***2019***



**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： | CS160X |
| 学 号： | U20091224 |
| 姓 名： | 李珍帅 |
| 电 话： | 1345565666 |
| 邮 件： | [13456@qq.com](mailto:13456@qq.com) |

目 录

[1 课程设计概述 1](#_Toc474706961)

[1.1 课设目的 1](#_Toc474706962)

[1.2 设计任务 1](#_Toc474706963)

[1.3 设计要求 1](#_Toc474706964)

[1.4 技术指标 1](#_Toc474706965)

[2 总体方案设计 1](#_Toc474706966)

[2.1 单周期CPU设计 1](#_Toc474706967)

[2.2 中断机制设计 1](#_Toc474706968)

[2.3 流水CPU设计 1](#_Toc474706969)

[2.4 气泡式流水线设计 1](#_Toc474706970)

[2.5 数据转发流水线设计 1](#_Toc474706971)

[2.6 动态分支预测机制（未完成部分可不写，直接删除） 1](#_Toc474706972)

[3 详细设计与实现 1](#_Toc474706973)

[3.1 单周期CPU 实现 1](#_Toc474706974)

[3.2 中断机制实现 1](#_Toc474706975)

[3.3 流水CPU实现 1](#_Toc474706976)

[3.4 气泡式流水线实现 1](#_Toc474706977)

[3.5 数据转发流水线实现 1](#_Toc474706978)

[3.6 动态分支预测机制实现 1](#_Toc474706979)

[4 实验过程与调试 1](#_Toc474706980)

[4.1 测试用例和功能测试 1](#_Toc474706981)

[4.2 可自行安排章节 1](#_Toc474706982)

[4.3 性能分析 1](#_Toc474706983)

[4.4 主要故障与调试 1](#_Toc474706984)

[4.5 实验进度 1](#_Toc474706985)

[5 设计总结与心得 1](#_Toc474706986)

[5.1 课设总结 1](#_Toc474706987)

[5.2 课设心得 1](#_Toc474706988)

[参考文献 1](#_Toc474706989)

# 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，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。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前27条基本32位MIPS指令；
2. 支持教师指定的4条扩展指令；
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. 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 | 中断相关，可简化，选做 |
| 26 | MTC0 | 访问CP0 | 中断相关，可简化，选做 |
| 27 | ERET | 中断返回 | 异常返回，选做 |
| 28 | XOR | 异或 |  |
| 29 | LUI | 立即数加载至高位 |  |
| 30 | LB | 加载字节 |  |
| 31 | BLEZ | 小于等于 0 转移 |  |

# 总体方案设计

## 单周期CPU设计

单周期CPU的设计我们采用硬布线控制进行实现。在本次实验中首先需要完成单周期CPU的设计，然后在其基础之上完成老师布置的4条扩展指令的添加。将系统分成指令存储器、硬布线控制器、寄存器文件、运算器、数据存储器、地址转移逻辑六大模块。根据各条指令实现的功能，完成数据通路。为了避免取指令和取数据造成的数据冲突，在这里我们采用了哈佛结构，把指令存储器和数据存储器分开，用硬件实现的方式完美地解决了这一问题。总体结构图如图 2.1所示。



图 ‑ 总体结构图

### 主要功能部件

#### 程序计数器PC

程序计数器主要负责存储和输出所取指令的地址，在时钟上升沿到来时输出取指令的地址并且存储下一个取指令的地址。与此同时，计数器还需具有使能端满足停机指令的要求，因此我们可以用一个寄存器实现。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 | 加载到寄存器的值 |
| PC\_out | 输出 | 32 | 寄存器输出的值 |

#### 指令存储器IM

指令存储器只负责指令的读取，MIPS指令位宽为32位，4个字节。因为在本实验中指令的数量并不是很庞大，所以我们取指令时需要去掉低两位，然后再取低10位即可获得指令的地址。所以在这里我们可以用32位ROM存储器实现。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 |
| --- | --- | --- | --- |
| A | 输入 | 10 | 指令地址 |
| D | 输出 | 32 | 执行指令 |

#### 数据存储器DM

数据存储器负责数据的读取和写入，数据位宽为32位，4个字节。因为在本实验中数据的数量并不是很庞大，所以我们在访问数据时需要去掉低两位，然后再取低10位即可获得数据的地址。所以在这里我们可以用32位RAM存储器实现。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 |
| Addr | 输入 | 10 | 读取数据的地址 |
| Din | 输入 | 32 | 写入的数据 |
| Dout | 输出 | 32 | 读出的数据 |

#### 寄存器堆RF

寄存器堆里面包含了MIPS里所需要的所有的寄存器。寄存器堆可以同时读出两个寄存器的数据，并且可以根据写使能信号WE来判断是否需要将输入数据Din存储到W#处。输入输出引脚描述如表 2‑4

表 ‑ 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 |
| WE | 输入 | 1 | 写使能信号 |
| R1# | 输入 | 5 | 读取数据的地址 |
| R2# | 输入 | 5 | 读取数据的地址 |
| W# | 输入 | 5 | 写入的地址 |
| Din | 输入 | 32 | 写入的数据 |
| R1 | 输出 | 32 | 读出的数据 |
| R2 | 输出 | 32 | 读出的数据 |

#### 运算器

运算器接受两个32位操作数A和B进行运算，根据S端输入运算器的操作码来对操作数进行相应的运算。无论两个操作数进行何种运算，运算器都会比较它们是否相等并且从=端输出。对于操作数Y的移位的运算，所移位数可以直接从shamt端输入。输入输出引脚描述如表 2‑5

表 ‑ ALU引脚与功能描述

| 引脚 | 输入/输出 | 位宽 | 功能描述 |
| --- | --- | --- | --- |
| A | 输入 | 32 | 操作数A |
| B | 输入 | 32 | 操作数B |
| S | 输入 | 4 | 操作码，决定运算方式 |
| shamt | 输入 | 5 | 操作数Y所移位数 |
| R | 输出 | 4 | 运算结果 |
| = | 输出 | 1 | 两操作数是否相等的标志 |

操作码对应的运算方式如表 2‑6

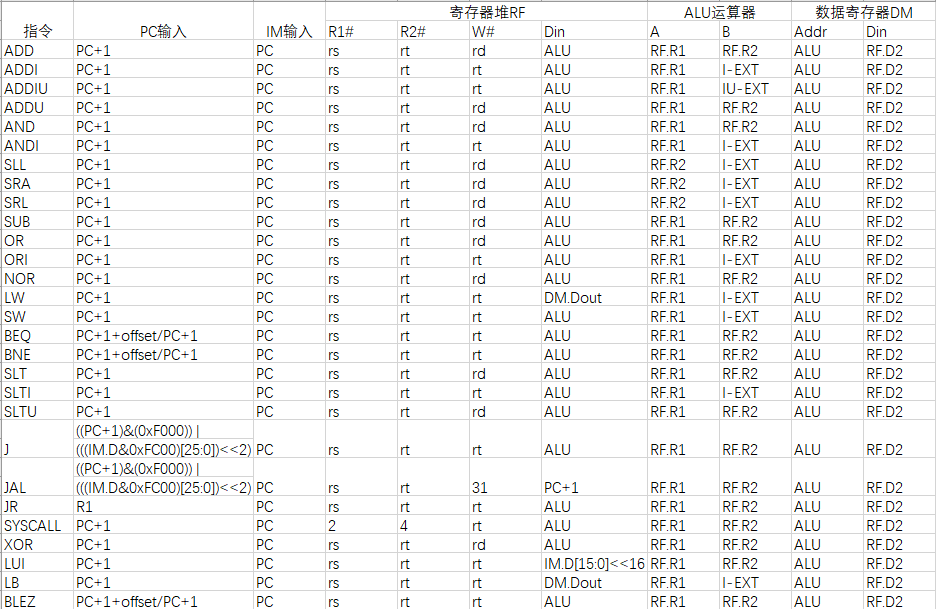
表 ‑操作码功能描述

| 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 |

### 数据通路的设计

对于主要的功能部件，下面的表格列出了功能部件的数据来源。当功能部件对于默写输入用不到时，表格中采用了默认输入的方式。也就是说，输入的数据不一定在该功能部件中有效。得到的数据通路部件输入输出为表 2‑7

