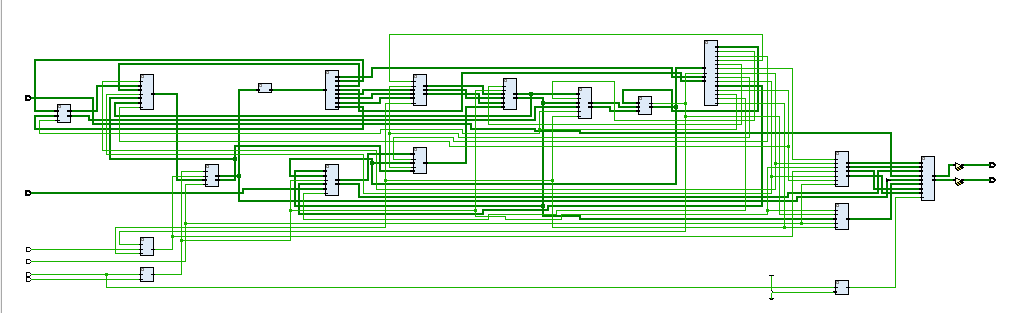


图 3.7 单周期CPU数据通路（FPGA）

### 控制器的实现

1. Logisim实现

根据指令op和func字段判断指令类型，用比较器实现，部分电路如图 3.8所示。

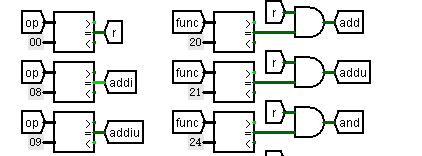


图 3.8 解析指令类型

得到指令类型信号以后，结合表 2.9中给出的控制信号，根据每种控制信号对应的指令类型，用或门将这些指令生成各控制信号，部分电路如图 3.9所示。

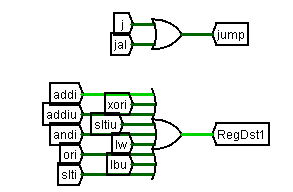


图 3.9 控制信号生成

1. FPGA实现

Logisim 中的比较逻辑在Verilog中可以用“==”操作符实现，逻辑门可以用“|”、“&”等操作实现，整个操作控制器都是组合逻辑，所以全部使用assign赋值，虽然代码量大，但是逻辑简单，部分代码如下：

assign ret = (func==6'h08)& r;

assign syscall = (func==6'h0c)& r;

assign tmpsys = r & (~(ret | syscall));

assign tmpbr = equal | (alur==32'h00000001);

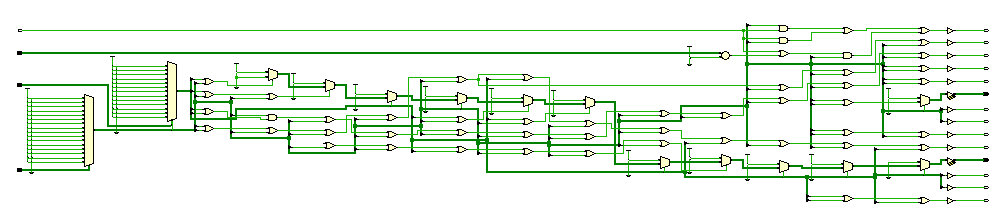


图 3.10 主控制器原理图

用RTL分析得到主控制器原理图如图 3.10所示。

## 中断机制实现

### 硬件实现

1. 中断等待信号产生

如图 3.11所示，用两个D触发器实现，中断按键信号接到第一个触发器的时钟端；D触发器1的输出和复位信号取反相与，作为第二个触发器的输入。

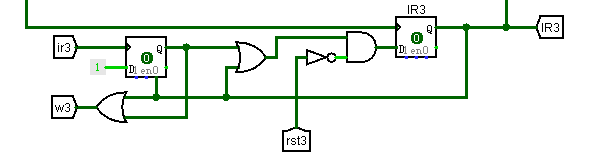


图 3.11 中断等待信号产生电路

1. 中断屏蔽寄存器

用四个D触发器来实现，级别为0、1、2、3，依次增加。触发器输出的补值与中断请求相与得到中断信号。当mtc0指令执行时，选择信号选择待写入的值进行写入，改变中断优先级。电路实现如图 3.12所示。

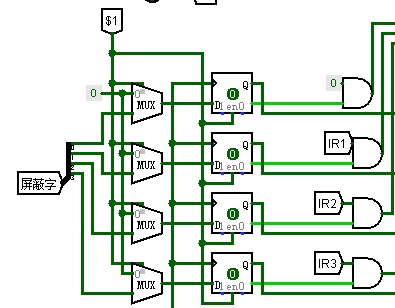


图 3.12 中断屏蔽寄存器电路

1. 中断入口地址选择

如图 3.13，将中断请求信号接入优先编码器，输出编码代表当前优先级最高的中断，将各中断的入口地址用常数形式接到多路选择器的输入，用中断编号作为选择信号，选出的地址即为当前需要响应的中断的入口地址。

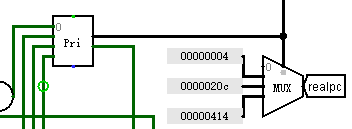


图 3.13 中断入口地址选择

1. 等待信号清零

与入口地址选择一样，也用优先编码器实现，将当前中断请求信号接到优先编码器上，输出编码再用解码器解码，就生成了同步清零信号。这里接入的是原始中断请求信号，编码结果表示当前执行的中断，而图3.14中接入的是经过屏蔽字处理后的信号。电路如图 3.14所示。

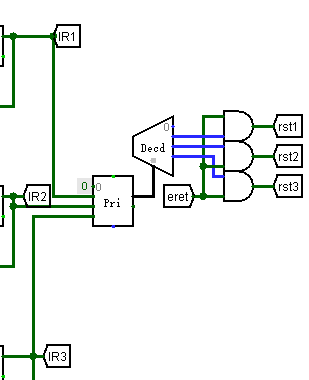


图 3.14 等待信号清零信号产生电路

1. IE和CP0寄存器实现

IE寄存器用D触发器实现；CP0寄存器用32位宽的寄存器实现。

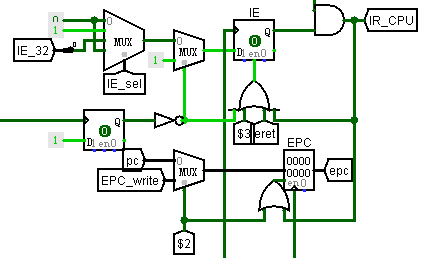


图 3.15 中断寄存器实现

电路启动时，IE写入1；硬件关中断时，IE写入0；mtc0执行时，IE写入通用寄存器的值，所以IE输入端需要多路选择器进行选择，选择信号由控制电路生成。IE使能端接一个或门的输出，当有中断请求、返回指令以及其他两个控制信号时允许写入，其他情况锁死。

CP0寄存器正常时写入当前PC+4值，mtc0执行时写入通用寄存器值，用二路选择器选择后送入输入端，控制信号由控制电路产生。使能端接控制信号与中断使能信号相或的结果。电路如图 3.15所示。

1. 中断控制信号生成

原单周期cpu不支持中断指令，需增加控制逻辑。三条中断指令的op字段均为0x10，这还不足以区分三条指令，但这三条指令的rs寄存器字段各不相同，所以可以作为判断依据。如图 3.16所示，通过比较器对op字段和rs字段综合判断，得出三条指令的控制信号。

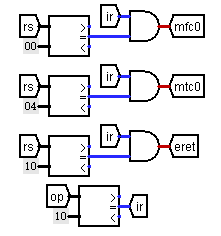


图 3.16 中断指令识别

### 软件实现

中断程序执行前，先将32个通用寄存器中需要用到的寄存器压栈，堆栈在数据存储器中选取一段实现，用sw指令实现压栈，每次对sp寄存器减4。紧接着读取屏蔽字、CP0寄存器的值，为了方便，用MFC0，$1,$1表示将屏蔽字送入1号通用寄存器，MFC0 $2,$2表示将CP0送入2号通用寄存器。然后用MTC0向寄存器写入1，实现软件开中断。执行中断程序体，执行完成后出栈恢复各寄存器的值，顺序与压栈顺序相反。压栈部分代码如下，出栈部分与之类似：

addi $sp,$sp,-4 #通用寄存器压栈

sw $25,0($sp)

MFC0 $1,$1 #读屏蔽寄存器并压栈

addi $sp,$sp,-4

sw $1,0($sp)

