华中科技大学

2020

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：	CS1707
学　　号：	U201714781
姓　　名：	罗子成
电　　话：	18860800554
邮　　件：	2567213444@qq.com

目 录

1 课程设计概述

1.1 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。

1.2 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。

1.3 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。

1.4 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	

#	指令助记符	简单功能描述	备注
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 $v0==10 halt(停机指令)
24	SYSCALL	系统调用	else 数码管显示$a0 值
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	SUBU	无符号减	运算指令
29	SLTIU	小于立即数置 1(无符号)	运算指令
30	LHU	加载半字(无符号)	存储访问指令
31	BLEZ	小于等于 0 转移	跳转指令

2 总体方案设计

2.1 单周期 CPU 设计

对于单周期 CPU 的设计这里采用的是硬布线的方法。通过解析指令，产生对应的控制信号，在一个周期内完成该指令需要完成的动作。通过 PC 寄存器给出正确的指令地址，从指令存储器中取出数据送到控制器，控制器解析该指令并给出对应的控制信号，这些控制信号给出各处多路选择器的选择信号、寄存器组的写使能信号、内存的写使能信号以及立即数的扩展方式，从而完成指令的正确执行。

总体结构图如图 2.1 所示。

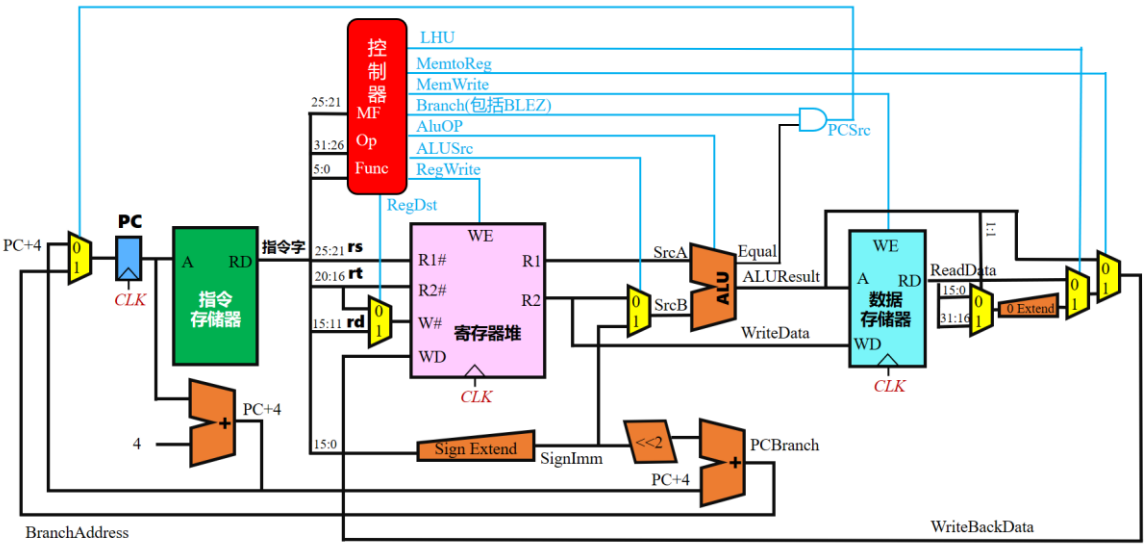

图 2.1 单周期 CPU 总体结构图

2.1.1 主要功能部件

1. 程序计数器 PC

程序计数器 PC 的值决定了整个程序的走向，必须保证能够提供正确的地址，因此是程序正确执行的过程中最重要的一环。当然，由于程序的结构复杂性，PC 是不能够简单的进行加 4 处理的。因此，这里采用的是寄存器，通过在寄存器的加载端给

出正确的下一条指令的地址以实现 PC 的正确更新。

正确的下一条指令的地址则由多路选择器给出，若当前指令不是跳转指令或跳转失败则选择 PC+4 作为下一条指令的地址，若当前指令跳转成功则选择跳转后的地址。

2. 指令存储器 IM

指令存储器采用 ROM 实现，将镜像文件加载进 ROM 之后则可以根据 PC 寄存器给出的指令地址取出对应的指令向后传递。

3. 运算器

此处采用的运算器支持 13 种不同的运算，具体功能描述如表 2.1 所示。引脚位宽以及各引脚功能如表 2.2 所示。

表 2.1 ALU 规格

ALUOP	十进制值	运算功能
0000	0	Result=X<<Y 逻辑左移 (Y 取低五位) Result2=0
0001	1	Result=X>>Y 算术右移 (Y 取低五位) Result2=0
0010	2	Result=X>>Y 逻辑右移 (Y 取低五位) Result2=0
0011	3	Result=$(X*Y)_{[31:0]}$; Result2=$(X*Y)_{[63:32]}$无符号乘法
0100	4	Result=X/Y; Result2=X%Y 无符号除法
0101	5	Result = X + Y (Set OF/UOF)
0110	6	Result = X – Y (Set OF/UOF)
0111	7	Result = X & Y 按位与
1000	8	Result = X | Y 按位或
1001	9	Result = X⊕Y 按位异或
1010	10	Result = ~(X | Y) 按位或非
1011	11	Result = (X < Y) ? 1 : 0 符号比较
1100	12	Result = (X < Y) ? 1 : 0 无符号比较

表 2.2 算术逻辑运算单元引脚与功能描述

引脚	输入/输出	位宽	功能描述
X	输入	32	操作数 X
Y	输入	32	操作数 Y
ALU_OP	输入	4	运算器功能码，具体功能见表 2.1 ALU 规格
Result	输出	32	ALU 运算结果
Result2	输出	32	ALU 结果第二部分，用于乘法指令结果高位或除法指令的余数位，其他操作为零
OF	输出	1	有符号加减溢出标记，其他操作为零
UOF	输出	1	无符号加减溢出标记，其他操作为零
Equal	输出	1	Equal=(x==y)?1:0, 对所有操作有效

4. 寄存器堆 RF

采用 CS3410.jar 提供的 Register File 元件，实现 32 个通用寄存器。各引脚位宽以及功能描述如表 2.3 所示。

表 2.3 寄存器组引脚功能描述

引脚	输入/输出	位宽	功能描述
R1#	输入	5	读寄存器编号
R2#	输入	5	读寄存器编号
W#	输入	5	写寄存器编号
WE	输入	1	寄存器组写使能
Din	输入	32	待写入的数据
CLK	输入	1	时钟信号
R1	输出	32	R1#寄存器中的数据
R2	输出	32	R2#寄存器中的数据

2.1.2 数据通路的设计

对于不同的指令，数据通路中的数据来源不尽相同。将指令字的 21-25 位约定为 rs，16-20 位约定为 rt，11-15 位约定为 rd，0-15 位约定为 imm，0-25 位约定为 j_pc，

ALU 的运算结果约定为 alu，从 ram 中读取的数据约定为 data，则各条指令的数据通路如表 2.4 所示。

表 2.4 指令系统数据通路框架

指令	PC	IM	RF				ALU		DM	
			R1#	R2#	W#	RDin	A	B	Addr	MDin
R 型指令	PC+4		rs	rt	rd	alu	r1	r2		
syscall	PC+4		2#	4#			r1	r2		
Store 指令	PC+4	imm	rs	rt			r1	imm	alu	r2
Load 指令	PC+4	imm	rs	rt	rt	Data	r1	imm	alu	r2
I 型指令	PC+4 || 分支成功时：PC+4+(imm<<2)	imm	rs	rt	rt	Alu	r1	imm		
J 型指令	$[PC+4]_{31:28}\|\|j\_pc\|\|0^2$									

2.1.3 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测试大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 2.5 所示。

表 2.5 主控制器控制信号的作用说明

控制信号	取值	说明
AluOP	0-12	ALU 功能选择信号
MemToReg	0	寄存器写数据来源是 ALU
	1	寄存器写数据来源是 Mem
MemWrite	0	Mem 不可些
	1	Mem 可写
AluSrcB	0	ALU 输入端口 B 的取值是 r2
	1	ALU 输入端口 B 的取值是 imm
RegWrite	0	寄存器组不可写
	1	寄存器组可写

控制信号	取值	说明
Syacall	0/1	当前指令是否为 syscall
Signedext	0	imm 使用 0 扩展
	1	imm 使用符号扩展
RegDst	0	写寄存器编号为 rt
	1	写寄存器编号为 rd
BEQ	0/1	当前指令是否为 BEQ
BNE	0/1	当前指令是否为 BNE
JR	0/1	当前指令是否为 JR
JMP	0/1	当前指令是否为直接跳转指令
JAL	0/1	当前指令是否为 JAL
MFC0	0/1	当前指令是否为 MFC0
MTC0	0/1	当前指令是否为 MTC0
ERET	0/1	当前指令是否为 ERET
LHU	0/1	当前指令是否为 LHU
BLEZ	0/1	当前指令是否为 BLEZ
R1_used	0/1	当前指令是否使用 r1 所在寄存器
R2_used	0/1	当前指令是否使用 r2 所在寄存器