表 2‑7 数据通路部件输入输出



### 控制器的设计

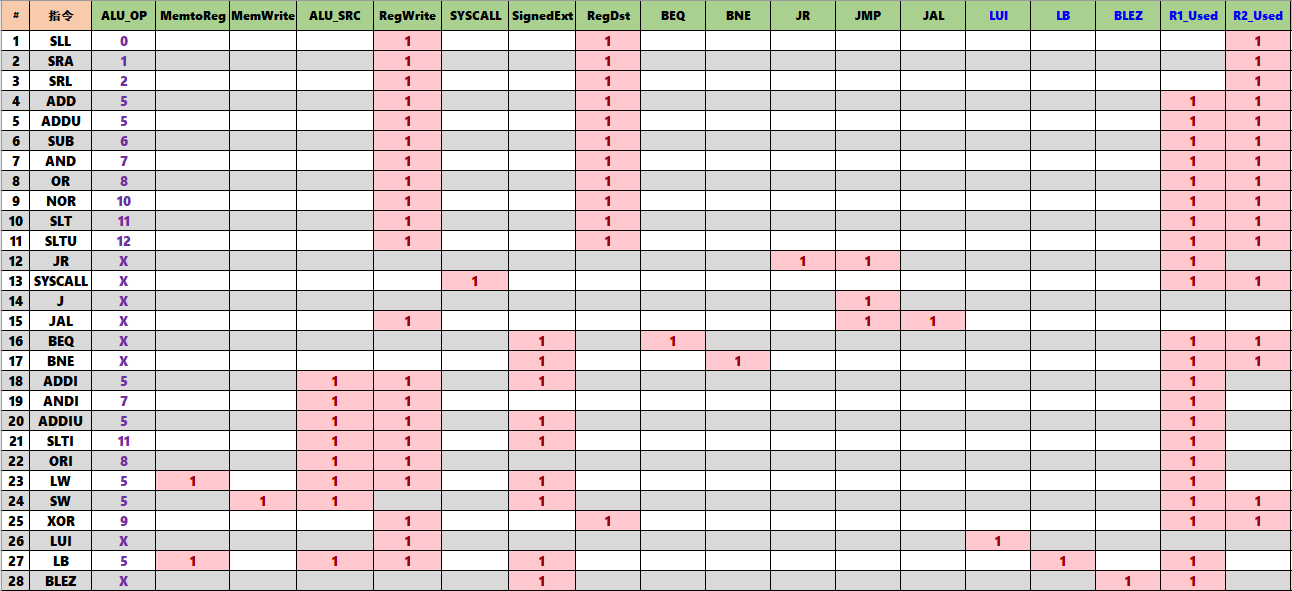
首先对于控制信号进行统计，包括各个主要部件所需要输入的控制信号，以及数据通路合并表中所示的具有多输入的主要部件需要进行输入选择的控制信号，并且对各个统计信号的各种取值情况进行定义，统计得到的控制信号以及说明如表 2‑8

表 2‑8主控制器控制信号的作用说明

| 控制信号 | 取值 | 说明 |
| --- | --- | --- |
| MemToReg | 0/1 | 写入寄存器的数据来自存储器 |
| MemWrite | 0/1 | 写内存控制信号 |
| ALU\_SRC | 0/1 | 运算器B端口的输入选择 |
| RegWrite | 0/1 | 寄存器写使能控制信号 |
| SysCALL | 0/1 | 系统调用信号，判断停机还是数码管显示 |
| SignedExt | 0/1 | 是否为符号扩展的判断信号 |
| RegDst | 0/1 | 写入寄存器选择控制信号 |
| Beq | 0/1 | Beq指令译码信号 |
| Bne | 0/1 | Bne指令译码信号 |
| JR | 0/1 | JR指令译码信号 |
| JMP | 0/1 | J型指令译码信号 |
| JAL | 0/1 | JAL指令译码信号 |
| LUI | 0/1 | LUI指令译码信号 |
| LB | 0/1 | LB指令译码信号 |
| BLEZ | 0/1 | BLEZ指令译码信号 |
| ALU\_OP | 0x0-0xF | 运算器操作控制码 |

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，对于与本条指令无关的控制信号，控制信号的取值一律为0，以简化控制器电路的设计。该控制信号表的框架如表 2‑9

表 2‑9主控制器控制信号框架



## 多级中断机制设计

### 总体设计

多级中断响应的执行流程图如图 2‑2

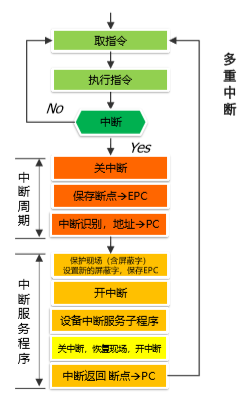


图 2‑2多级中断响应流程

在本次实验中，采用了独立请求的方式实现中断。不同的中断对应不同的入口地址，在这里我们入口地址都是固定的，因此我们可以用常量保存中断程序的入口地址。当不同的中断产生时，我们可以根据中断产生的信号源，通过一个多路选择器来选择入口地址常量。

中断发生时，需要保护现场，而多级中断是可以发生嵌套中断的，所以我们需要在保护现场的时候不能发生中断嵌套。所以多级中断还需要开中断和关中断的指令，在这里我们可以用极简的的方式实现，让MFC0和MTC0分别充当为开中断和关中断指令。

当中断执行完毕的时候，我们需要回到程序的中断处继续执行程序，因此我们需要把中断的地址保存下来。因为在本次实验中只有三个中断源，因此返回地址的存储我们可以通过硬件堆栈的方式。

### 硬件设计

1. 硬布线控制器的修改

因为我们需要添加两条特殊的指令MFC0和MTC0，所以需要堆控制器进行简单的修改。MFC0和MTC0与常规指令的判断有所不同，我们的控制器需要新加输入端来判断MFC0和MTC0指令，同时需要添加两个输出端表示这两个指令的译码信号。

1. 硬件堆栈的实现

三个中断发生中断嵌套时，中断前的下一条指令需要保存起来，用来完成中断后现场的恢复工作。这里我们用硬件堆栈的方式实现，用硬件的方式替代软件上的入栈和出栈。

1. 中断优先级判断

中断嵌套发生的条件是当前中断的优先级高于正在处理的中断的优先级，我们需要用硬件的方式来判断当前是否能够发生中断嵌套。

### 软件设计

硬件堆栈的方式只能实现EPC值的保护，而对于某些寄存器的值改变却无能为力。因此，我们在设计中断服务程序的时候，在中断发生后中断程序执行前需要对中断程序中修改的寄存器通过sw指令保存起来。中断程序结束后，我们需要用lw进行现场的恢复。

## 流水CPU设计

### 总体设计

MIPS 单周期 CPU 设计实现简单，控制器部分是纯组合逻辑电路，但该 CPU 所有指令执行时间均是一个相同的周期，即以速度最慢的指令作为设计其时钟周期的依据。为了提高CPU 的效率，我们可以把CPU指令执行过程设计成流水的阶段。这样，下一个阶段处理数据的同时上一个流水阶段同样可以执行其它工作，流水线的逻辑架构和时空图如图 2‑3和图 2‑4