MFC0 $2,$2 #读CP0寄存器并压栈

addi $sp,$sp,-4

sw $2,0($sp)

addi $1,$0,0x3 #写屏蔽寄存器，该变中断优先级

MTC0 $1,$1

addi $3,$0,0x1 #软件开中断

MTC0 $3,$3

## 流水CPU实现

### 流水接口部件实现

以EX-MEM接口为例，如图 3.17所示，所有输入信号都用对应位宽的寄存器进行输出，时钟上升沿触发，使能端接停机信号halt，清零信号接多路选择器的选择端，正常情况下选择输入信号，清零信号到来时选择0，实现同步清零。

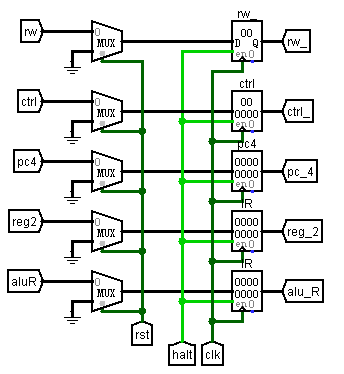


图 3.17 流水接口部件实现

### 理想流水线实现

理想流水线数据通路如图 3.18所示。

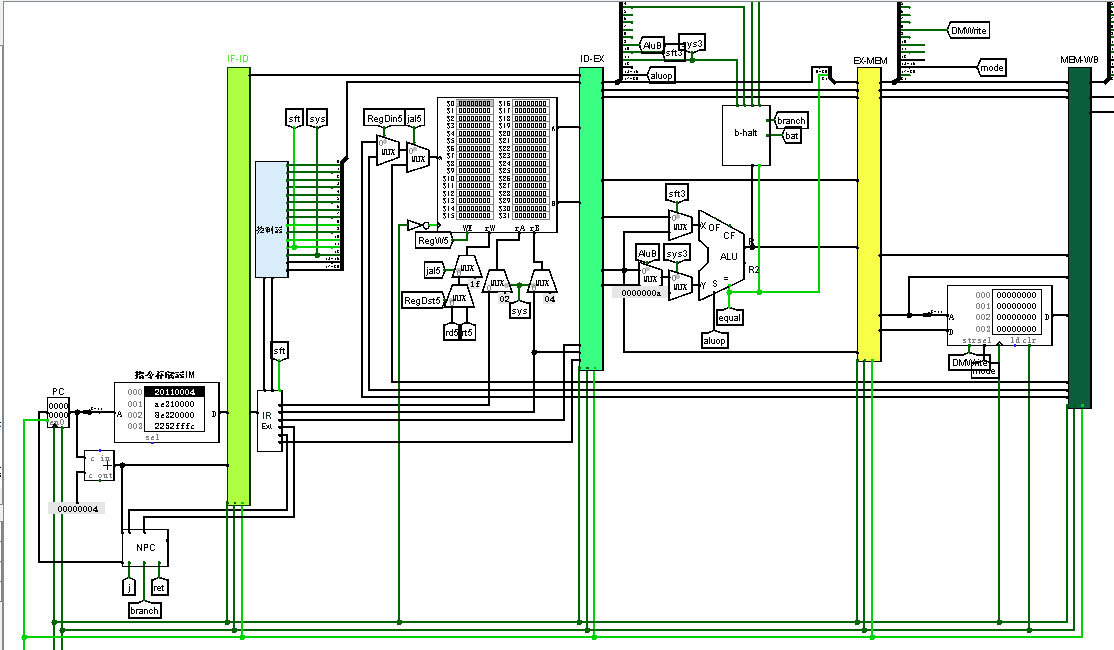


图 3.18 理想流水线实现

取指段，PC的输出送IM的输入，取出的指令通过IF-ID接口部件传输到译码段，PC+4的值也向后传输。译码段，指令解析器解析出指令各字段并送到操作控制器和立即数扩展器，操作控制器输出的控制信号有部分就作用在本段，其他不需要用到的信号用集线器整合，送往下一段，寄存器输出值、立即数扩展器的输出也向后传输。

执行段，用分线器分出本段需要用到的控制信号，剩下的用集线器打包后继续向后传送。alu的运算结果、寄存器输出和PC+4向后传送。访存段，寄存器的第二个输出作为DM的输入，alu运算结果作为访存地址，用分线器从控制信号中分出访问模式和写使能信号，剩下的信号继续向后传送。访存输出继续向后传送。

写回段，alu输出、寄存器输出、PC+4的值均送回到寄存器的输入端，选择信号从第五段的控制信号中用分线器分出。此外，停机信号也在此段生成并起作用，如果在其他段起作用，那么系统会提前几个周期停机。

## 气泡式流水线实现

### 控制冲突处理实现

如图 3.19所示，“=”表示alu的相等输出，alur为alu的运算结果。Beq信号和相等信号相与表示beq成功跳转；同理，bne信号和相等信号的非相与，表示bne跳转成功；blez相对复杂，在寄存器值小于或者等于0时跳转，小于0的判断用的是alu的小于0置1功能，所以要判断alu的结果是否等于1，用比较器比较，在和相等信号相或，表示二者满足其一即可跳转。将生成的信号接到ID-EX和IF-ID流水接口部件的清零端即可。

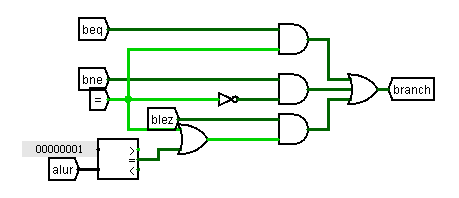


图 3.19 有条件成功跳转信号生成电路

### 数据冲突处理实现

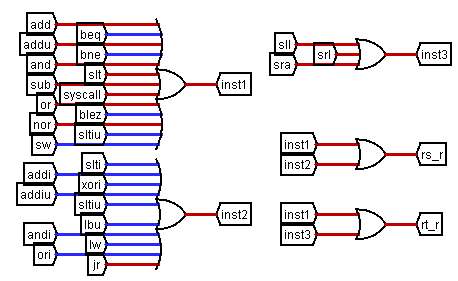


图 3.20 读寄存器使能信号

如图 3.20，在操作控制器中增加读寄存器控制信号生成电路，inst1表示同时读rs和rt寄存器的指令，inst2表示只读rs的指令，inst3表示只读rt寄存器的指令，将三者两两相或，得到读rs和rt寄存器的控制信号。

气泡生成需满足：译码段访问rs/rt寄存器中的至少一个；译码段读寄存器的编号与执行段写寄存器编号之一相同，用比较器比较；执行段写寄存器编号不为0，也用比较器；执行段写寄存器使能信号为1，表示该指令需要写寄存器。将这四个条件用与门与起来得到气泡逻辑。上述是译码段与执行段的数据冲突，译码段与访存段的处理方式完全相同，只是写寄存器编号要从访存段送回。具体电路如图 3.21所示。

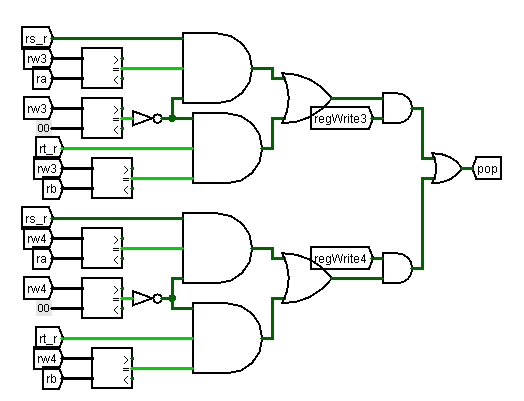


图 3.21 气泡生成电路

## 数据转发流水线实现

### Load-Use相关处理实现

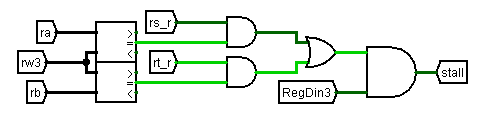


图 3.22 Load-Use相关处理电路

如图 3.22所示为load-use相关时插入气泡的逻辑电路，用比较器判断ID段的读寄存器编号ra、rb是否与执行段的写寄存器编号rw3相同；rs\_r、rt\_r是ID段的读寄存器标志，由控制器生成，在气泡流水线中已经介绍过；RegDin3是执行段的load指令标志，是控制器中原本存在的一个控制信号，仅当lw和lbu指令时为1，所以刚好可以用来作为load标志。当ID段读至少一个寄存器，且该寄存器编号与EX段写寄存器相同，且EX段为load指令，则生成气泡，在ID-EX流水接口部件插入。