对照所有控制信号，对 31 条指令逐个分析，确定该指令执行过程中需要哪些控制信号，并将该信号置 1。该控制信号表的框架如表 2.6 所示。对于只有对应指令出现才会为 1 的信号则不在表格列出。

表 2.6 主控制器控制信号框架

指令	ALU_OP	Memto Reg	Mem Write	ALU_SRCB	Reg Write	Signed Ext	Reg Dst	JMP	R1_used	R2_used
SLL	0				1		1			1
SRA	1				1		1			1
SRL	2				1		1			1
ADD	5				1		1		1	1
ADDU	5				1		1		1	1

指令	ALU_OP	MemtoReg	MemWrite	ALU_SRCB	RegWrite	SignedExt	RegDst	JMP	R1_used	R2_used
SUB	6				1		1		1	1
AND	7				1		1		1	1
OR	8				1		1		1	1
NOR	10				1		1		1	1
SLT	11				1		1		1	1
SLTU	12				1		1		1	1
JR								1	1	
SYSCALL									1	1
J								1		
JAL					1			1		
BEQ						1			1	1
BNE						1			1	1
ADDI	5			1	1	1			1	
ANDI	7			1	1				1	
ADDIU	5			1	1	1			1	
SLTI	11			1	1	1			1	
ORI	8			1	1				1	
LW	5	1		1	1	1			1	
SW	5		1	1		1			1	1
MFC0										
MTC0										1
ERET										
SUBU	6				1		1		1	1
SLTIU	11			1	1	1			1	
LHU	5	1		1	1	1			1	
BLEZ						1			1	

2.2 中断机制设计

2.2.1 总体设计

对于有三个中断源的多级中断，首先要绘制三个中断产生电路，这可以由中断按键参考电路修改完成。由于低级中断的执行过程可以被优先级更高的中断打断，则需要通过设置中断屏蔽字 INM 实现，当中断请求产生时，该信号首先要和 INM 对应位做与操作，用以判断该请求中断是否被允许执行，如果 INM 允许其中断执行，还需要通过该请求信号与中断使能信号 IE 相与，用以判断 CPU 当前能否被中断，若 INM 和 IE 都允许该中断发生，则通过 MFC0 将 EPC 和 INM 先送到寄存器组，并通过 SW 压如内存栈，当保护现场操作完成后通过 MTC0 设置新的 INM 并将 IE 置 1，即打开中断。中断程序执行完毕后，先通过 MTC0 将 IE 置 0 关中断，将寄存器恢复，使用 MTC0 指令将 INM 和 EPC 也恢复之后，通过 ERET 指令结束中断，ERET 信号同时将 IE 置 1，使得后续中断可以正常请求。

对于流水多级中断，采用 WB 段检测的方式，当 WB 端检测到中断产生信号时，等待 MEM 段的指令不为 nop 后给出中断确认产生信号，清空流水，执行中断程序。MTC0 指令在 EX 端执行，MFC0 和 ERET 指令在 WB 段执行。

2.2.2 硬件设计

1. 中断请求电路

使用三份中断按键参考电路，分别产生 IR1#，IR2#，IR3#三个中断请求信号。

2. 中断产生电路

首先，产生的中断请求信号需要和 INM 对应位做与操作。由于只有三个中断源，可以使用三个 D 触发器作为 INM 的低三位。中断请求信号和 INM 对应位的与结果作为第一次筛选出来的中断请求信号。为了满足优先级 IR3＞IR2＞IR1 的要求，使用优先编码器将筛选过的三个中断请求信号输入，同时输出优先级最高的中断请求信号。该信号需要再和 IE 寄存器相与，最后产生的信号若为 1 则中断产生一个中断产生信号，通过该信号和中断编号查询中断向量表获取对应的中断入口地址，该中断产生信号可同时作为 PC 多路选择器的一个选择信号。在下一个时钟到来的时候，EPC 的内

容会更新并抹去中断产生信号。

2.2.3 软件设计

对于 MFC0、MTC0、ERET 三条中断相关的扩展指令，我对他们进行了一些改造。由于电路中不存在 CP0 协处理器的寄存器组，为了简化设计，将 INM、EPC、IE 分别视为 CP0 中寄存器组中的三个寄存器，编号为 3、2、1。

1. MFC0 指令

MFC0 指令的用法为 MFC0 rt，rd，其功能是将 CP0 中寄存器组中 rd 的内容送到通用寄存器组中的 rt 里面。实际上，由于只需要将 INM 和 EPC 的内容送到内容保护，且约定的编号分别为 3，2，因此只需要通过在 MFC0 信号产生的时候根据 rd 的最低位即可判断需要送入通用寄存器组的数据是 INM 还是 EPC。例如 MFC0 $a0, $3 即代表将 INM 的内容送入通用寄存器组的 4 号寄存器。

2. MTC0 指令

MTC0 和 MFC0 指令类似，用法为 MTC0 rt，rd，只不过功能恰好相反，是将通用寄存器组 rt 中的数据传递到 CP0 中寄存器组中的 rd 里面。由于可以通过指令控制 IE、EPC、INM 三个不同的数据，因此在 MTC0 信号产生的同时需要根据 rd 编号的低两位通过解复用器给出对应的写使能信号即可实现 IE、EPC、INM 的正确传输。

3. ERET 指令

ERET 指令的设计相对来说比较简单，只需要根据 ERET 信号根据中断编号给出给出对应的中断结束信号以及将 IE 置 1 即可。

2.3 流水 CPU 设计

2.3.1 总体设计

此处采用的流水线结构为经典的五段流水，分别为取值(IF)、译码(ID)、执行(EX)、访存(MEM)、写回(WB)。通过流水接口将上一段的 PC、IR 和需要的控制信号及数据流入下一段以实现流水操作。

2.3.2 流水接口部件设计

对于五段流水线，需要设计四个流水接口用于传递 PC、IR 以及相应的控制信号。这里将四个流水接口分别命名为 IF/ID、ID/EX、EX/MEM、MEM/WB。

1. IF/ID 流水接口

除了 PC 寄存器的值需要向后传递以外，IR 的值以及分支预测相关的预测结果、命中行号、是否命中的值向后传递。因此使 2 组 32 位的寄存器分别将 PC 和 IR 的值向后传递，将分支预测的结果通过分线器组合成 6 位宽的数据向后传递(由于 BHT 表采用 8 行全相连 cache，需要三位行号，两位预测结果以及一位命中结果)。

2. ID/EX 流水接口

ID 段获取了 IF 段传递过来的 IR 后对 IR 进行解析，控制器产生的控制信号都需要向后传递。因此将控制信号分为四组，分别为 WB 段、MEM 段、EX 段需要使用的控制信号以及中断相关信号(MF)。在 ID 段访问寄存器组之后需要将取出来的数据向后传递，对于 I 型指令，经过位扩展单元后的数据也要向后传递。由于数据相关的存在，在 ID 段需要进行数据相关检测，在重定向流水线中还需要将冲突检测的结果向后传递。同时，IF 段传递过来的 PC、IR 以及分支预测相关数据还需要向后传递。

3. EX/MEM 流水接口

EX 段会给出计算的结果，因此计算结果 ALU 需要向后传递。对于 syscall 指令，为了确保指令能够执行完整，它需要在 WB 段再进行判断，因此 A 和 B 的值还要继续向后传递。同时，PC、IR、MEM 段和 WB 段的控制信号以及中断相关的信号还需要还需要向后传递。

4. MEM/WB 流水接口

访存指令会在 MEM 段给出数据结果，因此需要将访存结果 Mdata 向后传递。同时，PC、IR、WB 段的控制信号、中断相关信号、A 和 B 的值、ALU 运算结果还需要向后传递。

2.3.3 理想流水线设计

理想流水线不存在冲突，因此只需要保证数据传递正确即可。寄存器组在 ID 段只可访问，WB 才可写，因此 W#和 Din 都应该由 WB 段给出。由于 syscall 指令的判断在 EX 段便可以完成，但在 EX 直接停机会造成 MEM 段、WB 段的指令并没有执行完整，所以 syscall 的判断应该修改到 WB 段，使用流水接口传递过来的 v0 和 a0 的值。