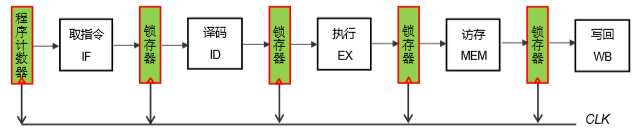


图 2‑3指令流水线逻辑架构

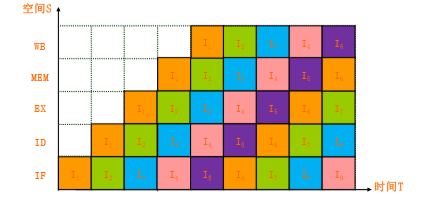


图 2‑4五段流水MIPS CPU时空图（无冲突）

### 流水接口部件设计

MIPS 指令流水线通常将指令执行过程细分为取指令 IF、 指令译码 ID、指令执行 EX、访存 MEM、写回 WB 共五个阶段，在每个阶段的后面都需要增加一个锁存器（又称为流水接口部件，用于锁存本段处理完成的数据或结果），以保证该阶段的执行结果给下一个阶段使用。每个流水接口部件只需要把下一个流水阶段需要的数据通过寄存器保存起来即可，各段流水接口部件的设计相似，只是每个部件锁存的数据可能有所不同。

### 理想流水线设计

理想流水线的设计比较简单，只需要把指令的执行分为五个阶段，然后链接流水接口部件进行数据的传输即可。

1. 取指：程序计数器PC，指令存储器IM，下一条指令地址的选取与计算
2. 译码：立即数扩展器，硬布线控制器，寄存器堆
3. 执行：运算器ALU
4. 访存：数据存储器DM。
5. 写回：寄存器堆，系统调用的判断，选择写入寄存器的数据来源

## 气泡式流水线设计

理想流水线只适用于非常简单的情况。当发生流水线冲突时，理想流水线便不能够正常的运行。在这里我们主要处理数据冲突和控制冲突。当前指令要用到前面指令的操作结果，而这个结果尚未产生或尚未送达指定的位置，会导致当前指令无法继续执行，称为数据冲突。当流水线遇到分支指令或其它会改变 PC 值的指令时，分支指令后续已经载入流水线的相邻指令可能不能进入执行阶段，这种冲突称为控制冲突。

1. 数据冲突处理

我们选择在ID段判断这条指令与前两条指令的关系。ID 段与WB段的数据相关可以通过寄存器先写后读的方式解决，具体方式是寄存器文件写入数据时采用下跳沿触发，由于寄存器的读出是组合逻辑，数据写入后立刻可获得寄存器新值，所以在下跳沿成功写入后，上跳沿触发流水线进行指令传送时，送入 ID/EX的操作数已经是最新的寄存器值。将ID段读取的寄存器编号与EX和MEM段写寄存器编号相比较，假如有两者相同，那么就发生了数据冲突，但是寄存器0是个特殊情况我们不做处理。发生数据冲突时，IF/ID段暂停，ID/EX段插入一个气泡。

1. 控制冲突处理

我们在EX段判断是否发生了控制冲突，此时的控制冲突即是判断是否需要执行跳转指令。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流水部件插入气泡，清除误取指令。

1. 执行阶段
2. 取指：程序计数器PC，指令存储器IM，下一条指令地址的选取与计算
3. 译码：立即数扩展器，硬布线控制器，寄存器堆
4. 执行：运算器ALU，跳转地址的计算，跳转地址的选取，数据冲突和控制冲突的判断
5. 访存：数据存储器DM。
6. 写回：寄存器堆，系统调用的判断，选择写入寄存器的数据来源

## 数据转发流水线设计

引发数据冲突大量的地插入气泡肯定会影响指令流水线的性能。因为各条指令的指令都是在EX阶段执行，所以我们可以在EX段重定向正确的数据，减少气泡的插入。EX 段的指令可能与MEM段，WB段的两条指令均存在数据相关，此时EX段取得的操作数应该是错误的数据，正确的数据分别存放在EX/MEM以及MEM/WB流水接口部件中，还未写回到寄存器中，此时可以直接将正确数据从其所在位置重定向到 EX段合适的位置，这样就可以避免插入气泡引起的流水线性能下降。

但是有一种特殊的数据相关不能采用重定向计数解决，成为Load-use相关。当前一条指令是lw指令时，我们不能使用重定向解决。假如使用了重定向相关，那么EX段时延=MEM时延+ALU时延，成为了关键路径，流水线性能将会下降。重定向示意图如图 2‑5

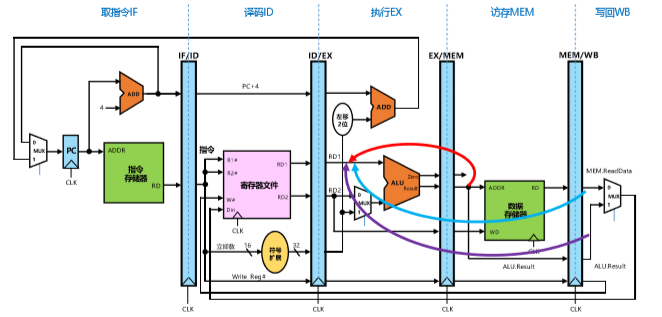


图 2‑5数据重定向示意图

## 动态分支预测机制

重定向流水线对于数据冲突有了很大的优化，但是对于控制冲突却没有优化。在流水线执行跳转指令之前，我们可以预测该跳转指令是否跳转，从而决定跳转指令的下一条指令的读取。动态分支预测流程如图 2‑7

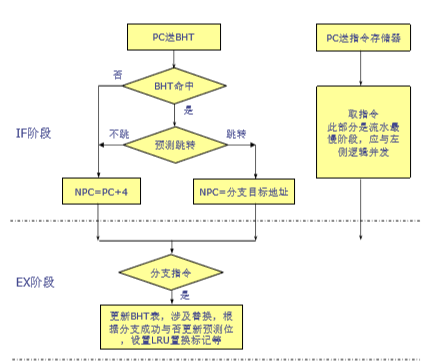


图 2‑6动态分支预测流程流程

动态分支预测主要依赖于BHT表，BHT表本质上就是一个全相联Cache。BHT 表中存放分支指令的地址，分支目标地址，历史跳转信息描述位（分支预测历史位）、有效位以及LRU算法结构。BHT 表会放在取指令阶段，利用 PC的值作为关键字进行全相联比较。

在BHT表中比较重要的就是分支预测历史位。双预测位可在较低的成本下实现很高的预测准确率，所以现在普遍采用双位预测。采用双位预测的BHT状态转换如图 2‑7

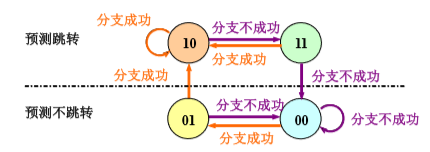


图 2‑7采用双位预测的 BHT 状态转换

# 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。



1. FPGA实现：

程序计数器PC的Verilog代码如下：

always@(negedge clk,posedge clear)

begin

if(clear)

pc\_out<=0;

else if(!halt)

pc\_out<=pc\_in;

end

1. 指令存储器（IM）
2. Logism实现：

使用一个只读存储器ROM实现指令存储器（IM）。设置该只读存储器的地址位宽为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，故将32位指令地址高位部分和字节偏移部分直接屏蔽，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. FPGA实现：

直接使用Vivado中自带的ROM作为指令存储器，其设置如**错误!未找到引用源。**所示。选择ROM的数据位宽为32位，因为该ROM的地址位宽为10位，所以选择ROM的大小选择为1024。

指令存储器IM的Verilog代码如下：

pc pcmeml(im\_in[11:2],im\_out);

直接调用之前设置的ROM作为指令存储器，输入为指令地址的2-11位，输出为该指令。

### 数据通路的实现

本次课程设计采用的工程化的设计模式，一次性构建所有的数据通路。主要实现方法为，对于每一条指令，将其改写成RTL（Register Transfer Level），忽略控制类信号，仅保留数据类信号，根据RTL功能填写对应指令的数据通路表，描述五大部件之间的连接关系，记录各部件输入端数据来源。

根据总体方案设计中数据通路设计那一小节的详细内容，具体分析每一条指令在执行过程中各个主要部件的输入和输出端口的连接，完成指令系统数据通路表的填写，如表 3.1所示。