### 重定向实现

1. Logisim实现：

如图 3.23所示为重定向的四个控制信号的生成电路。四个信号生成逻辑基本相同，仅说明sel1的生成：rs\_r、regWrite4和比较器的比较结果用逻辑门相与，结果为选择信号，其中rs\_r为EX段读rs寄存器的标志，regWrite4是访存段写寄存器使能，比较器比较访存段写寄存器编号与执行段读寄存器编号ra是否相同。

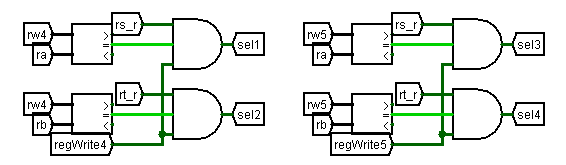


图 3.23 重定向控制信号生成电路

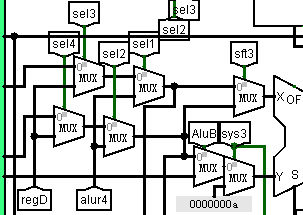


图 3.24 重定向电路

图 3.23中生成的重定向信号最终作用在alu的输入端，如图 3.24所示，选择电路极为复杂，其中右半部分为单周期中已经存在的控制逻辑，左半部分的四个多路选择器是新增的。alur4是访存段的alu结果，是ex段与mem段数据冲突时需要重定向的数据；regD是写回段写寄存器的值，是ex段与wb段数据冲突时需要重定向的数据。

1. FPGA实现：

将logisim中实现的重定向流水线转换成Verilog代码，转换过程比起单周期cpu复杂很多，因为流水线的连线更复杂，所以需要实现对每一根线进行命名，所有的模块严格按照命名规范来定义输入输出接口。代码量太大，不予展示，最终生成的RTL电路图如图 3.25所示。

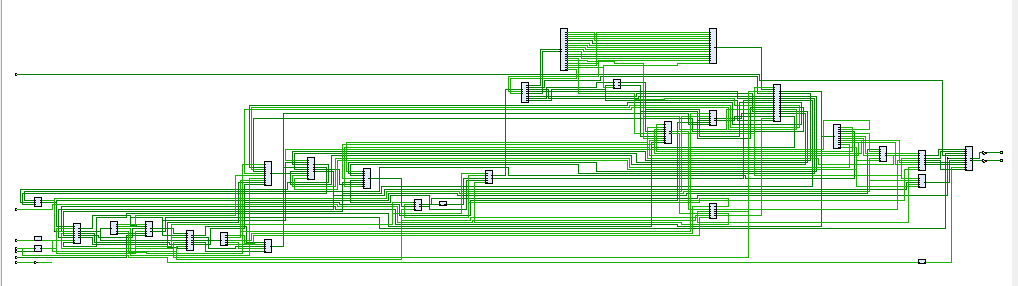


图 3.25 重定向流水线RTL图

## 动态分支预测机制实现

### BHT实现

列出预测历史位状态转移图的真值表，用logisim自动生成电路，结果如图 3.26所示。

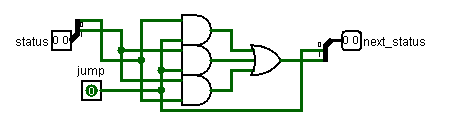


图 3.26 预测历史状态转移逻辑

如图 3.27所示，8个表项一共8行，每行5个寄存器，分别为有效位、指令地址、目标地址、预测历史位和lru值，时钟上升沿触发。

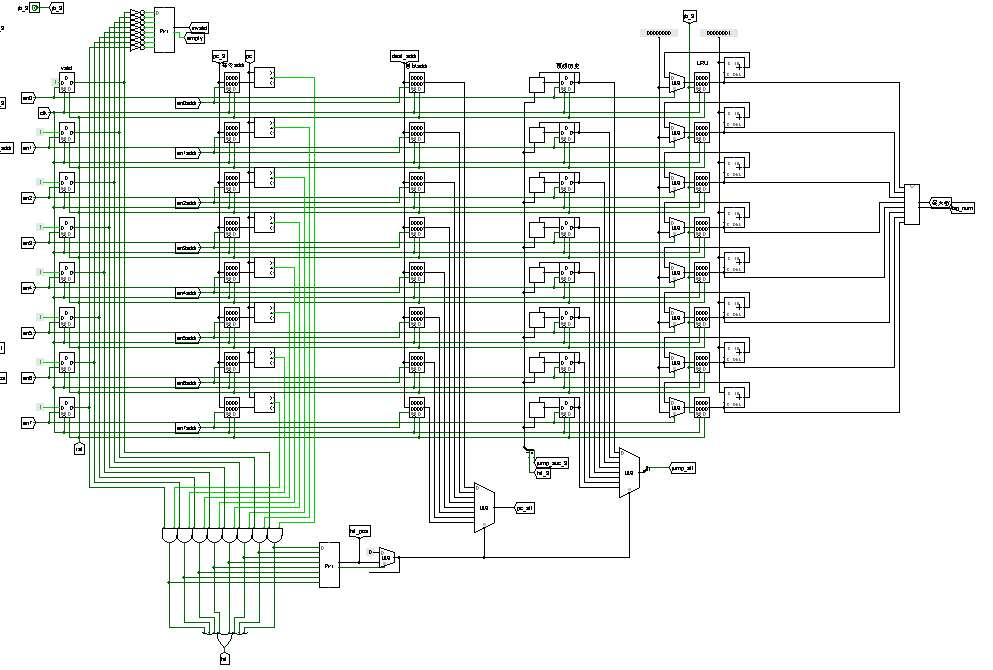


图 3.27 分支历史表

如图 3.28是淘汰表项所需的最大数选择电路，将8个lru值分成四组进程比较，选出4个最大值，继续比较，用三次二分法选出最大数的编号。

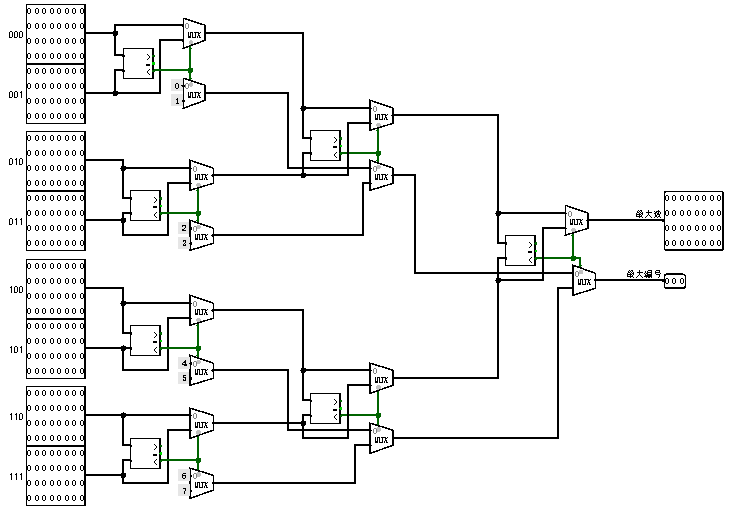


图 3.28 最大数选择电路

如图 3.29是BHT各寄存器的使能信号产生电路。Bignum是图 3.27中选出的最大数表项编号，invalid是用优先编码器生成的未使用表项编号，当有表项为空时，选择invalid，否则，需要发生替换，选择bignum。Hitpos\_3是在取指段命中的表项传到执行段以后再传回的值，hit3也是传回的命中标志，作为选择器的控制信号。当hit3为0，未命中，需要写入新表项，所以选择bignum/invalid，hit3为1表示命中，选择命中的表项编号。将编号传入解码器，输出为1的一路表示当前激活的表项。Jb3是执行段传回的信号，表示指令是跳转指令，通过与门和译码器的输出相与，得到en0-en7，作为有效位、预测历史位的使能信号；en0-en7进一步与hit3取反后的信号相与，得到en0addr-en7addr，作为指令地址、目标地址寄存器的使能信号。