2.4 气泡式流水线设计

在理想流水线的基础上，气泡流水线需要解决的问题是数据相关以及分支相关。

1. 分支相关处理

由于分支的结果可以在 EX 段通过 alu 给出，这里设计的气泡流水线是在 EX 段执行分支指令。EX 段可以使用 PC 和 IR 计算出直接跳转指令的目标地址 J_PC，使用 ID 段传递的 imm 和 PC 计算出分支跳转指令的目标地址 B_PC，同时根据 EX 段给出的跳转信号更新 PC 寄存器的值，这时，IF 和 ID 段的指令为误取指令，需要清空，因此会产生两个气泡。

2. 数据相关处理

ID 段和 WB 段的数据相关冲突可以通过更改寄存器组的时钟为下降沿解决，对于 ID 段和 EX 段或 ID 段和 MEM 段的数据冲突则需要加入气泡解决。当 ID 段需要使用的源寄存器是 MEM 段或者 EX 段的目标寄存器的时候，流水线会发生写后读冲突，ID 段获取的数据就不是正确的值，因此需要将 IF/ID 接口锁住，并在 ID/EX 接口插入气泡，直到 ID 段和 EX、MEM 段不存在数据相关。

2.5 数据转发流水线设计

在气泡流水线的基础上，为了解决数据相关带来的过多气泡，可以在数据相关产生的时候判断能否通过重定向的方式解决，能够重定向解决的则不需要通过气泡带来额外的开销。这时候，在 ID 段检测出的数据相关可以分为 load-use 相关和其他相关，即 ID 段的某一源寄存器是 EX 段的目标寄存器，且 EX 段是一条 load 指令则发生

load-use 相关，此时必须通过在 ID/EX 接口插入一个气泡的方式解决，否则流水线中的关键时延将会变长，影响整体效率。而对于其他相关，则可以通过具体的种类再下一个时钟周期将 MEM 段的 alu 结果、WB 段的 alu 结果或者 WB 段的访存结果重定向到 A 或 B 的位置解决数据相关。

2.6 动态分支预测机制

在重定向流水线的基础上，为了尽最大限度减少分支指令带来的气泡个数，可以在 IF 段添加 BHT 表实现动态分支预测。

1. BHT 表设计

BHT 表可以看做一个 cache，BHT 表每行包括有效位(1 位)、PC 值(32 位)、预测结果(2 位)、计数值(32 位)以及分支目标地址(32 位)，这里设计了 8 行全相连 cache，PC 作为关键字，当 PC 相等且有效位为 1 时视为命中，根据预测结果给出目标地址(预测失败是 PC+4，预测成功则给出目标地址)。当 BHT 表满的时候，使用 LRU 替换算法替换掉计数器值最大的记录。

2. 数据通路设计

IF 段将 PC 同时传入 BHT 表和指令存储器，BHT 给出下一条指令的地址，同时将预测结果、命中结果以及命中的行号组合成一个 6 位的数据。在 EX 段根据预测结果和实际结果更新预测值，并将记录更新。若预测结果不符程序进行，则需要将 IF、ID 段的指令清空，给出正确的地址以确保程序的正确执行。

3 详细设计与实现

3.1 单周期 CPU 实现

3.1.1 主要功能部件实现

1) 程序计数器（PC）

使用一个 32 位寄存器实现程序计数器 PC，触发方式为上升沿，输入为下一条将要执行的指令的地址，输出为当前执行指令的地址。Run 是运行使能信号，接在集群器的使能端，当 run 为低电平时忽略时钟输入。具体实现如图 3.1 所示。

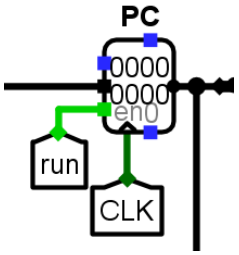

图 3.1 程序计数器 PC

2) 指令存储器（IM）

使用一个只读存储器 ROM 实现指令存储器（IM）。设置该 ROM 的地址位宽为 10 位，数据位宽为 32 位。因此 PC 给出的字节地址应通过选取 2-11 位转换成字地址。具体实现如图 3.2 所示。

图 3.2 指令存储器 IM

3.1.2 数据通路的实现

将 24+3+4 条指令对应到表 2.4 指令系统数据通路框架中的指令类型，根据不同类型指令所使用的不同来源使用多路选择器根据不同的选择信号选择对应的输入，并

将主要功能部件连接，构建正确的数据通路。如图 3.3 所示。

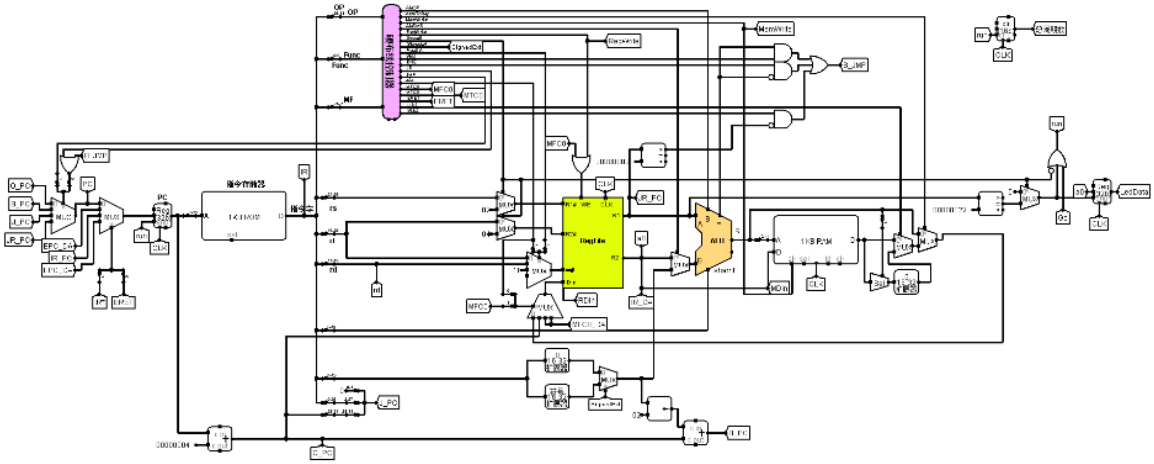

图 3.3 单周期 MIPS 数据通路

3.1.3 控制器的实现

根据表 2.6 主控制器控制信号框架通过 logisim 自动生成电路的功能填入表达式分别生成运算控制信号电路、控制信号电路以及寄存器使用情况电路。再将指令字 IR 的对应位分别传入三个电路生成所有的控制信号。如图 3.4 所示。

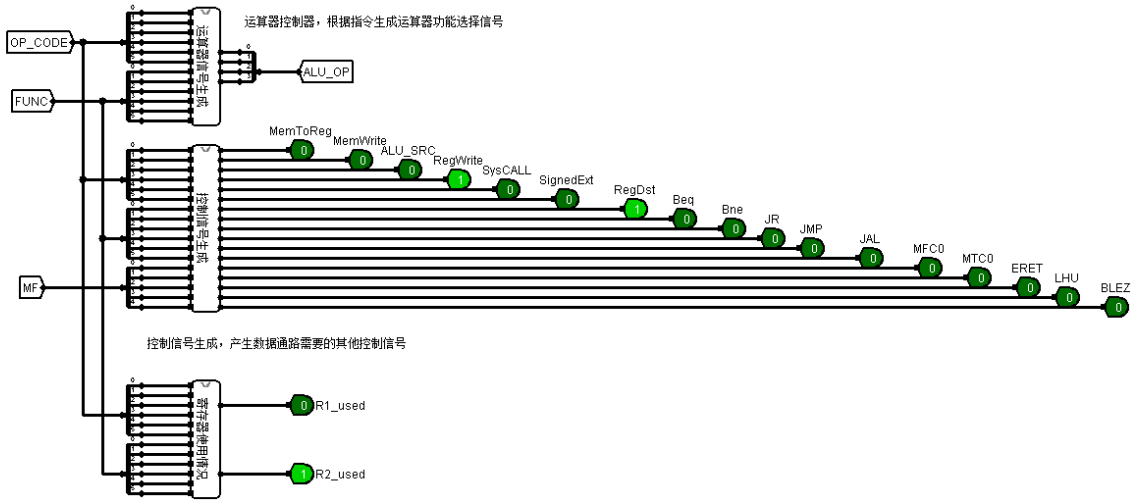

图 3.4 控制器实现

3.2 中断机制实现

3.2.1 中断请求电路的实现

根据中断按键参考电路绘制中断请求电路，其中 IR_作为输入源信号，W_作为等

待信号，IR_#作为中断请求信号，IRx_over 信号作为中断清零信号。如图 3.5 所示。

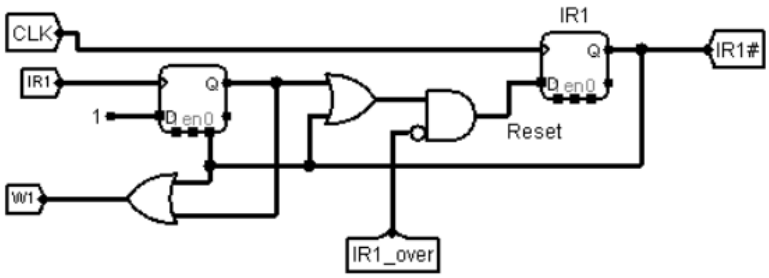

图 3.5 中断请求电路

3.2.2 中断相关指令实现

在主控制器里已经给出了 MTC0 信号、MFC0 信号以及 ERET 信号。但对于 MTC0 和 MFC0 而言，需要控制不同的数据，因此还需要对信号进行扩展。

根据约定，对 CP0 的 1、2、3 号寄存器执行 MTC0 操作实际上是更改 IE、EPC 或者 INM 的值，因此需要通过解复用器将 MTC0 信号实例化成对应寄存器的写使能信号，如图 3.6 所示。