表 ‑指令系统数据通路表

| 指令 | PC | IM | RF | | | | ALU | | | DM | | Tube |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| R1# | R2# | W# | Din | A | B | OP | Addr | Din |
| ADD | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 5 |  |  |  |
| ADDI | PC+4 | PC | rs |  | rt | alu | r1 | 立即数 | 5 |  |  |  |
| ADDIU | PC+4 | PC | rs |  | rt | alu | r1 | 立即数 | 5 |  |  |  |
| ADDU | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 5 |  |  |  |
| AND | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 7 |  |  |  |
| ANDI | PC+4 | PC | rs |  | rt | alu | r1 | 立即数 | 7 |  |  |  |
| SLL | PC+4 | PC |  | rt | rd | alu | r2 | 立即数 | 0 |  |  |  |
| SRA | PC+4 | PC |  | rt | rd | alu | r2 | 立即数 | 1 |  |  |  |
| SRL | PC+4 | PC |  | rt | rd | alu | r2 | 立即数 | 2 |  |  |  |
| SUB | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 6 |  |  |  |
| OR | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 8 |  |  |  |
| ORI | PC+4 | PC | rs |  | rt | alu | r1 | 立即数 | 8 |  |  |  |
| NOR | PC+4 | PC | rs | rt | rd | alu | r1 | r2 | 10 |  |  |  |

在完成指令系统数据通路表的填写之后，根据列出的数据通路表，进行多指令数据通路的合并输入数，表，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。



图 3‑3 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 3.4所示。



图 3‑4单周期CPU数据通路（FPGA）

### 控制器的实现

根据总体方案设计中控制器的设计那一小节的相关内容，分别在Logism和Vivado上进行主控制器、Branch控制器、SYSCALL控制器的具体实现。

主控制器

对照表 3.2所示。

表 3‑2主控制器控制信号

| 指令 | R | RW | WE | X | EXT | Y | ALUop | MemWrite | MemRead | Din | Branch | SYSCALL |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|
| ADD | 00 | 00 | 1 | 0 | 0 | 00 | 0101 | 0 | 0 | 00 | 00 | 0 |
| ADDI | 00 | 10 | 1 | 0 | 0 | 10 | 0101 | 0 | 0 | 00 | 00 | 0 |
| ADDIU | 00 | 10 | 1 | 0 | 0 | 10 | 0101 | 0 | 0 | 00 | 00 | 0 |
| ADDU | 00 | 00 | 1 | 0 | 0 | 00 | 0101 | 0 | 0 | 00 | 00 | 0 |
| AND | 00 | 00 | 1 | 0 | 0 | 00 | 0111 | 0 | 0 | 00 | 00 | 0 |
| ANDI | 00 | 10 | 1 | 0 | 1 | 10 | 0111 | 0 | 0 | 00 | 00 | 0 |
| SLL | 00 | 00 | 1 | 1 | 0 | 01 | 0000 | 0 | 0 | 00 | 00 | 0 |
| SRA | 00 | 00 | 1 | 1 | 0 | 01 | 0001 | 0 | 0 | 00 | 00 | 0 |
| SRL | 00 | 00 | 1 | 1 | 0 | 01 | 0010 | 0 | 0 | 00 | 00 | 0 |
| SUB | 00 | 00 | 1 | 0 | 0 | 00 | 0110 | 0 | 0 | 00 | 00 | 0 |
| OR | 00 | 00 | 1 | 0 | 0 | 00 | 1000 | 0 | 0 | 00 | 00 | 0 |
| ORI | 00 | 10 | 1 | 0 | 1 | 10 | 1000 | 0 | 0 | 00 | 00 | 0 |
| NOR | 00 | 00 | 1 | 0 | 0 | 00 | 1010 | 0 | 0 | 00 | 00 | 0 |
| LW | 00 | 10 | 1 | 0 | 0 | 10 | 0000 | 0 | 1 | 10 | 00 | 0 |
| SW | 00 | 00 | 0 | 0 | 0 | 10 | 0000 | 1 | 0 | 00 | 00 | 0 |
| BEQ | 00 | 00 | 0 | 0 | 0 | 00 | 0000 | 0 | 0 | 00 | 01 | 0 |
| BNE | 00 | 00 | 0 | 0 | 0 | 00 | 0000 | 0 | 0 | 00 | 01 | 0 |
| SLT | 00 | 00 | 1 | 0 | 0 | 00 | 1011 | 0 | 0 | 00 | 00 | 0 |
| SLTI | 00 | 10 | 1 | 0 | 0 | 10 | 1011 | 0 | 0 | 00 | 00 | 0 |
| SLTU | 00 | 00 | 1 | 0 | 0 | 00 | 1100 | 0 | 0 | 00 | 00 | 0 |
| J | 00 | 00 | 0 | 0 | 0 | 00 | 0000 | 0 | 0 | 00 | 10 | 0 |
| JL | 00 | 01 | 1 | 0 | 0 | 00 | 0000 | 0 | 0 | 01 | 10 | 0 |
| JR | 00 | 00 | 0 | 0 | 0 | 00 | 0000 | 0 | 0 | 00 | 11 | 0 |
| SYSCALL | 11 | 00 | 0 | 0 | 0 | 11 | 0000 | 0 | 0 | 00 | 00 | 1 |

部生成。

1. FPGA实现

根据在Logism实现中得到的各个一位控制信号的表达式，直接使用数据流建模，使用assign分的Verilog代码过于冗长，故只取对于控制信号X的生成代码举例如下：

assign X=(~OP[5]&~OP[4]&~OP[3]&~OP[2]&~OP[1]&~OP[0]&~F[5]&~F[4]&~F[3]&~F[2]&~F[1]&~F[0])|(~OP[5]&~OP[4]&~OP[3]&~OP[2]&~OP[1]&~OP[0]&~F[5]&~F[4]&~F[3]&~F[2]&F[1]&F[0])|(~OP[5]&~OP[4]&~OP[3]&~OP[2]&~OP[1]&~OP[0]&~F[5]&~F[4]&~F[3]&~F[2]&F[1]&~F[0]);