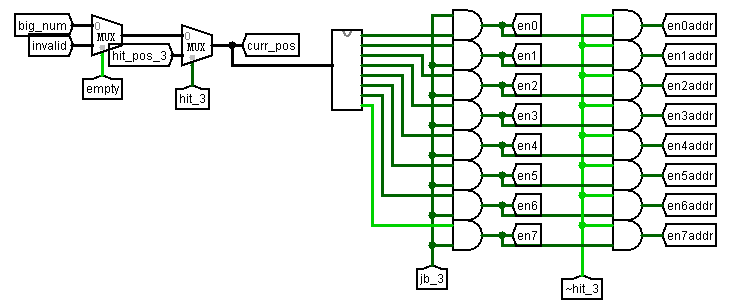


图 3.29 使能信号生成电路

### 地址转移逻辑实现

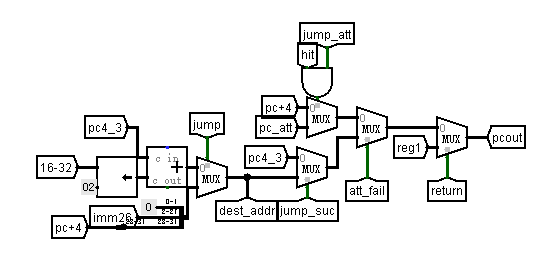


图 3.30 地址转移逻辑

重定向流水线中的跳转指令周期与分支预测中不相同，所以地址转移逻辑也需要修改。如图 3.30，从最左侧的多路选择器开始，选择信号jump为无条件分支标志，为1时选择26位立即数生成的目标地址，为0时表示有条件分支，选择16位立即数生成的目标地址；多路选择器的输出destaddr输出到BHT中，写入分支目标地址寄存器。Jumpsuc表示实际跳转成功，为0时选择pc4\_3，也就是按照重定向流水线中的方式在执行段跳转，否则，选择destaddr；att\_fail是分支预测失败的标志，为0表示预测成功，选择从BHT输出的pc\_att，也就是命中的表项取出的分支目标地址，否则选择上一个选择器的输出；最后一个多路选择器选择信号return为jr指令标志，为1时选择31号寄存器的值，否则选择上一个寄存器的输出。

# 实验过程与调试

## 功能测试和性能分析

对各个任务分别单独测试，除多级中断要执行中断服务程序外，其他任务都用benchmark标准程序进行测试。在benchmark程序的最后还添加了四条扩展指令的测试程序，每执行一条暂停一次，手动按键继续执行。

### 单周期CPU

用benchmark\_ccmb标准程序进行测试。

1. Logisim测试

运行完benchmark的基础指令后的四个周期数如图 4.1所示，符合预期。

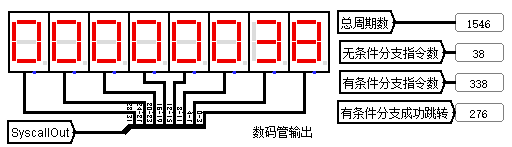


图 4.1 单周期logisim测试周期数

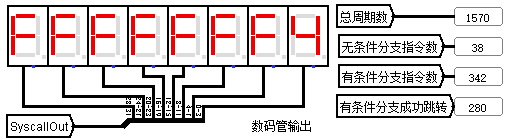


图 4.2 blez指令测试输出

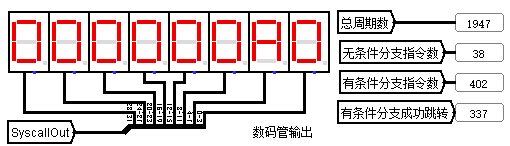


图 4.3 lbu指令测试输出

继续执行ccmb指令，该过程重在观察数码管输出，此处无法演示动态过程，只截取部分中间结果，如图 4.2到图 4.5所示。

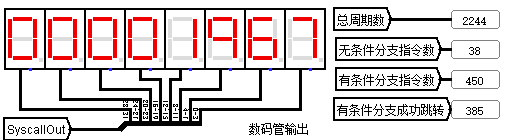


图 4.4 sltiu指令测试输出

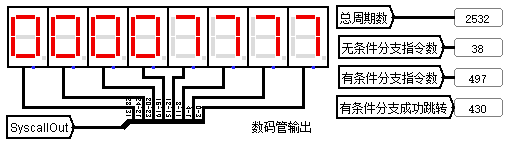


图 4.5 xori指令测试输出

1. FPGA测试

在FPGA开发板上运行标准测试程序，最终显示结果如图 4.6所示，与logisim结果相同。

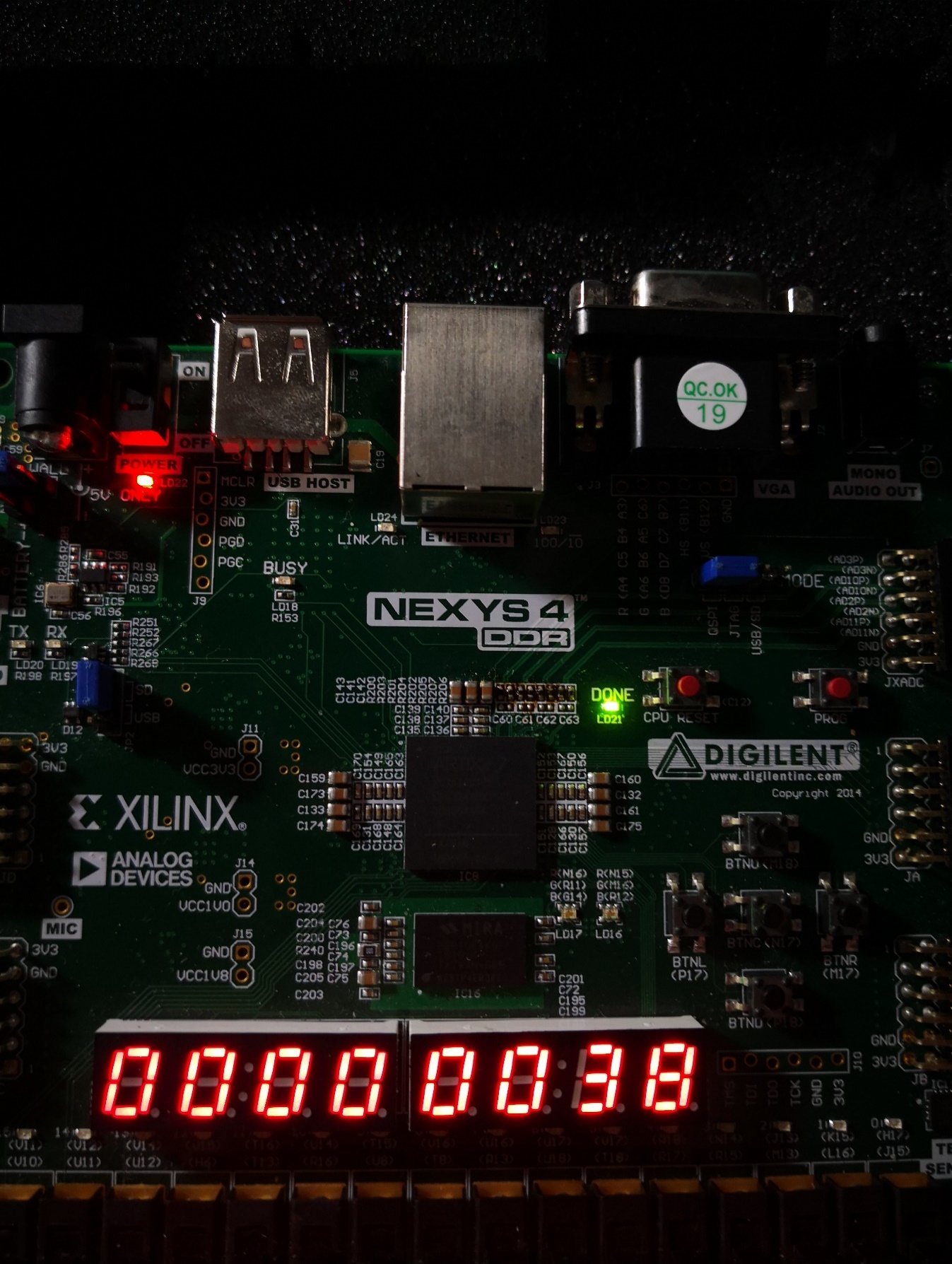


图 4.6 单周期cpu上板测试结果

### 多级嵌套中断

使用中断服务程序 + 正常执行程序的方式进行测试。中断服务程序分别是1、2、3的循环左移，正常执行程序从0x200递减。

先正常执行，输出如图 4.6所示，紧接着依次按下中断按钮2、3、1，依次执行的中断程序为2、3、2、1，分别如图 4.7到图 4.10所示。执行完成后再次回到主程序，如图 4.11所示。