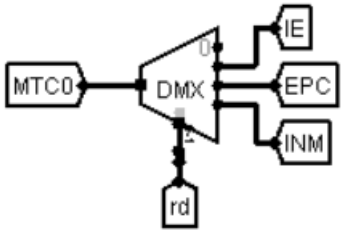

图 3.6 MTC0 指令实现

对于 MFC0 指令而言，2、3 号寄存器分别代表将 EPC 或 INM 的数据传递到通用寄存器组，因此可以根据 rd 编号的最低位选择传递的数据，如图 3.7 所示。

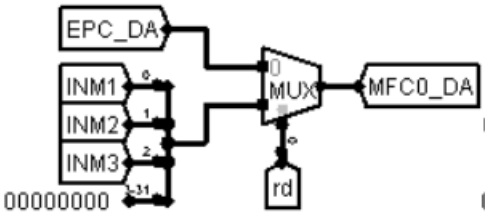

图 3.7 MFC0 指令实现

3.2.3 中断屏蔽字 INM 的实现

由于这里只有三个中断源，则只需要使用三个 D 触发器作为 INM 的低三位，分别对应三个中断源的中断屏蔽信号。MTC0 指令实例化的 INM 使能信号接入 D 触发器的使能端，加载的数据为 rt 寄存器中的数据。具体实现如图 3.8 所示。

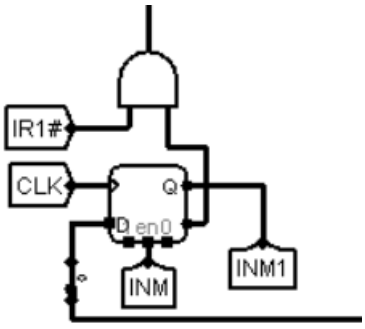

图 3.8 中断屏蔽寄存器 INM

3.2.4 中断使能寄存器 IE 和 EPC 寄存器的实现

中断使能信号只有一位，可以使用一个 D 触发器实现。中断使能信号是中断请求信号需要经过的最后一道关卡，优先编码器传递的信号和中断使能信号的与结果即是中断产生信号 IR*。这个信号用于控制 PC 的输入选择，因此它只能存在一个时钟周期。故将 IR*信号接入 IE 寄存器的使能端，加载端在 IR*的情况下为 0。MTC0 实例化后的 IE 使能信号也要接入使能端，同时，加载端的数据应是 rt 寄存器的数据。当 ERET 信号产生时，需要将 IE 置 1。具体实现如图 3.9 所示。

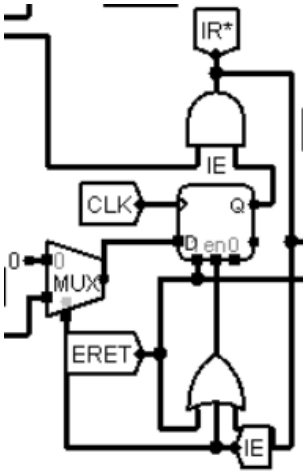

图 3.9 中断使能寄存器 IE

EPC 寄存器的实现较为简单，使用 32 位寄存器实现。当 IR*信号或 MTC0 实例化出的 EPC 信号出现时需要更改 EPC 的内容。因此将这两个信号接入 EPC 寄存器的使能端。IR*对应的加载端数据为 PC，EPC 使能信号对应的加载端的数据为 rt 的数据。具体实现如图 3.10 所示。

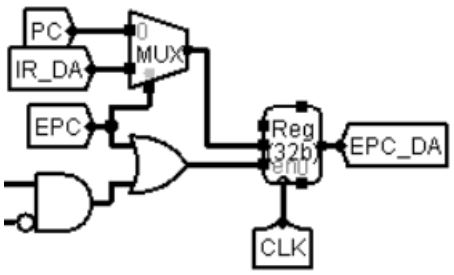

图 3.10 EPC 实现

3.2.5 中断清零信号和中断向量表的实现

当 ERET 信号产生时需要同步产生对应的中断清零信号以结束一个中断程序。因此，通过解码器将优先编码器给出的中断源编号与 ERET 信号的与结果给出中断清零信号。如图 3.11 所示。

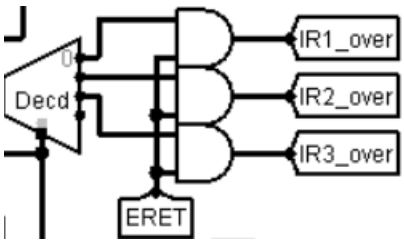

图 3.11 中断清零信号

优先编码器给出的中断源编号同时可作为中断向量表的选择信号，通过 MARS 编译中断程序给出对应指令地址即可完成。如图 3.12 所示。

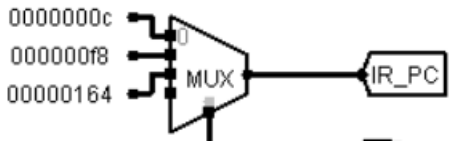

图 3.12 中断向量表

3.2.6 中断产生电路的实现

根据中断机制设计中的中断产生电路逻辑，加入一个优先编码器，绘制出中断产

生电路，如图 3.13 所示。

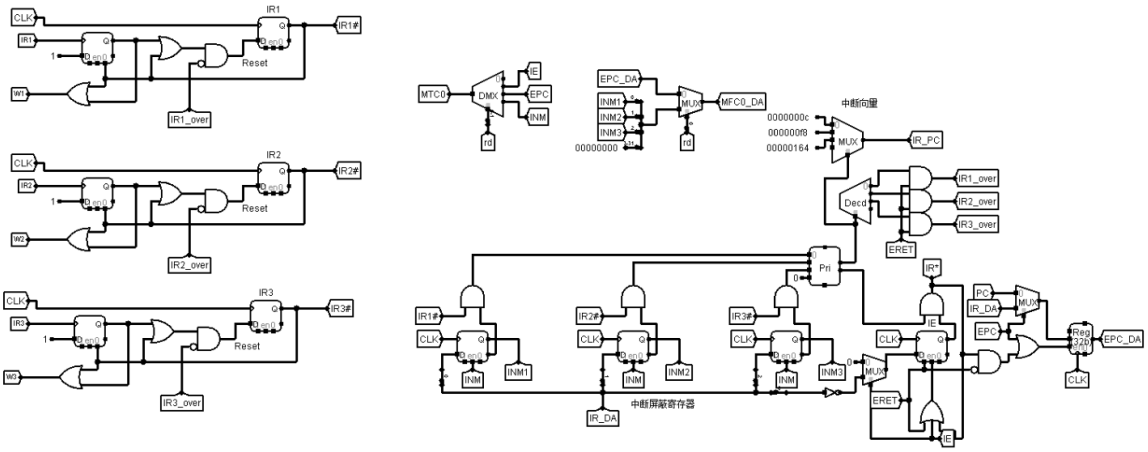

图 3.13 中断产生电路

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

采用寄存器作为流水接口中的存储单元，同步清零信号选择寄存器的加载端是 0 或者前一段的数据，使能信号 stall 控制寄存器是否更新自身数据。每个流水接口中的寄存器及数据位宽由该流水接口两侧需要传递的数据个数及位宽决定。具体实现如图 3.14 所示。