以此类推，最终便可以实现整个主控制器中所有控制信号的生成。在Vivado中使用Verilog语言构成的主控制器原理图如图 3.5所示。



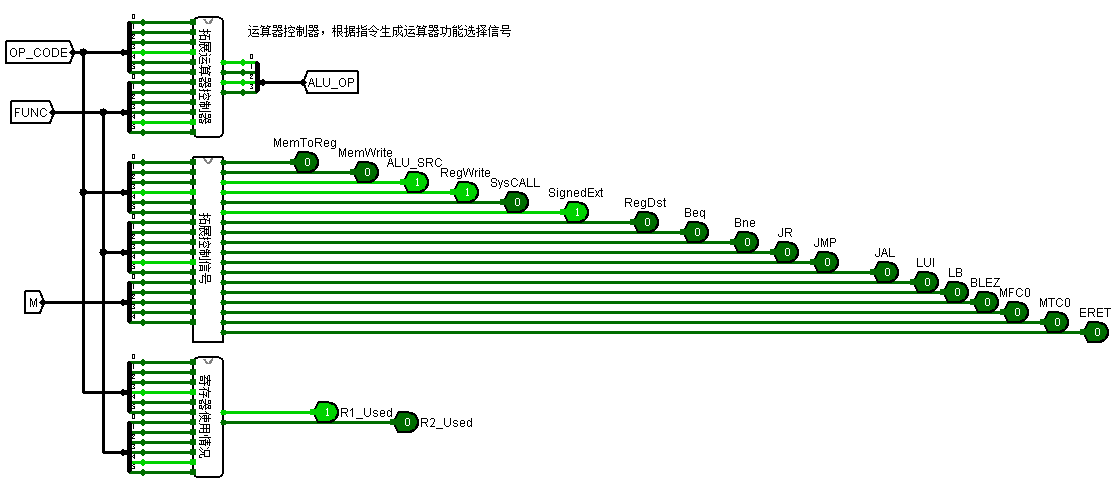
图 3‑5主控制器原理图

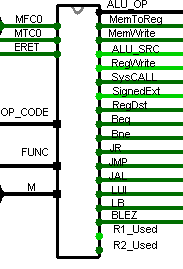
## 多级中断机制实现

### 硬件实现

1. 硬布线控制器的修改

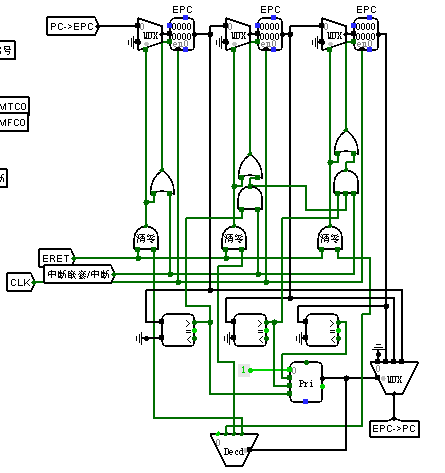
在这里硬布线控制器只是简单的增加了M输入端用来区分MFC0和MTC0两条指令，当然也增加了两个输出端用于两条指令的译码信号。





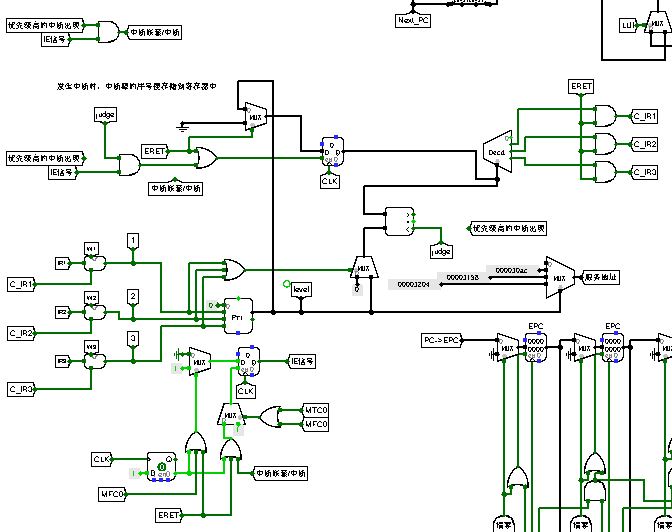
1. 硬件堆栈的实现

在这里基本实现了根据优先级判断的硬件堆栈功能，当前中断的优先级小于新产生中断的优先级时，新产生中断的中断程序入口地址便会存储到“栈”中，当需要返回地址时，优先输出优先级最高的中断返回地址，从优先级上实现了硬件堆栈的功能。



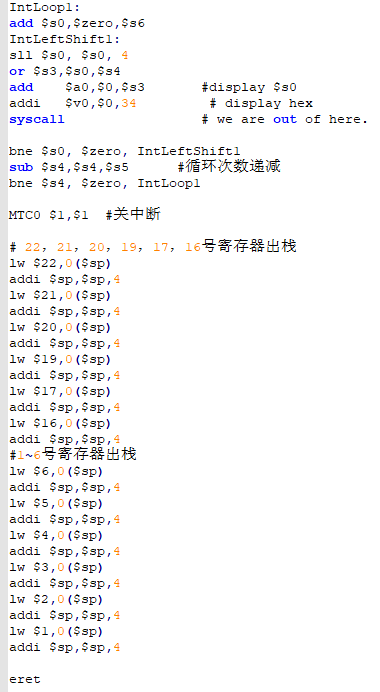
1. 中断优先级判断

实现了中断优先级的判断，根据新产生中断的优先级与当前中断的优先级进行比较，从而确定是否产生中断嵌套信号。根据中断的优先级选取中断程序的入欧地址。



### 软件实现

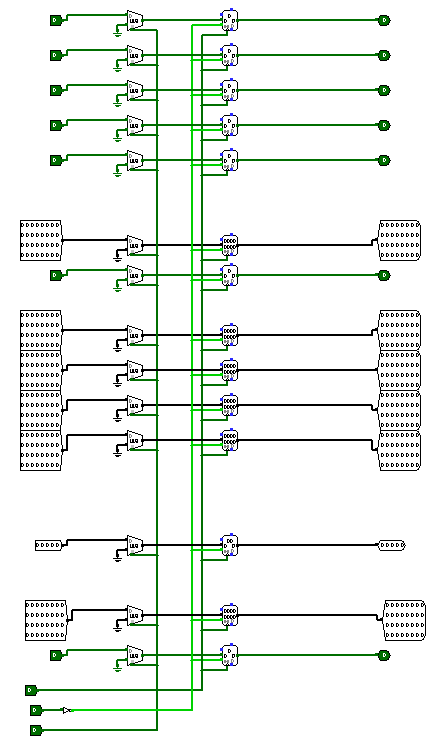
代码中有详细的解释，这里不再赘述。



## 流水CPU实现

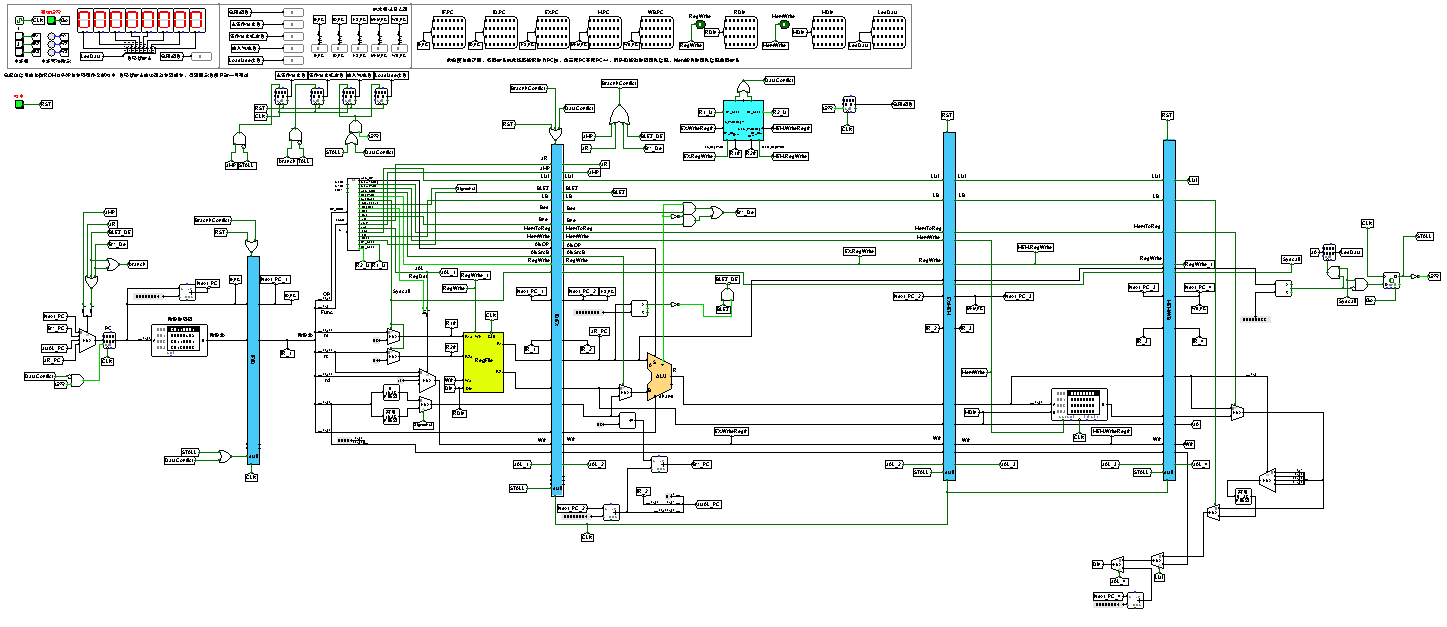
### 流水接口部件实现

我们取样EX/MEM流水接口部件，所有数据的输入、存储和输出均通过寄存器实现，使能端就是所有寄存器的使能端。有个特别注意的地方是，所有的流水接口部件清零必须使用同步清零，异步清零会对实验造成影响。