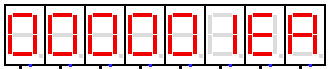


图 4.7 主程序执行

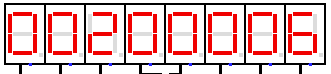


图 4.8 中断服务程序2

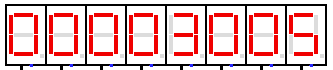


图 4.9 中断服务程序3

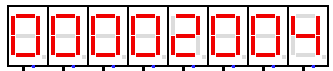


图 4.10 再次回到中断服务程序2

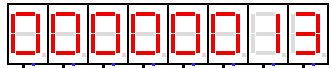


图 4.11 中断服务程序1

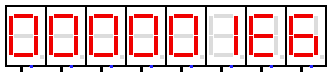


图 4.12 再次回到主程序

### 理想流水线

用几条简单的顺序执行指令进行测试，理想周期数应为21，如图 4.12所示，实测总周期数为21。运行完毕后数据存储器中存储的数据如图 4.13所示，前四个字节分别存有0、1、2、3。

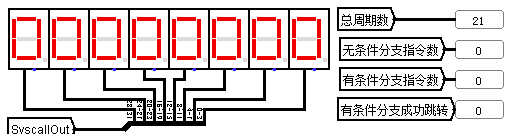


图 4.13 理想流水线总周期数



图 4.14 数据存储器内容

### 气泡流水线

1. 测试：用benchmark-ccmb测试程序进行测试，运行完毕后各周期数如图 4.14所示。
2. 性能分析：

总周期数3624，与单周期cpu的1546相比增加了一倍多。虽然每个周期的时间比单周期要短，但是有大量时钟周期cpu在空转，插入了多大1447个气泡，性能不好。

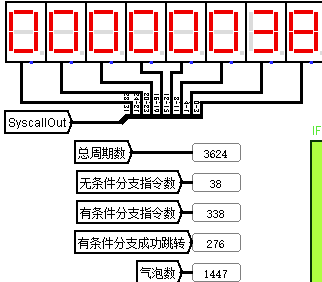


图 4.15 气泡流水线运行结果

### 重定向流水线

1. Logisim测试：用benchmark-ccmb测试程序进行测试，运行完毕后各周期数如图 4.15所示。

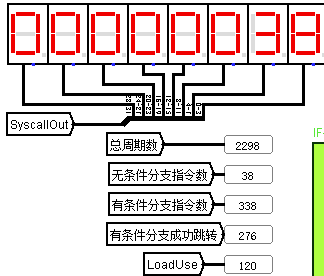


图 4.16 重定向流水线运行结果

1. FPGA测试：

在FPGA开发板上运行标准测试程序，跑马灯效果和各周期数均与logisim中相同，load-use数如图 4.17所示。

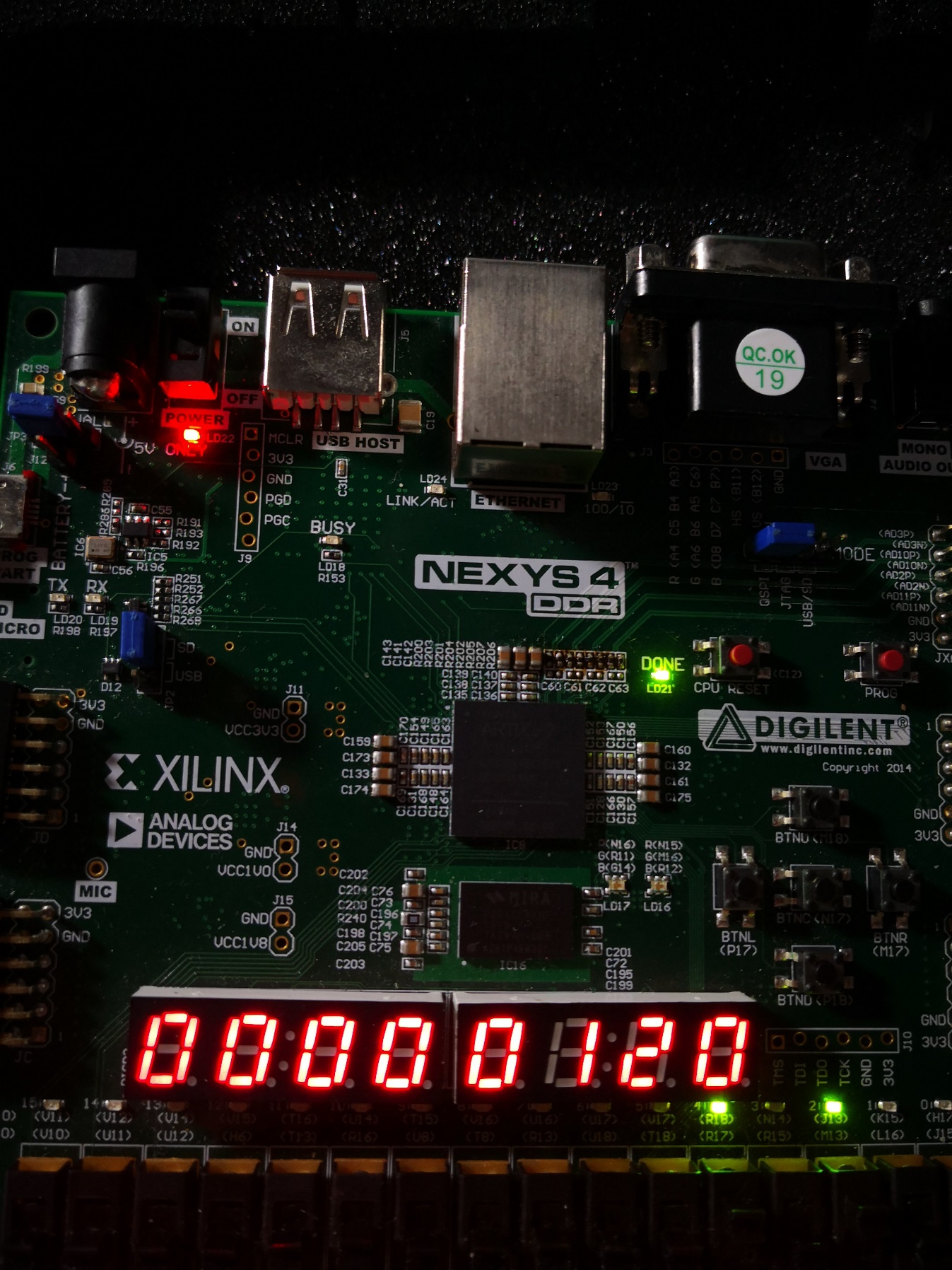


图 4.17 重定向流水线上板测试结果

1. 性能分析：

总周期数2298，气泡数为120，比气泡流水线减少了1000多个周期，性能极大提升，不过分支跳转时浪费的周期达到了（38+338-276）\*2，仍然可以继续优化。

### 动态分支预测

1. 测试：用benchmark-ccmb测试程序进行测试，运行完毕后各周期数如图 4.16所示。

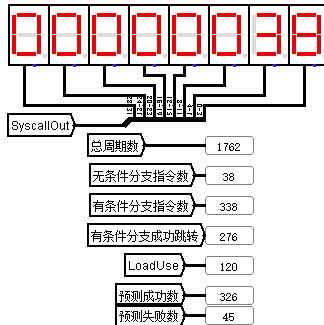


图 4.18 动态分支预测运行结果

1. 性能分析：

总周期数1762，气泡数120，跳转指令浪费的周期数为90，相较于重定向流水线减少了很多。1762与单周期的1546已经比较接近，再考虑到流水线中每个周期的时间更短，所以分支预测流水线的性能比原来的单周期cpu要好很多，基本上达到了极限。如果需要继续优化，则需要在指令级并行的软硬件优化上着手，增加硬件，编译时优化，这已经不是本课程要考虑的问题。