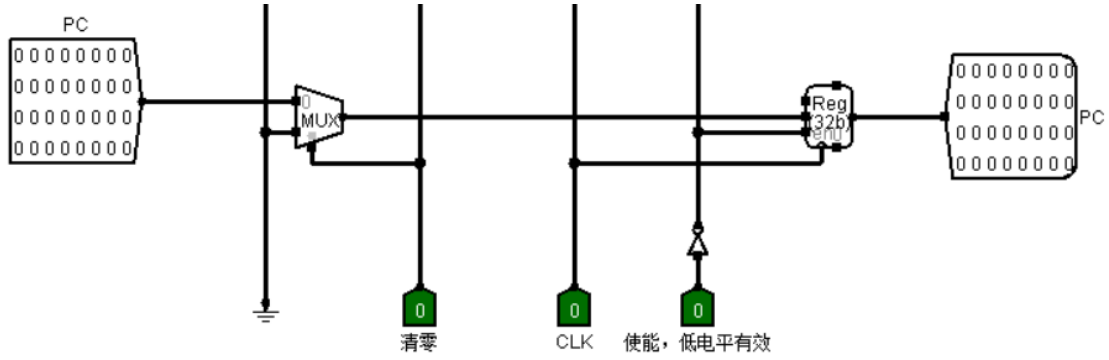

图 3.14 流水接口

3.3.2 理想流水线实现

在单周期 MIPS 的基础上，在指令存储器和寄存器组之间插入 IF/ID 接口，在寄存器组和 ALU 之间插入 ID/EX 接口，在 ALU 和 MEM 之间插入 EX/MEM 接口，在 MEM 之后放入 MEM/WB 接口将数据通路分为五部分，分别为 IF、ID、EX、MEM 和 WB。syscall 的判断在 WB 段执行，寄存器组的写寄存器编号以及待写数据由 WB

段提供。具体实现如图 3.15 所示。

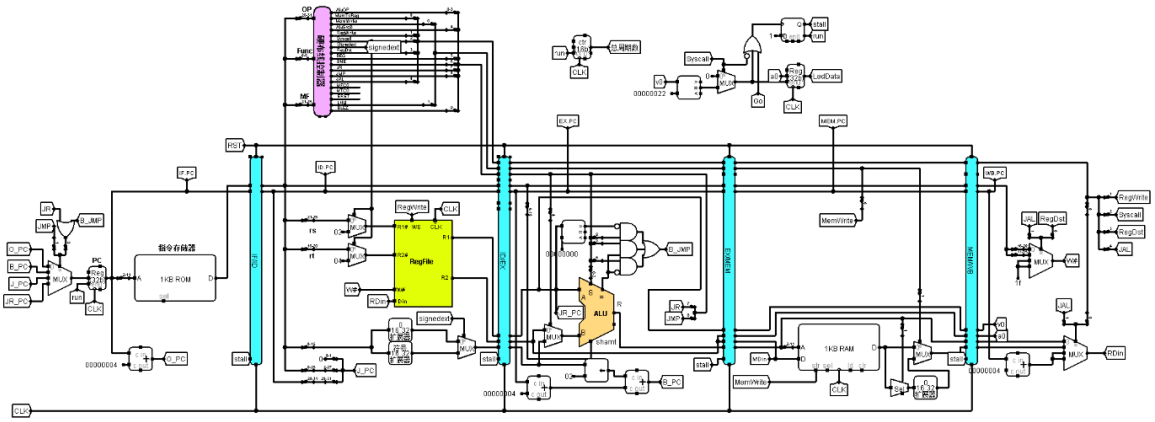

图 3.15 理想流水

3.4 气泡式流水线实现

3.4.1 数据相关检测实现

在 ID 段需要对指令进行数据相关检测，对于四种需要检测的数据相关，其发生条件如表 3.1 所示。

表 3.1 数据相关产生条件

类型	R1_used	R2_uesd	R1=EX.W#	R2=EX.W#	R1=MEM.W#	R2=MEM.W#	EX.RegWrite	MEM.RegWrite
A.EX.hazard	1		1				1	
A.MEM.hazard	1				1			1
B.EX.hazard		1		1			1	
B.MEM.hazard		1				1		1

由此绘制数据相关检测电路如图 3.16 所示。

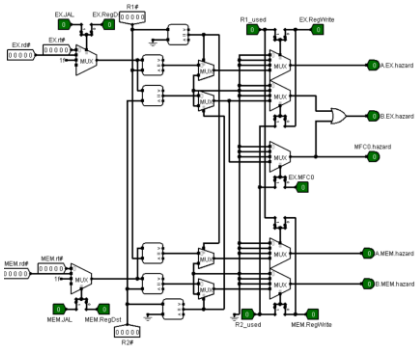

图 3.16 数据相关检测电路

3.4.2 气泡流水的实现

在理想流水的基础上，当 ID 段检测到数据相关时，需要对 ID/EX 接口清零并将 IF/ID 接口锁存。当 EX 段是分支指令且分支成功时，即 B_JMP 或 JMP 信号为高电平，需要对 IF/ID 接口、ID/EX 接口执行清空操作，并在 PC 寄存器的加载端加载分支目标地址。具体实现如图 3.17 所示。

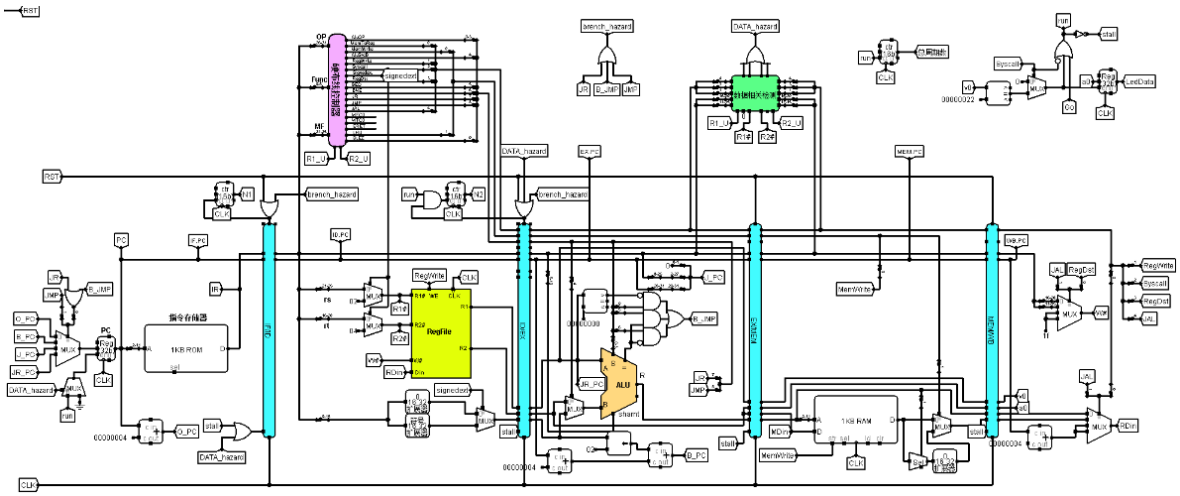

图 3.17 气泡流水线

3.5 数据转发流水线实现

3.5.1 数据重定向逻辑实现

对于数据相关冲突，可以通过数据重定向的方式减少气泡的插入。而当 ID 段与 EX 段发生数据相关冲突且 EX 段的 memtoreg 信号为高电平时会发生数据相关冲突，必须插入一个气泡。其他情况下则可以通过重定向的方式使得可以在 EX 段获取正确的数据而不用等待数据写会。因此，数据的重定向来源如表 3.2 所示。

表 3.2 重定向数据来源

	EX.hazard		MEM.hazard	
	MEM.Memtoreg=0	MEM.Memtoreg=1	WB.Memtoreg=0	WB.Memtoreg=1
A	MEM.alu	nop	WB.alu	WB.mem
B	MEM.alu	nop	WB.alu	WB.mem

由此绘制数据重定向电路如图 3.18 所示。

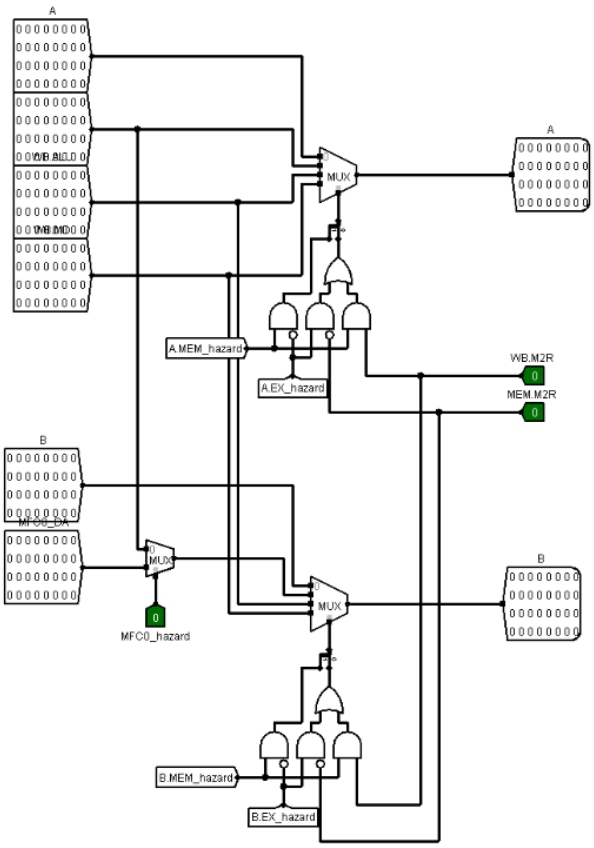

图 3.18 数据重定向电路

3.5.2 重定向流水线实现

在气泡流水线的基础上，将 EX 段的 AB 数据输入通过重定向电路更新，并将数据相关冲突引起的 ID/EX 段清空修改为由 load-use 相关引起的冲突清空，具体实现如图 3.19 所示。