### 理想流水线实现

理想流水线的数据通路和与前面介绍的单周期的指令的数据通路一样，与单周期CPU指令不同之处在于，使用的数据都经过对应的寄存器组进行缓存，然后再传送到下一个阶段使用。理想流水线的数据通路如



## 气泡式流水线实现

1. 数据冲突判断

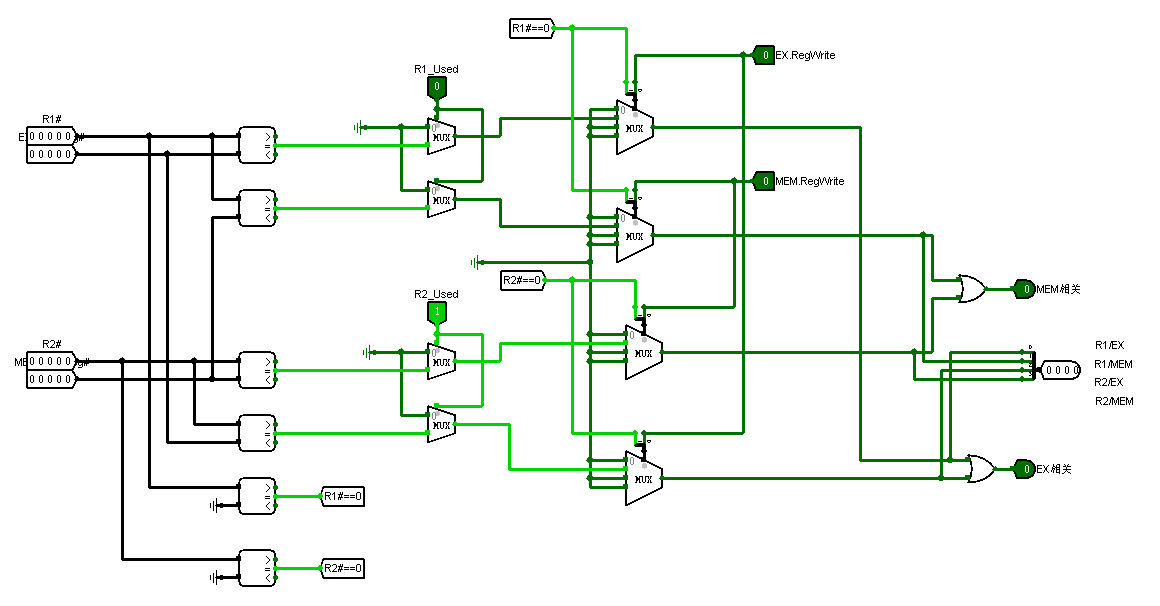
数据冲突的判断逻辑为：

(ID.ReadReg# = = EX.WriteReg#)&&(EX.RegWrite = = 1)

(ID.ReadReg# = = MEM.WriteReg#)&&(MEM.RegWrite = = 1)

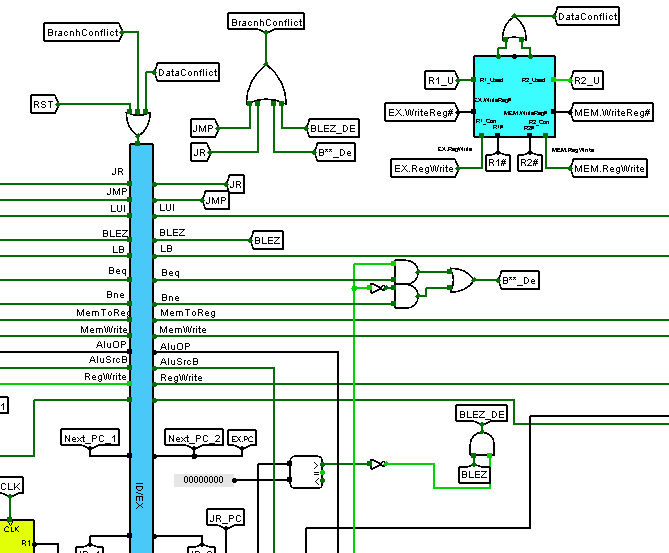
0号寄存器不考虑相关性，R1\_Used和R2\_Used均为两个源寄存器R1、R2是否使用的标识

数据冲突的判断如

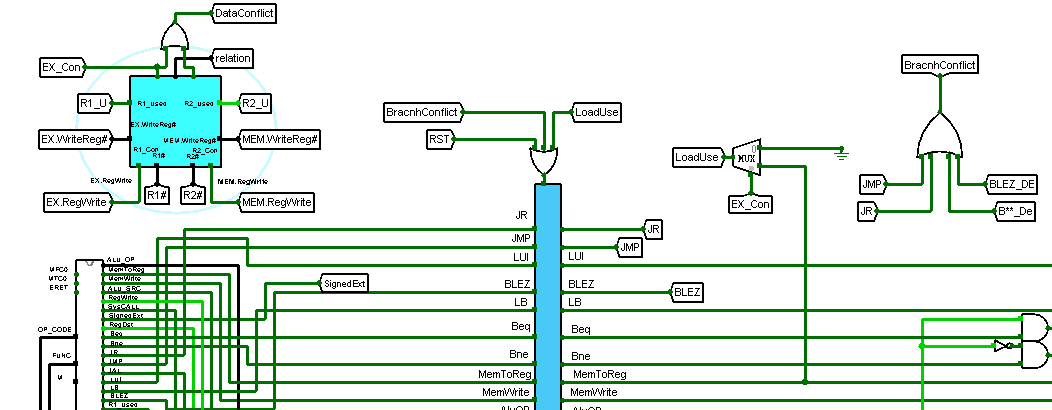


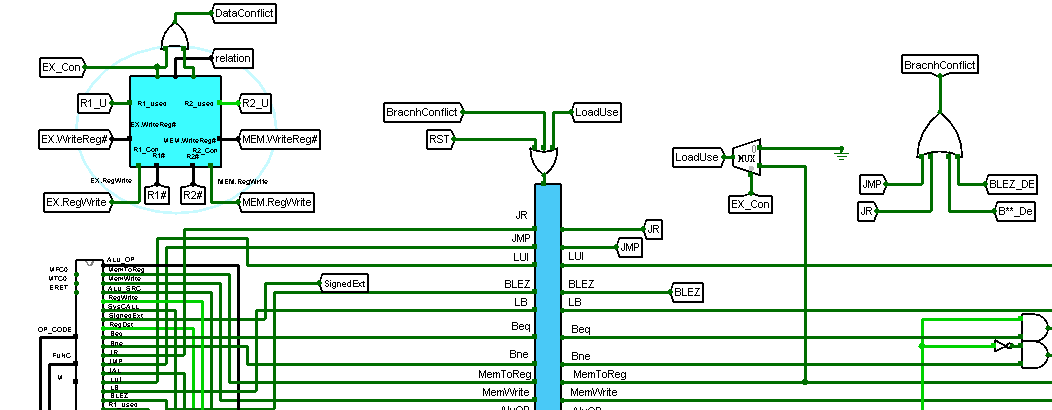
1. 控制冲突判断

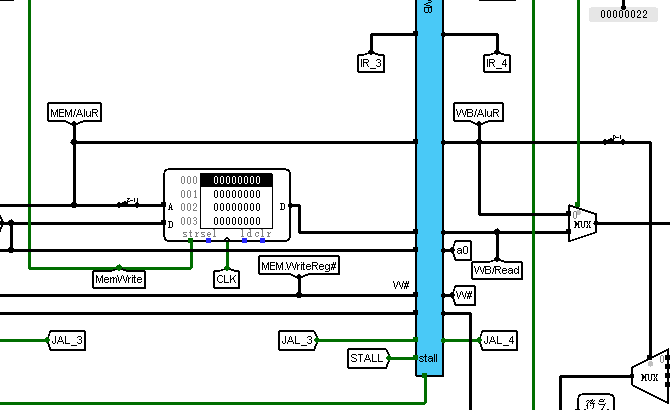
控制冲突判断比较简单,在执行段判断当前指令是否为J指令，或者为B指令且成功跳转。控制冲突判断如