## 主要故障与调试

### 数据存储器综合故障

单周期CPU： 顶层模块综合问题。

**故障现象：**执行顶层模块的综合时，在进行到DM（数据存储器）模块时会停滞，既没有报错，也没有综合成功，无论等待多久都没有效果，vivado可能进入了某种死循环。如图 4.17所示，前面模块的综合都成功，在DM的仿真处停滞不前。

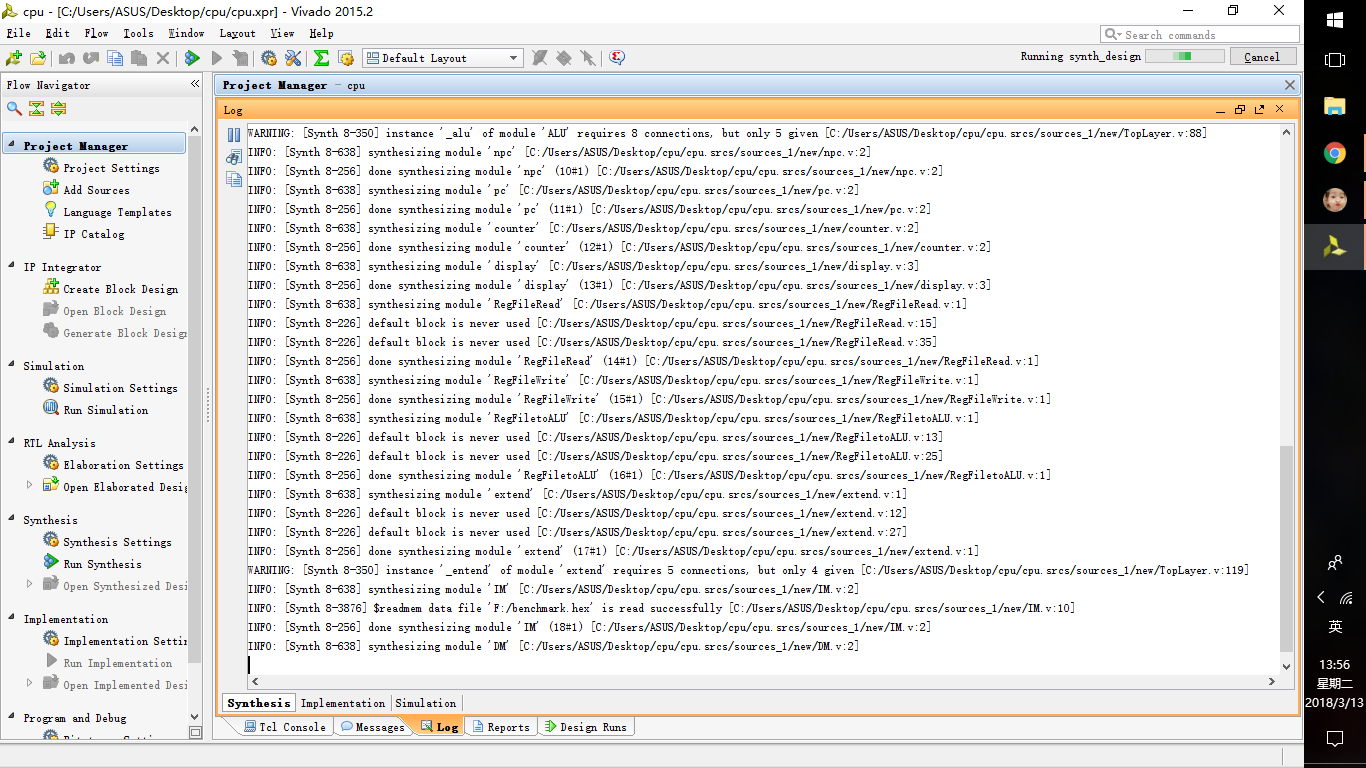


图 4.19 数据存储器综合报错信息

**原因分析：**综合过程的提示信息如图 4.18，这一类的报错信息还有很多，这里只截取了部分。小组讨论后认为是存储器的分配的空间太大，询问老师后排除了这一可能。然后对DM模块的语句逐一注释，再次综合，以确定故障模块的位置。在将其中的一个for循环注释以后，综合成功，因此可以锁定是for循环的使用不当导致了故障。

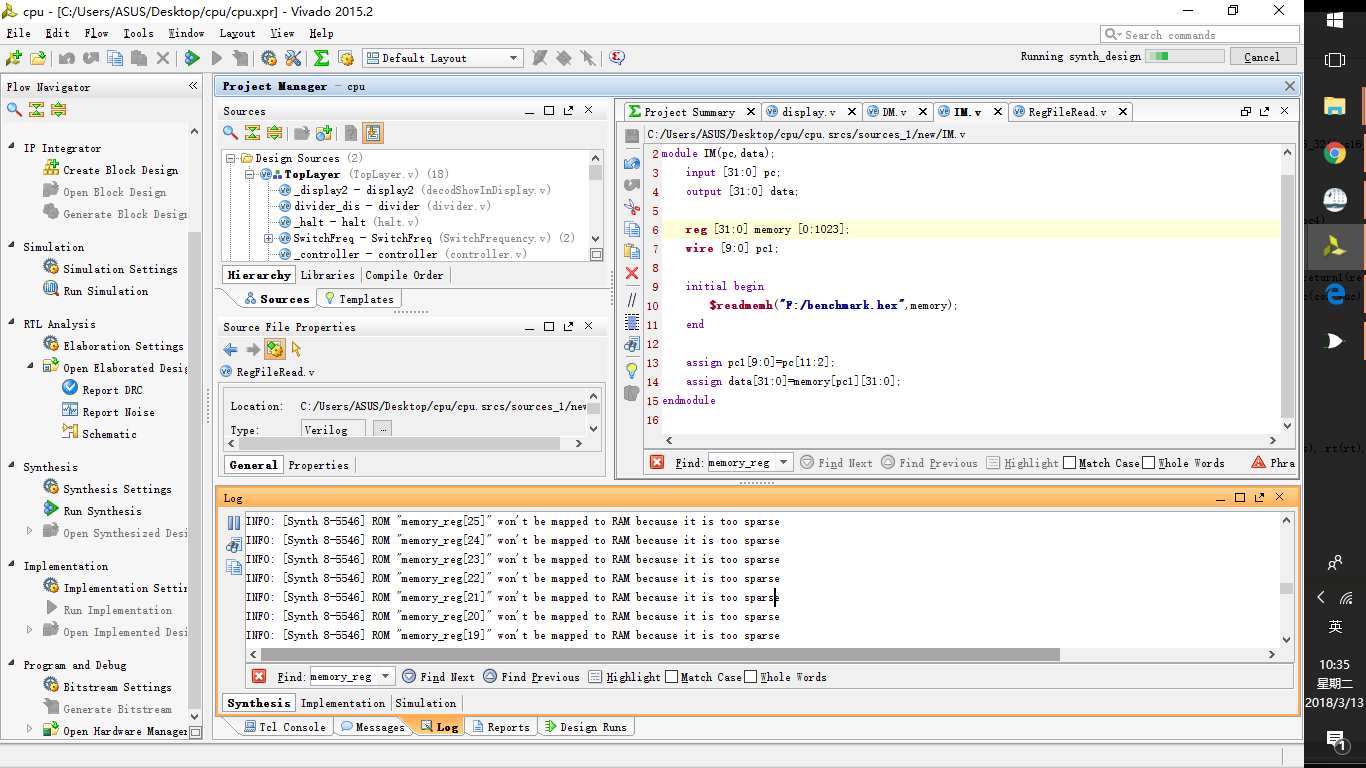


图 4.20 数据存储器综合报错示意图

**解决方案：**for循环是用来对数据存储器进行初始化赋值的，也就是将所有的存储单元存入0，这一功能是可有可无的，因为数据存储器的初值非0即1，不会有不确定状态，而且对内存的操作都是先写后读，所以可以不用初始化，直接删去初始化语句即可。

### 总周期数故障

单周期CPU：benchmark程序运行总周期数与预期不符。

**故障现象：**上板运行，跑马灯效果符合预期，但总周期数显示为1556，有条件成功跳转次数位271，这两者的正确值分别为1546和276。

**原因分析：**实测周期与理想周期相差不是很大，应该某几条指令的逻辑不正确，导致少数情况下计数错误，而且跑马效果正确。为了确定是哪些指令有误，对j指令、b指令、跑马测试、排序等小程序分别单独测试，结果j指令、b指令和跑马测试的运行结果和周期计数均与logisim上的结果一致，而排序测试的总周期多了10，有条件分支成功少了5，这与benchmark的错误相同，所以可以断定benchmark的错误全部发生在排序程序段中。对比发现排序程序中的lw、sw指令较多，所以可能是访存错误。

**解决方案：**数据存储器读取数据部分的Verilog代码如下，程序逻辑看起来正确，仔细观察才意识到使用了时序逻辑，而存储器读数据应该是组合逻辑，只要送入地址，就应该立马得到数据。将always行为描述模块改用assign语句赋值即可实现组合逻辑读取数据。同样，在寄存器文件中也存在着类似的问题，所以一并对寄存器文件也进行了读数据的组合逻辑修改。

always @(posedge clk) begin

if(mode[3]) Dout[31:24]=memory3[addr1][7:0];

else Dout[31:24]=8'b00000000;

if(mode[2]) Dout[23:16]=memory2[addr1][7:0];

else Dout[23:16]=8'b00000000;

if(mode[1]) Dout[15:8]=memory1[addr1][7:0];

else Dout[15:8]=8'b00000000;

if(mode[0]) Dout[7:0]=memory0[addr1][7:0];

else Dout[7:0]=8'b00000000;

end

### 气泡流水线分支故障

气泡流水线： 跑马效果不正确，分支跳转错误。

**故障现象：**对气泡流水线进行benchmark程序测试，前一段时间跑马灯执行正常，到移位测试阶段，如图 4.19所示，本该一直向右移动的数字2没有移动，执行到数字3的移位时，显示效果如图 4.20，也没有移动，后续的数字显示效果与之类似。