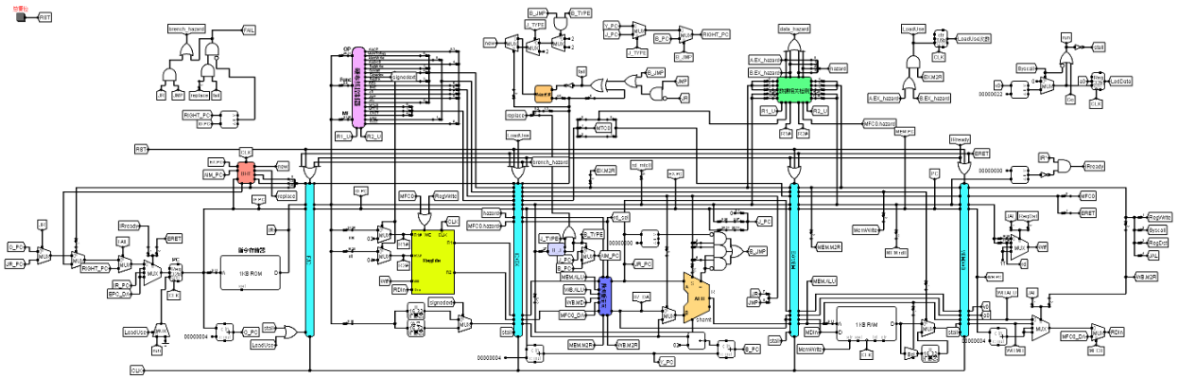

图 3.19 重定向流水线

3.6 动态分支预测机制实现

3.6.1 BHT 表的实现

采用全相连映射的方式构造 BHT 表，总共包含 8 条记录，每条记录由 1 位有效位，32 位 PC，2 位预测值，32 位计数器，32 位目标地址组成。PC 的值作为每条记录的标识，当有效位为 1 且 PC 值相等时命中该条记录，根据预测值的最高位给出下一条指令的地址。命中的同时需要把命中的记录的序号输出，否则输出计数器数值最大的记录序号。具体实现如图 3.20 所示。

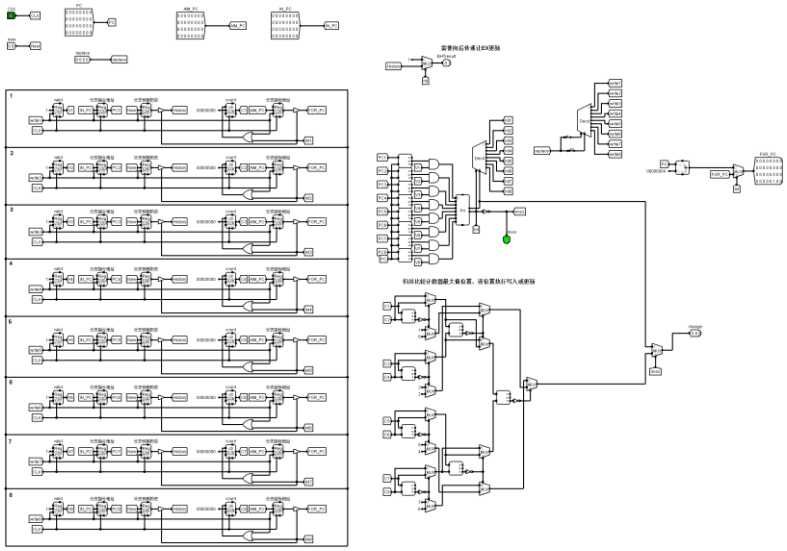

图 3.20 BHT 表

3.6.2 预测值更新逻辑和分支预测实现

根据表 3.3，使用 logisim 自动生成电路功能填写真值表生成电路。

表 3.3 预测值更新逻辑

旧值	分支成功	分支失败
00	01	00
01	10	00
10	10	11
11	10	00

在 IF 段将 PC 的值同时传入 BHT 和指令寄存器，并在下一个时钟加载 BHT 给出的预测结果。EX 段对 BHT 的预测结果进行校正并更新预测值，具体实现如图 3.19 所示。

4 实验过程与调试

4.1 测试用例和功能测试

由于理想流水只支持无数据相关和分支相关的指令，因此采用特殊的理想流水线测试.asm。

对于单周期 MIPS CPU、气泡流水以及重定向流水，采用的测试用例都是多级中断_benchmark_ccmb.asm。

4.1.1 多级中断_benchmark_ccmb

本测试用例用于对多级中断功能、24 条基本指令+4 条拓展指令功能进行测试。基准测试程序和四条拓展指令测试程序为实验包中提供的测试程序，中断测试程序则是自己修改完成。核心代码及框架如下：

```
.text
addi $sp,$zero,4092
mtc0 $zero,$1 #设置 IE
j start
sw $s6,($sp)   #保护现场
addi $sp,$sp,-4
#……#
mfc0 $a0,$2            # epc
sw $a0,($sp)
addi $sp,$sp,-4
mfc0 $a0,$3       #    inm
sw $a0,($sp)
addi $sp,$sp,-4
addi $a0,$zero,1      #设置中断屏蔽字
mtc0 $a0,$3
#开中断
```

```
mtc0 $zero,$1
#中断测试程序#
#关中断
addi $a0,$zero,1
mtc0 $a0,$1
#恢复现场
#……#
eret
#中断程序 2#
#中断程序 3#
start：
#benchmark_ccmb#
```

4.1.2 单周期 MIPS 基准程序测试

指令寄存器中加载 benchmark_ccmb(中断).hex 镜像文件，benchmark 程序理论周期数为 1545，由于电路刚上电需要执行三条初始化指令，理论周期数应为 1548 且最终 MEM 中的数据为降序的-1~14。实际执行总周期数如图 4.1 所示，停机时 MEM 中数据如图 4.2 所示。

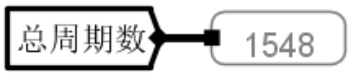

图 4.1 执行总周期数

```
000 0000000e0000000d0000000c0000000b
004 0000000a000000090000000800000007
008 0000000600000005000000040000003
00c 00000002000000010000000   ffffffff
```

图 4.2 MEM 中数据

4.1.3 单周期 MIPS 扩展指令测试

在执行完 benchmark 的基础上，使用 Go 按键继续执行扩展指令测试程序。四条拓展指令的理论执行结果及实际结果对照如表 4.1 所示，具体执行过程见附件视频。

表 4.1 扩展指令测试对照结果

指令	理论结果	实际结果
SUBU	FFFFFFF0	FFFFFFF0
SLTIU	00001949	00001949
LHU	0000C0BF	0000C0BF
BLEZ	00000000	00000000

4.1.4 单周期 MIPS 多级中断测试

由于三个中断程序主体共用，2 号、3 号中断通过 j 指令跳转执行，因此三个中断程序的时钟周期数分别为 355、356、356。选取中断组合执行 benchmark 程序，理论结果与实际执行结果对照如表 4.2 所示。

表 4.2 多级中断执行结果对照

组合	理论周期数	实际周期数
1	1903	总周期数 1903
1，2	2259	总周期数 2259
1，2，3	2615	总周期数 2615

4.1.5 理想流水测试

理想流水测试程序的理论周期为 20，同时在 MEM 中存储 0、1、2、3，测试结果与理论结果相同。

4.1.6 气泡流水 benchmark_ccmb 测试

EX 段执行分支程序的气泡流水线标准周期数为 3623，无条件分支数为 38，条件分支成功数为 276，数据相关产生气泡数为 1446，满足 3623=5-1+1545+(38+276)*2+1446。实际执行结果如图 4.3 所示。