## 数据转发流水线实现

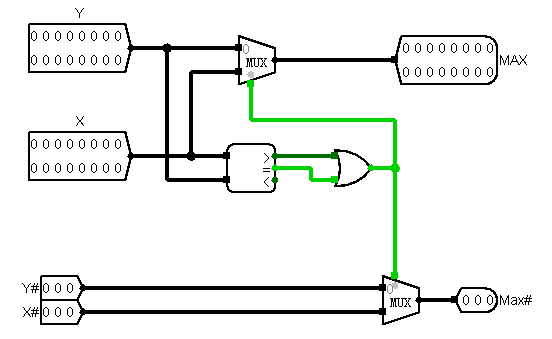




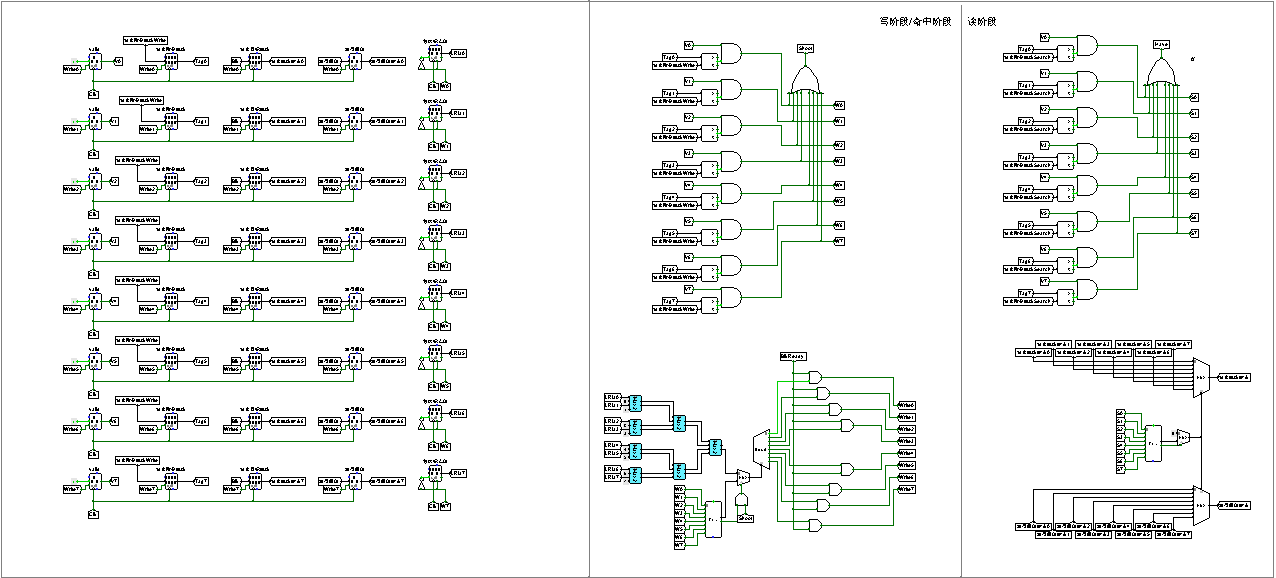


## 动态分支预测机制实现

在BHT表中用到了Max2归并比较器，同于比较两个数值的大小，并且输出数值较大的地址和数值。归并比较器如

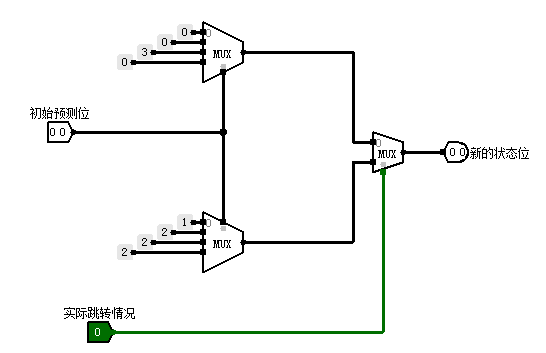


如图所示，8个表项一共8行，每行5个寄存器，分别为有效位、指令地址、目标地址、预测历史位和淘汰标志位，所有的寄存器均为时钟上升沿触发。

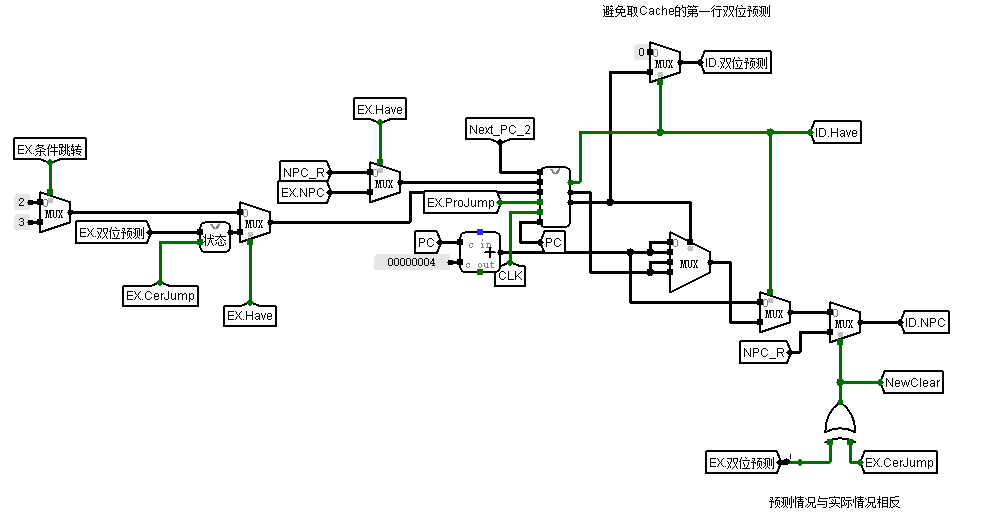


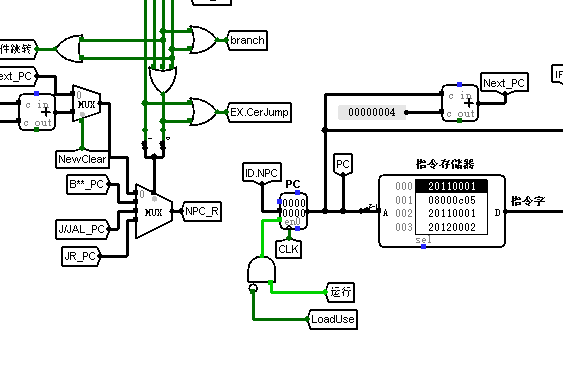
BHT表

在动态分支预测流水线中，对于动态分支的实现还用到了双位预测状态机。状态机根据初始预测位和实际跳转情况来产生新的状态位。双位预测状态机如



地址转移逻辑以及各个隧道的含义如





# 实验过程与调试

## 测试用例和功能测试

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

### 测试用例1

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

### 测试用例1

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

## 可自行安排章节

XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

## 性能分析

**分析不同方案时钟周期数差异**, XXXXX XXXXX……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是……(请自行扩展修改)指令的格式设计是……指令的寻址方式设计是

## 主要故障与调试

### XXX故障

理想流水线： 接口处数据传输问题。

**故障现象：**执行halt指令时控制信号无法通过ID/EX接口。

**原因分析：**如图 4.1，寄存器设置为上升沿刷新，但当D端有输入且clk变化一个周期后寄存器中的数据仍然没有改变，检查时注意到此时ALUControl端输入的值为不确定，这是由于在控制器电路中未给halt指令相应的ALUControl值，为设计控制器时的失误。如图 4.1，寄存器设置为上升沿刷新，但当D端有输入且clk变化一个周期后寄存器中的数据仍然没有改变，检查时注意到此时**ALUControl**端输入的值为不确定，这是由于在控制器电路中未给halt指令相应的ALUControl值，为设计控制器时的失误。