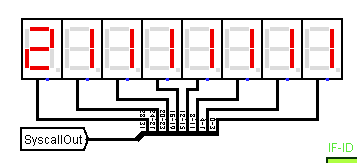


图 4.21 移位测试跑马故障1

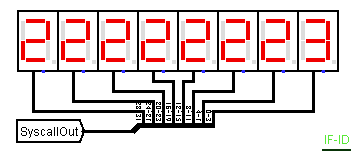


图 4.22 移位测试跑马故障2

**原因分析：**程序能够按顺序显示程序的跑马输出，说明程序在有序执行，跳转指令的逻辑没有问题，但是输出的结果没有移位，说明是移位指令出了错或者根本没有被执行，也就是跳转的位置不对，这导致了每次循环时有两条语句没有执行，而这两条语句中恰好有移位指令，所以导致跑马效果没有移位。

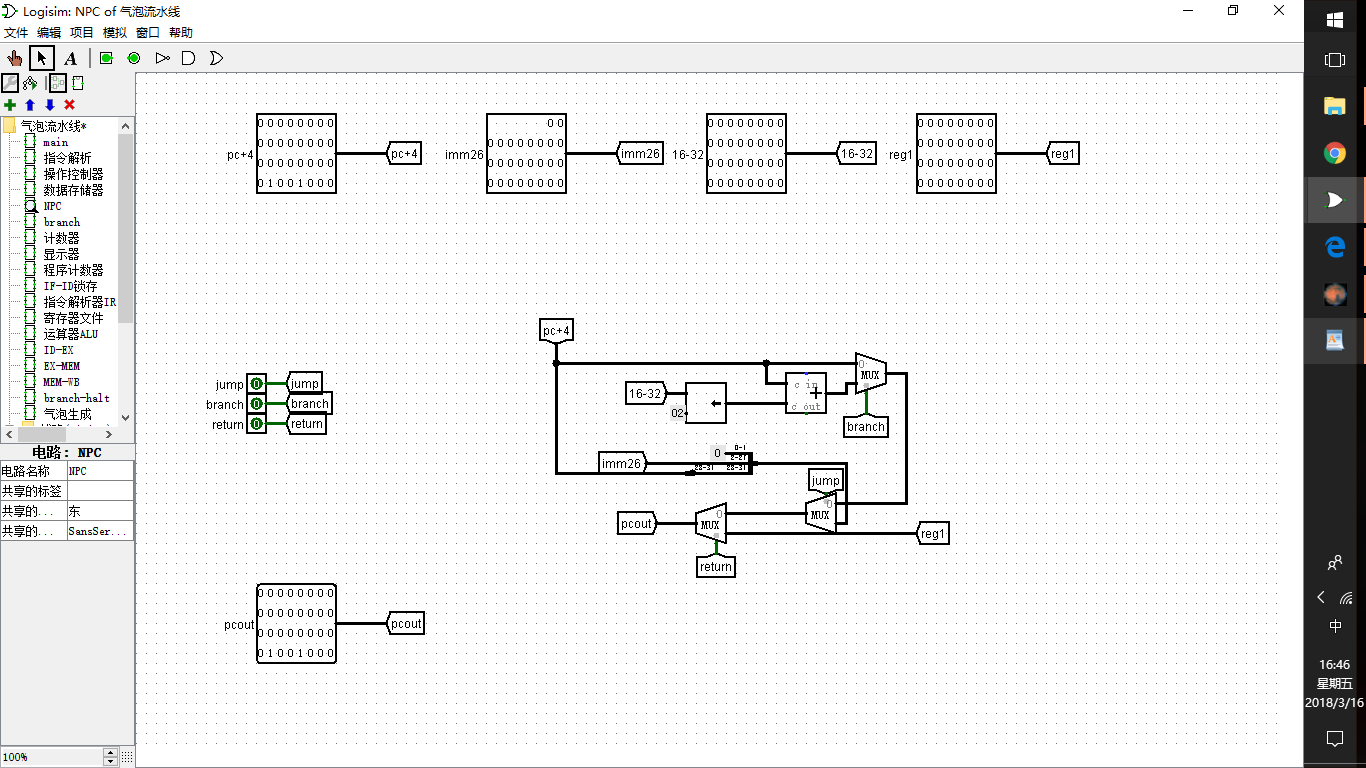


图 4.23 地址转移逻辑

**解决方案：**对移位测试程序进行单步执行，其中涉及到循环，当循环判断条件不成立时往回跳转，发现跳转的位置不正确，总是在正确位置的下下条指令处，也就是pc值比正确值大8，问题一定出在npc地址转移逻辑。Npc转移逻辑电路如图 4.21，当b指令成功跳转时，选择的目标地址是pc+4加上16位立即数扩展的值，理论上没有问题，但是该电路调用时，传入的pc+4是从IF段接入的，而b指令跳转发生在ex段，两者相差两个周期，所以pc+4的值差了8，故应该送入ex段的pc+4值。

### 重定向流水线跑马故障

重定向流水线：跑马灯效果不正确。

**故障现象：**对重定向流水线进行benchmark程序测试，程序的跑马效果从一开始就完全不对，紧接着程序执行完全混乱，如图 4.22所示。

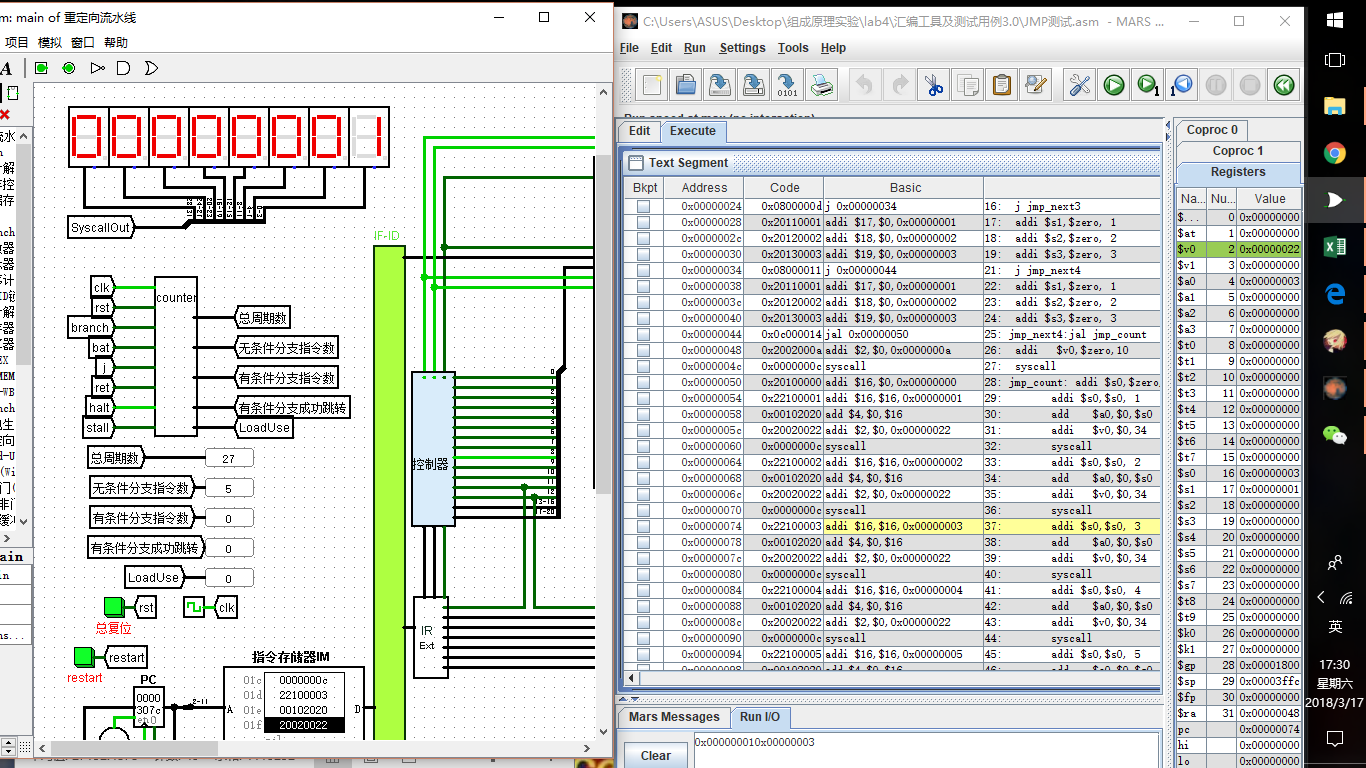


图 4.24 重定向跑马灯故障

**原因分析：**程序运行出现了这种完全不正确的情况很难定位错误，只能单步执行。将benchmark测试程序在Mars中打开，logisim也同时打开，两者交替单步执行，每执行完一步观察logisim中的寄存器和数据存储器的值是否与Mars中相同。在执行了数十条指令后，寄存器和存储器的值都正确，但是syscall应该在数码管显示的值没有显示出来，所以是显示模块出了问题。