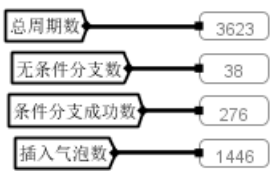

图 4.3 气泡流水统计结果

4.1.7 重定向流水动态分支预测测试

重定向流水线应使得时钟周期数降至 1800 以下，本处测试结果为 1786。由于程序在 benchmark 程序的基础上加入了三条初始化指令(包括一条无条件跳转指令)，因此 benchmark 实际执行周期为 1781。运行结果如图 4.4 所示。

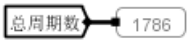

图 4.4 分支预测运行结果

4.1.8 重定向流水多级中断测试

流水中断的周期数由于中断时间的不同，会导致插入的气泡数目不同，因此采用运行结果正确性的方式进行验证，即最终 MEM 中的数据应为-1~14 的降序。实际运行结果如图 4.5 所示。

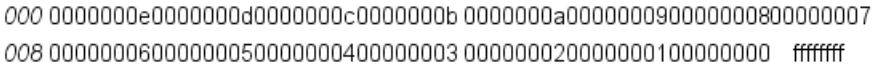

图 4.5 流水中断运行结果

4.2 性能分析

虽然流水线 MIPS CPU 的时钟周期数比单周期 MIPS CPU 的时钟周期数要多，但是每个周期的时延比单周期MIPS小很多。单周期在一个时钟周期最多要完成取值、译码、执行、访存、写回五个动作，需要耗费较长的时间，而流水线 CPU 在一个时钟周期只需要完成五个动作中的一个，时延为时延最长的访存阶段。因此流水线的性能会比单周期的性能好很多。

分支预测重定向和气泡流水线的关键时延都为访存阶段，而分支预测重定向流水线的周期数更少，因此分支预测的性能会更好。

4.3 主要故障与调试

4.3.1 数据显示故障

气泡流水线： 程序运行正常但是数码管不显示对应数据。

故障现象：数据显示指令异常，数码管不能显示正确的数据。

原因分析：整个流水线程序能够正确运行，只有数据不能正确显示，则大概率是

待显示的数据没有正确传输到 WB 阶段(syscall 指令要在 WB 阶段执行)。检查线路发现，ID 阶段正确读取的数据并传递到 EX 段，但是 EX 段只传递了运算后的数据 ALU 以及第二操作数 B，则 a0 寄存器和 v0 寄存器的数据并没有完成的传输到 MEM 阶段，也就不可能完成传输到 WB 阶段。

解决方案：在 EX/MEM 和 MEM/WB 接口中添加 a0 和 v0 数据的传送接口，如图 4.6 所示。

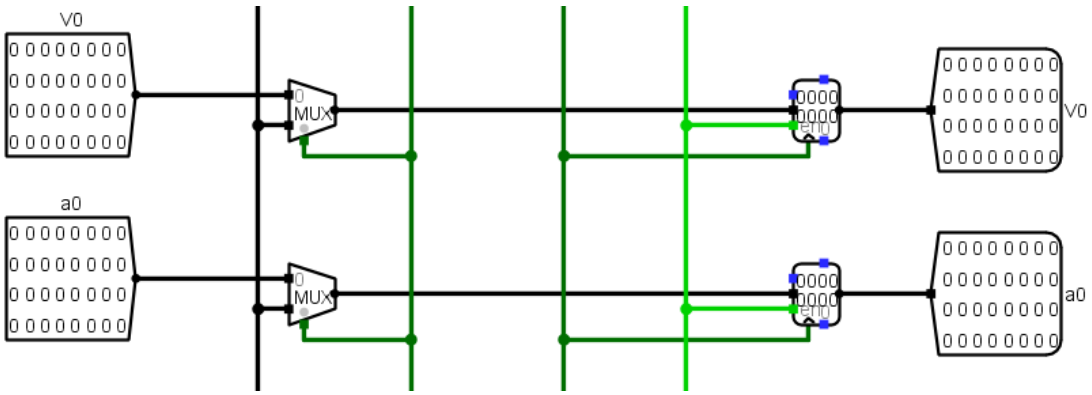

图 4.6 新增流水传送接口

4.3.2 数据重定向故障

重定向流水线中有部分数据的重定向发生错误，最终的时空图和标准时空图有部分差异。

故障现象：重定向流水线中部分数据的重定向发生错误，与标准时空图对照有部分时空图不一致，最后内存的数据不正确，但是总周期数正确。

原因分析：在根据标准对照后的时空图错误位置定位后，发现共性特征是 EX 和 MEM 段都出现了数据冲突。在原本的数据重定向逻辑设计中没有能够考虑到这种情况的产生，只单独考虑了 EX 段或 MEM 段发生数据冲突时的处理情况。

解决方案：修改重新向数据的选择逻辑，多路选择器(4 选 1)的选择端高位为 MEM 段冲突信号与 EX 段冲突信号取非的与结果，低位为 MEM 段冲突信号和 WB 段 MEMTOREG 信号的与结果或上 EX 段冲突信号与 MEM 段 MEMTOREG 信号取非的与结果。这样当选择端为 00 时，数据即为原数据，当选择端为 01 时，数据重定向为 MEM 段的 ALU 结果，当选择端为 10 时，数据重定向为 WB 段的 ALU 结果，当选择端为 11 时，数据重定向为 WB 段的访存结果。这样修改使得同时发生 EX 段和 MEM 段的数据冲突且 EX 段的数据冲突不为 load-use 冲突时，数据被重定向为最新

的 MEM 段的 ALU 结果，这与逻辑事实相符合。

4.4 实验进度

表 4.3 课程设计进度表

时间	进度
第一天	阅读课设任务书，阅读 MIPS 指令手册，完成单周期 MIPS 控制信号表，完成主控制器，完成 24+3+4 单周期 MIPS CPU 数据通路及功能测试。
第二天	完成理想流水、气泡流水设计及功能测试。
第三天	完成重定向流水设计及功能测试，阅读中断相关资料，构思中断设计思路。
第四天	完成单周期中断设计和流水中断设计及正确性测试。
第五天	阅读分支预测相关资料并完成 BHT 表的设计。
第六天	完成分支预测的设计及优化，使得重定向流水线时钟周期数降至 1800 以下。
第七天	研究 CS3410 扩展包中器件使用方法，构思团队任务可以完成哪些功能。
第八天	初步确定了团队任务要做的东西为基于 LCD 屏播放小猪佩奇并开始尝试改造 LCD 屏以提高刷新效率。
第九天	LCD 屏改造成功但是多核并行不知道如何解决但最终确定了团队任务为设计一台集小游戏和 ppt 图片放映为一体的小电视，并开始第二次改造 LCD 屏以契合单核运行。
第十天	屏幕改造成功，运行方法解决，但是由指令加载一个像素过慢，改造的目的无法实现，开始构思解决办法。
第十一天	对电路进行改造，通过外存控制整幅图片加载，团队任务基本完成。
第十二天	试玩并解决了小游戏中存在的 bug，对电路进行优化封装，任务全部完成。

5 团队任务

5.1 任务描述

我们团队在综合现有元器件以及多次讨论的基础上决定设计一台基于支持多级中断的单周期 MIPS CPU 的集小游戏、ppt 放映、图片查看为一体的小电视，在 128*128 像素的小屏幕上使用我们自主设计的 CPU 带来足够震撼的观感。

5.2 个人分工

队内分工明确，我在本任务中主要负责电路图的绘制封装以及元器件的改造。元器件的改造主要是针对 LCD 屏幕元件。

5.3 详细设计

5.3.1 LCD 屏幕改造

由于 CS3410 提供的 LCD 屏幕在一个时钟周期仅支持更新一个像素，使得更新一帧 128*128 的图片需要上万个时钟周期，这大大影响观看体验。因此我将 LCD 的坐标和 RGB 整合为 1 个输入，同时一个时钟周期支持处理四个这样的输入，使得加载速度近似提升了 4 倍。主要代码如下：

```
       int xyc1 = addr(circuitState, P_INONE);
       int xyc2 = addr(circuitState, P_INTWO);
       int xyc3 = addr(circuitState, P_INTHREE);
       int xyc4 = addr(circuitState, P_INFOUR);
       int x = (xyc1 >> 24) % (1 << 7);
       int y = (xyc1 >> 16) % (1 << 7);
       int color = (xyc1) % (1 << 16);
       /*使用x2-x4、y2-y4、color2-color4处理xyc2-xyc4*/
       if (state.tick(val(circuitState, P_CLK)) && val(circuitState, P_WE) ==
Value.TRUE) {
           Graphics g = state.img.getGraphics();
           g.setColor(new Color(state.img.getColorModel().getRGB(color)));
```

```
            g.fillRect(x*2, y*2, 2, 2);
            /*对x2-x4，y2-y4，color2-color4执行同样操作*/
        }
```