**解决方案：**在控制器中给halt指令（OP为12）一个ALUControl信号0000一边让控制信号在接口处顺利传递。



图 4‑1无法向寄存器中写入数据示意图

### XXX故障

Verilog实现动态分支预测：状态机转移错误。

**故障现象：**如图 4.2所示，当判断信号judge为10b，即判断错误，predict\_f信号为10b，即判断指令不发生跳转，可判断指令最终发生了跳转，状态state[1]应向11b方向转移，即由01b转移至10b，可此时状态并未发生相应转移。



图 4‑2程序故障图

**原因分析：**添加测试信号signal用来判断程序在此时进入了判断状态的case语句中的哪一个状态分支，结果显示，程序竟然进入了状态为00b的分支，再通过仔细分析可以得到，此模块的敏感变量表内为judge、predict\_t两个信号，在编写时采用的是电平触发，可能会产生毛刺，并最终导致状态机状态转移错误。

**解决方案：**修改动态分支预测器的敏感变量表，将此模块修改为时序控制，使用时钟的下降沿进行控制，在上一个时钟上升沿，分支指令进入EX段，并且判断出分支预测结果的正误，随后的时钟下降沿，使用判断结果作为依据进行状态机的状态转移，便可实现状态机状态的正确转移。

### XXX故障

Verilog实现动态分支预测：预测错误之后重新计算地址错误。

**故障现象：**如图 4.3，judge判断信号为10b，表示预测错误，nop信号，即分支跳转信号为0，表示分支指令没有发生跳转，此时情况为分支指令并未发生跳**转而**动态分支预测器预测指令会跳转，此时需要删去误取的指令，并将不跳转的地址重新计算，应为0x0000316c，然而此时计算出的不跳转地址为0x00003158，错误。

**原因分析：**分析地址计算模块发现代码编写错误。当预测器预测指令跳转而实际指令并未跳转时，应计算出此条分支指令的下一条指令的地址，即在此**分支指令**的地址基础上加上4，而之前的代码在计算时，直接在如今处于的IF段的指令的地址上加上4，必然导致程序出错。



图 4‑3程序故障图

**解决方案：**修改地址计算模块**，**当预测器预测指令跳转而实际指令并未跳转时，因为此时那条被判断的分支指令处于EX段，于是应当使用EX段指令的地址作为基础加上4，便可以得到修改错误预测之后的下一条指令的正确地址**。**

## 实验进度

表 ‑ 课程设计进度表

| 时间 | 进度 |
| --- | --- |
| 第一天 | 复习组成原理CPU相关理论知识，阅读课设任务书，阅读MIPS指令手册，并列出CPU各部件的数据通路表，并完成数据通路的基本构建。 |
| 第二天 | 完成单周期CPU的控制信号表，使用Logisim搭建控制器，实现了单周期CPU并且通过了测试。完成部分Logism单周期CPU故障报告。 |
| 第三天 | 完成Logism单周期CPU的故障报告，并且通过了Logism单周期CPU的检查。使用Verilog实现了部分单周期CPU的重要部件，并通过仿真检查。 |
| 第四天 | 继续使用Verilog进行实现单周期CPU的工作，完成了所有部件的编写、控制器的编写，以及所有部件以及控制器的仿真测试，正在进行数据通路的拼接。 |
| 第五天 | 使用Verilog完成单周期CPU数据通路的连接，并且通过仿真测试。使用Verilog完成时钟分频以及七段数码管的代码编写，正在调试。 |
| 第六天 | 完成CPU电路的功能仿真和时序仿真，并成功将生成bit流烧入FPGA板内实现预计功能。 |
| 第七天 | 复习关于指令流水线的知识点，完成理想流水线的verilog代码，正在调试。 |
| 第八天 | 调试成功理想流水线verilog代码，并成功将bit流烧至FPGA板中。完成冒险处理中的数据冲突处理和分支处理代码编写，正在调试。 |
| 第九天 | 完成冒险处理中的数据冲突和分支处理，并成功烧入FPGA板内。完成数据重定向的Verilog代码的编写，正在调试。 |
| 第十天 | 完成数据重定向的Verilog代码并成功烧入FPGA板内。成功实现动态分支预测，预测成功率显著提高，并成功将代码烧入FPGA板内。 |

# 设计总结与心得

## 课设总结

基于对象的存储是为了克服当前基于块的存储存在的诸多难题，在存储接口和结构层次的重要发展。可以根据应用负载选择优化的存储策略。作了如下几点工作：

1. 完成方案总结（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）。
2. 功能总结（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）（自行修订扩充）。
3. 其他需要总结的内容，（自行修订扩充）。

## 课设心得

本次课程设计可以说是迄今为止所有实验以及课程设计中难度最大的一门。两个星期从早到晚的不懈努力以及国庆节假期的辛苦加班才终于完成了整个课程设计的设计任务。现在再来回顾整个课程设计的整个过程，满满的成就感自是不用说，但是其中也有不少的细节值得我去深思与体会。

课程设计刚刚开始的时候，第一个任务是使用Logism设计单周期CPU，该任务和上学期实验课的任务类似，所以整个过程还算比较迅速。然而，在将该电路使用Verilog语言进行描述时，麻烦接踵而至。因为Logism对于电路是完全可视化的，连接过程清晰明了，但是使用Verilog语言进行数据通路时，因为各个部件的接口在定义时起名不是很规范，又因为只能使用各个wire型变量对于关键部件进行连接，而这些变量的名字在定义时同样没有做到一目了然，从而导致了我在完成了数据通路的连接后，使用了大量的时间对于整个电路进行排错和检查，最终还是将所有的接口和连线规范化的起了名字，才终于解决了问题。这也就说明了工程化、规范化对于电路设计或是程序设计的重要性。

紧接着，理想流水线CPU的设计并没有什么难度，但是使用插入气泡、数据重定向技术对于流水线CPU进行冒险处理时，因为这些方法书本上并没有，老师提供的PPT上也只有简单的一些描述，这就要求我不断地在网上搜索相关的知识内容，和小组内的成员进行相关探讨。随后的动态分支预测的设计难度更高，老师并没有提供任何相关的内容，这又使得我不断地去网上搜寻资料文献，阅读全英文的学术论文。从这一切也可以看出团队在一个工作中的重要性，以及自我学习能力的必要性。

然而对于本次课程设计，我还有一些小小的建议和改进。本次课程设计按照分组的机制，但是因为所有人所需要完成的工作相同，分组也只是更便于组内的同学在设计过程中进行互相交流，分组的作用并不是非常明显。分组内的同学们进度完全不同，且有些同学因为对于课程内容的掌握不是很好，逐渐边缘化，分组并不能调动所有同学加入到课程设计任务中。本课程设计的难度很大，但是时间又有一点不够充分，可以适当简化一下任务（并不是降低最终目标），使得在有限的时间内，在完成全部课程设计目标的基础上，适当减少任务中其中重复的繁杂的工作，以提高整个课程设计的效率。

最后在这里也感谢三位老师对于我在本次课程设计中无数问题的耐心解答，也感谢本组所有成员在课程设计中对于我的帮助和建议。我相信组成原理课程设计必将成为我整个大学生涯中一段无比难忘的回忆。

# 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. 谭志虎，秦磊华，胡迪青.计算机组成原理实践教程.北京：清华大学出版社，2018.
5. 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.
6. 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:** |