**解决方案：**显示模块是从原来的气泡流水线继承的，不应该有问题，所以显示错误是因为输入的数据有错误，观察发现输入显示模块的寄存器的第二个输出是重定向以前的数据，也就是说重定向的数据成功送入了alu，但是没有成功送到显示模块，导致程序正确执行却得不到正确的显示效果，将重定向以后的数据送到显示模块即可。

### 分支预测替换故障

动态分支预测：旧表项替换逻辑错误。

**故障现象：**对分支预测流水线进行benchmark测试，跑马效果正确，但是总周期数与重定向流水线的2298相比，仅仅减少了不到100，如图 4.23所示。

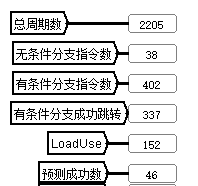


图 4.25 分支预测故障示意图

**原因分析：**该分支预测流水线的功能完全正确，但是性能不好，预测成功数太少，总周期数应该在2000以下。Bht中有8个表项，这不是个很小的数目，理应能命中多次，然而结果不如人意，一定是替换策略存在问题。

**解决方案：**先让测试程序执行一段时间，让bht表填满，然后再单步执行，遇到分支指令时，进入bht中观察，发现没有命中，则该分支的地址应该替换表中lru值最大的表项，结果发现替换的永远都是第0或者第1项，而不是lru最大的项，最大数的选择存在问题。

如图 4.24所示，在最大数选择电路中发现所有的多路选择器的输入端都接的是0、1，这导致选出的最大数的编号永远为0或者1。因为在画电路时很多重复的模块都是直接复制的，复制以后忘记修改常量的值，将0、1、0、1、0、1、0、1修改为0-7以后，电路功能就正常了。

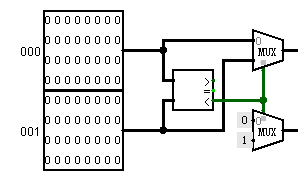


图 4.26 最大数选择逻辑

## 实验进度

表 4.1 课程设计进度表

| 时间 | 进度 |
| --- | --- |
| 第一天 | 进行了模块分工，确定了各模块间的接口，完成了操作控制器、指令解析器和总控模块。 |
| 第二天 | 全天都在仿真、综合调试，现已能够上班运行，指令计数部分功能正常，能够正确执行部分指令，但是跳转指令存在问题，还在调试中 |
| 第三天 | 继续调试单周期cpu，仿真与上板验证相结合，排除各模块故障。一开始无法正常跳转，解决这一问题后，程序总周期不正确，大家一同调试，最终完成了单周期cpu上板测试，最高频率50MHz |
| 第四天 | 对单周期cpu进行模块拆分，一开始由于结构不合理而进行了多次重复的封装。最终顺利完成了理想流水线，没有遇到太大的麻烦。 |
| 第五天 | 完成气泡流水线，运行结果出现了许多问题，调试 过程发现了单周期cpu中的诸多隐藏问题，修改以后的电路结构更为合理。 |
| 第六天 | 周末完成了气泡和重定向流水线，今天在将重定向流水线转换成Verilog语言，准备做流水线上板。 |
| 第七天 | 修改重定向Verilog代码中的部分bug，成功实现上板 |
| 第八天 | 基本完成单级中断，还有部分问题。 |
| 第九天 | 单级中断提交检查，还完成了多级嵌套中断。 |
| 第十天 | 完成BHT和分支预测的数据通路，正在进行功能调试。 |

# 设计总结与心得

## 课设总结

本次课程设计主要作了如下几点工作：

1. 完成了logisim到Verilog的转换，在FPGA上实现了单周期cpu；在单周期cpu上加入中断逻辑，实现了多级嵌套中断；对单周期cpu进行了合理的划分，得到了理想流水线；对理想流水线加入冲突处理逻辑，实现了气泡流水线；对气泡流水线进行优化，实现了重定向流水线；对重定向流水线加入bht，实现了动态分支预测。
2. FPGA上的单周期cpu与logisim上的单周期cpu功能相同，最高运行频率达到了50MHz；气泡流水线实现了流水执行，但性能低下；重定向流水线性能相比气泡流水线得到了大幅度提升；动态分支预测流水线改进了重定向流水线，增加了分支缓冲表，实现了性能的进一步提升。
3. 修正了课程实验中的单周期cpu中的部分bug，实现了对ccmb指令的支持；通过实践弄懂了很多问题，对cpu性能提升策略有了一定的了解；发现并改正了原来常犯的许多低级错误。

## 课设心得

虽然没有实现完全通关，但是拿到了105分，完成度稍微高于预期，对自己的工作比较满意。课设两周期间大脑一直处于高负荷运转状态，睡醒以后立马就能想起还有电路没有画好，周末也没有休息多少。虽然做得不算快，但是还比较顺利，每个任务都会遇到一些小问题，但是单步调试基本都能解决。将单周期cpu改造成流水线以后发现无法运行ccmb测试程序，这是因为课程实验中实现的单周期cpu没有ccmb标准测试程序，只用自己写的程序稍微验证了一下，没有涵盖所有的可能性，所以还存在问题。类似于这样的问题还有很多，每次都是做到高阶的任务时发现低阶版本中存在问题，这主要还是因为对知识的掌握不熟练，导致数据通路和控制信号的设计有不周全之处。

团队建设不太成功，大家合作的单周期cpu上板在平均水平以上，但是各自分工以后，某些队员的速度就明显变慢了，有人在气泡流水线上花费了好几天时间，遇到问题没有及时与其他人讨论，总是自己蛮干。虽然小组内所有人都至少拿到了70分，但是团队建设仍然不算成功，没有形成合作学习的氛围。

应该允许自由组队，这样的结果是关系好的人都分在同一组，大家一定会心齐，弊端就是学习差的人可能找不到队友，不过他们是可以到其他组问学习好的同学的。要做得开心就要和自己中意的队友一起，无所谓各自的水平，不会的问题大家一起讨论，或者一起去问其他人。七班的完成度最高，我不清楚他们具体的分组方法，老师也可以询问一下他们，参照他们的模式进行分组。不过每个班的风气不同，可能没有一个通用的方法。

最先通关的同学在课设结束前几天就完成了所有的任务，紧接着也有很多人通关，这说明课设的难度不够大，最优秀的学生拿到110分是因为他没有其他任务可以做了，只能拿110分，而其他人的110分是努力了很久才刚好拿到的，这两个110是有区别的。也不是说非要把差距拉开，但还是要让很早做完的同学有事可做，应该设置一个终极关卡，难度控制到只有极个别人能够实现。

课设完成了以后还有意犹未尽的感觉，虽然不想再来一遍了，但是还是一直在回味，的确很难忘，感谢老师为课程付出的心血。任务书中还有少许错别字，希望能够更正。

# 参考文献

1. DAVID A.PATTERSON(美).计算机组成与设计硬件/软件接口(原书第4版).北京：机械工业出版社.
2. David Money Harris(美).数字设计和计算机体系结构（第二版）. 机械工业出版社
3. 秦磊华，吴非，莫正坤.计算机组成原理. 北京：清华大学出版社，2011年.
4. 袁春风编著. 计算机组成与系统结构. 北京：清华大学出版社，2011年.
5. 张晨曦，王志英. 计算机系统结构. 高等教育出版社，2008年.

|  |
| --- |
| 一、原创性声明 |
| 本人郑重声明本报告内容，是由作者本人独立完成的。有关观点、方法、数据和文献等的引用已在文中指出。除文中已注明引用的内容外，本报告不包含任何其他个人或集体已经公开发表的作品成果，不存在剽窃、抄袭行为。  特此声明！  **作者签字:** |