5.3.2 电路绘制

使用者的每一个操作都可视为一个中断，因此需要设置电源、游戏功能、方向(上下左右)、PPT 功能、PPT 控制(前、后)以及 Home 键共 10 个按键，即十个中断。当然电源键和 Home 键的中断优先级最高，功能按键次之，实际操作按键(方向、PPT 控制)优先级最低，所以需要设置中断屏蔽字，这里采用的是两位的中断屏蔽字，方向键共用一位，PPT 控制键共用一位。如图 5.1 所示。

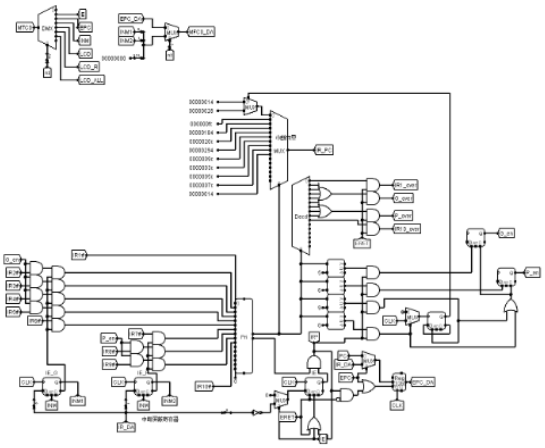

图 5.1 中断控制逻辑

由于小游戏的碰撞检测数据需要先存在 ROM 中，所以我们在 CPU 内部增加了一块 ROM，通过 rt 编号的最低位确定访问的数据在 ROM 中还是 MEM 中。

为了避免封装嵌套带来性能下降，我们设置了 256KB 的外存，用于存放固定不变的图片数据。通过 MTC0 指令对 6 号寄存器的操作使 CPU 暂停工作，由计数器控制数据传输，4096 个时钟周期后将控制权转移到 CPU。控制转移逻辑如图 5.2 所示。

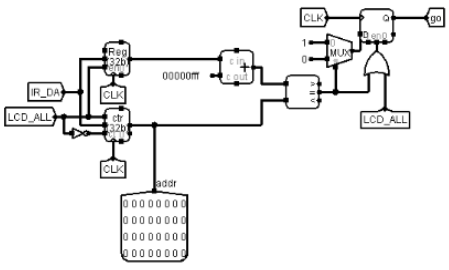

图 5.2 控制转移逻辑

5.4 成果展示

我们用队名将我们的 CPU 命名为齐得龙东强，小电视命名为东强 TV(DQTV)，CPU 以及外存的结构封装成 DQTV 的底座，最终成果如图 5.3 所示。

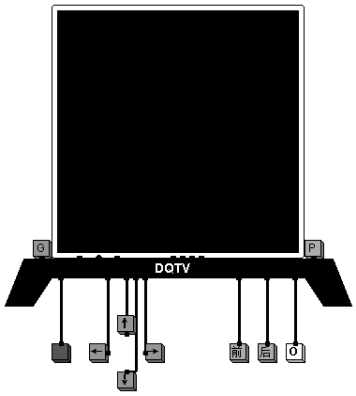

图 5.3 东强 TV 最终成果

游戏中的界面如图 5.4 所示，通关之后会有彩蛋，篇幅限制这里便不再展示。

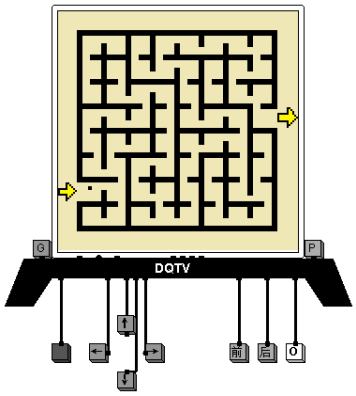

图 5.4 游戏功能界面

PPT 放映功能中的首页如图 5.5 所示，完整的 PPT 会在答辩中完整展示。

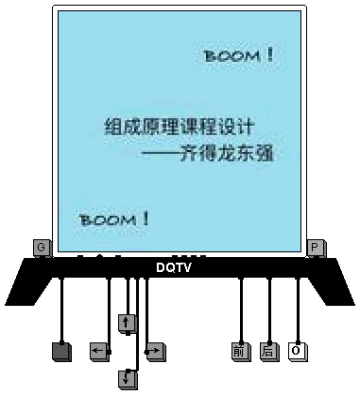

图 5.5 PPT 放映首页

6 设计总结与心得

6.1 课设总结

本次课程设计中我完成了如下几点工作：

1） 完成了单周期硬布线 MIPS CPU(24 条基本指令+4 条拓展指令+3 条中断相关指令)控制器的逻辑设计以及线路图绘制。

2） 完成了流水线接口设计。

3） 完成了理想流水线逻辑设计及线路图绘制。

4） 完成了气泡流水线的逻辑设计及线路图绘制。

5） 完成了重定向流水线的逻辑设计及线路图绘制。

6） 完成了 BHT 表的设计和分支预测更新逻辑的设计与实现。

7） 完成了基于重定向流水线的分支预测逻辑设计及线路图绘制。

8） 完成了基于单周期硬布线 MIPS CPU 的多级软中断逻辑设计和线路图绘制。

9） 完成了基于重定向分支预测流水线的多级软中断逻辑设计和线路图绘制。

10） 完成了基于 CS3410.jar 中 LCD 模块的功能改造。

11） 完成了团队项目的线路图设计。

6.2 课设心得

本次课程是线上课程的第一门实验课，但是整体感觉比较良好，部分原因是理论课程及先行实验课程都在上学期线下完成，有一定的基础，另一部分也离不开课程组的正确安排，课程整体体验是本学期体验最好的课程。

我在整个课程设计中的任务顺序是单周期硬布线到拓展指令，再转向流水线和中断设计，最后完成分支预测。我个人觉得这是一个很合理的实验顺序，单周期硬布线在上学期的实验课程基础上可以很容易的完成，拓展指令在有老师提供的表格情况下也很容易，后续再对流水线进行较为深入的研究，搞清楚每个段需要的数据来源以及是否需要传递到下一段就可以完成流水线的设计，并且老师提供的标准时钟对照表可以针对的找出错误部分，在临近时钟周期进行单步调试，配合反汇编探针也很容易就

能找出是哪些数据的传递出现了问题。分支预测本质上是一个 cache，cache 表可以根据上学期的 4 路组相连 cache 完成设计，而预测值的更新逻辑以及分支预测的实现逻辑在资料中都有详细介绍，只要弄清楚问题也不是很大。中断部分是我在实验的后半段才摸清它的设计方式，刚开始面对那个中断按键产生电路完全没有头绪。主要是不清楚对 MFC0,MTC0 以及 ERET 三条指令的运用，后来是在和同学的讨论之中才理清了思路，最后完成了多级中断的设计。

我们团队应该是最先全组拿到个人满分的(主要是在假期中提前看了任务，在课程正式开始时已经基本完成了气泡流水线的设计)，因此在第二周我们就开始了团队任务的设计。我们使用 ZOOM 和 QQ 电话进行了几次讨论，在已有条件的基础上确定最终的成品为能够展示视频/图片并具有一定交互性的设备。但是最初的 LCD 表现特别令人失望，一个时钟周期只能够完成一个像素点的更新会使得使用体验极大的下降，因此我们决定从元件入手。第一次更新的方案是用四块 LCD 和四块 CPU 进行并行更新，由于对并行执行的不够了解导致该方案被否定，所以我们使用第二种方案，用一块 CPU 和一块 LCD 进行更新像素点，同时在一个时钟周期内加大并行输入的像素点数据。综合考虑了 CPU 性能和时钟周期之后，我们决定在一个时钟周期内传入四个像素点的坐标及 RGB 值，最终的性能相较之前有了几乎四倍的提升。为了避免多层嵌套封装带来的性能下降，我们使用外存存储坐标和 RGB 信息，并通过控制权转移更新图像。最终的成品具有一定的交互性，更快的图像加载速度带来的交互体验也比较良好。

本次课程设计让我对 CPU 的工作方式有了更加深刻的理解，更加感受到对于 CPU 性能的提升是一件多么重要的事，收获颇丰。对于疫情中的第一门课，我觉得课程组所有老师的处理方式是特别好的，并没有造成特别大的影响。当然，几个 EXCEL 文件使得整个课程设计的难度下降了许多，建议可以取消几个表格的提供，除了几个时空对照表以外，其他的表都可以撤掉。我觉得中断部分的介绍稍微偏少，可以在慕课中添加一讲中断部分的内容。

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: