华中科技大学

2023

硬件综合训练　　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：	CS2106
学　　号：	U202115514
姓　　名：	杨明欣
电　　话：	13390396012
邮　　件：	ymx@hust.edu.cn

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

（8） 支持表 1.1 前 27 条基本 32 位 MIPS 指令；

（9） 支持教师指定的 4 条扩展指令；

（10）支持多级嵌套中断，利用中断触发扩展指令集测试程序；

（11）支持 5 段流水机制，可处理数据冒险，结构冒险，分支冒险；

（12）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。

（13）能运行教师提供的标准测试程序，并自动统计执行周期数

（14）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	

#	指令助记符	简单功能描述	备注
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	XOR	异或	
29	SLTIU	无符号小于立即数则置位	
30	LB	取字节，符号扩展写入	
31	BGE	大于等于时分支	

2 总体方案设计

2.1 单周期 CPU 设计

单周期 CPU 设计本次我们采用的方案是硬布线控制方案，将指令与数据分开存储，使用硬布线将二者进行联系。单周期 CPU 采用同步时序，CPU 内所有元器件均由同一时钟周期控制。在一个时钟周期内，首先控制器根据 PC 寄存器值读取指令并转化为信号，根据信号指令执行 ALU 的计算，写入内存或寄存器等操作，然后再取下一条指令循环往复。

总体结构图如图 2.1 所示。

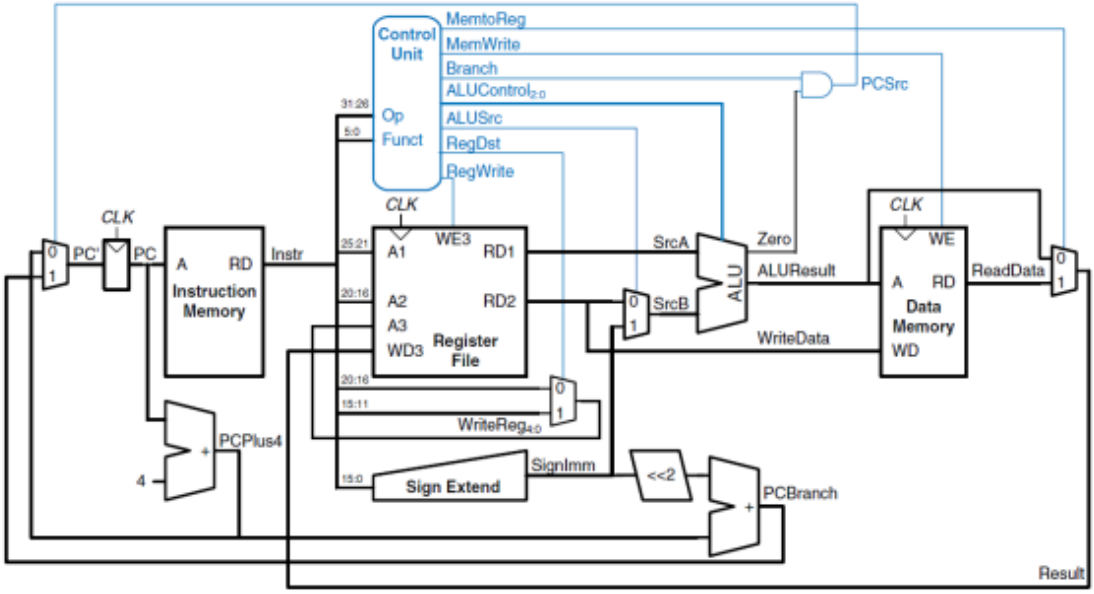


图 2.1 总体结构图

2.1.1 主要功能部件

以下为单周期 CPU 的几个核心部件的设计原理。

1. 程序计数器 PC

程序计数器 PC 的功能是存储指令的地址，在时钟上升沿时将下一条指令的地址载入寄存器，同时输出指令地址给指令寄存器 IM，根据指令的不同，下一条指令的

地址也有所不同，本次设计中主要包括四种地址：顺序地址 PC+4、B 型指令分支跳转地址、J 型指令分支跳转地址、JALR 跳转地址。

同时 PC 寄存器的另一个功能是控制 CPU 停机，将停机信号连接在寄存器使能端，在产生停机信号后 PC 寄存器不会再读出新的 PC 地址，即整个 CPU 将会循环执行最后一条指令 ecall，而 ecall 不涉及对任何模块的写入操作，通过控制 PC 寄存器实现了 CPU 的停机操作。

程序计数器 PC 还支持复位功能，通过 rst 指令清空当前 PC 寄存器的值。

2. 指令存储器 IM

指令存储器存储了程序的具体指令，具体执行指令地址由 PC 的输出地址决定。由于 RISC-V 为定长指令，每条指令均为 4 字节，因而 IM 中地址仅有 PC 输出地址中的 2-11 位决定。

3. 运算器

运算器的主要工作为，根据控制器传来的 ALUOP 指令，将两个操作数进行运算。其输入和输出接口如表 2.1 所示。

表 2.1 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，具体功能见下表
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

RISC-V 中有 32 个寄存器，这 32 个寄存器的值则存储在具有读写功能的寄存器堆 RF 中，寄存器堆有两个输出寄存器值的引脚，为了方便处理 R 型指令，相应的它也有两个读地址输入引脚；WE 信号控制寄存器堆将数据写入写地址对应的寄存器中，寄存器堆在单周期CPU中是上升沿触发，而在气泡和重定向流水线中是下降沿触发。

5. 数据存储器 MEM

和指令存储器结构类似，也是由 10 条地址线（2-11 位）进行寻址，要求访问按字对齐。寻址的地址来源于寄存器中的地址或者 ALU 的计算结果。如果需要访问某个特定的字（例如 LB 指令），或者写入到某个非对齐的地址，需要先将对应的字取出，然后使用 0-1 位来从中选择取出的字节，写入同理，写信号为 MemWrite，时钟上升沿时写入数据。

2.1.2 数据通路的设计

依照图 2.1 所示的总体结果，将各个模块相连构建数据通路。需要另外补充次要功能模块以完善数据通路。首先需要完成立即数生成模块 ImmGen，用于从 IR 中解析出立即数，我们根据硬布线控制器输出的控制信号可以得到不同类型指令的信号：B 型分支信号 branch = beq || bne || bge，S 型信号 S_Type，J 型分支信号 JAL，如果这三个信号都为 0 则对应的是 I 型指令，根据 RISC-V 的指令架构，可以利用分线器对于 IR 指令进行处理，得到四类指令对应的立即数，然后通过多路选择器进行选择即可，多路选择器的输入端从 0-3 分别对应 I、S、B、J 型指令的立即数。

同时需要引入差异化指令，数据通路的构造还需在原有数据通路的基础上完善。引入 ecall 指令，控制着单周期 CPU 的数据在七段译码器上的显示与停机，需要处理 a0 和 a7 的值，因此要在 Regifile 的输入端添加两个多路选择器，当 ecall 信号产生时，按照指令的要求，输入的地址不再是从 IR 中分线出的地址，而是 a0 和 a7 的地址。将 a7 与 34 作比较，如果不相等则产生停机信号 并利用 D 触发器锁存，反之则将 a0 的值输出至数码管。

因为 LB 指令的特殊性，需要对于取出和存入的字节而不是字进行处理，因此需要借助多路选择器和 0-1 号取址信号，按需选出字并进行符号扩展后再处理。同时因

为 LB 指令的 op 和 func3 均为 0，需要与 nop 指令进行区分，因此需要增加一个 LB 和 nop 指令区分模块。

2.1.3 控制器的设计

控制器需要从指令中得到源操作数的具体数值（如 I 型指令）或地址，以及对取出的数字进行的操作种类，和最后写入的寄存器编号或内存地址。其中最核心的就是控制信号，通过控制信号可以对整个电路的功能进行控制。控制器的控制信号定义与作用如表 2.3：

表 2.3 主控制器控制信号的作用说明

控制信号	取值	说明
AluOP	0-12	使用 ALU 的指令编号
BEQ	0、1	是否为 BEQ 指令
BNE	0、1	是否为 BNE 指令
MemToReg	0、1	是否需要从内存写入寄存器
MemWrite	0、1	是否需要写入内存
AluSrcB	0、1	第二操作数是为寄存器 2 中值（0）还是立即数（1）
RegWrite	0、1	是否需要写入寄存器
LB	0、1	是否为 LB 指令
JALR	0、1	是否为 JALR 指令
JAL	0、1	是否为 JAL 指令
BGE	0、1	是否为 BGE 指令
URET	0、1	是否需要中断返回
S_Type	0、1	是否为 S 型指令
Ecall	0、1	是否为 ecall 指令

下面为每条指令对应的硬布线信号器应该产生的信号逻辑，需要说明的是对于 BEQ 信号、BNE 信号、Jal 信号、jalr 信号、BGE 信号和 LB 信号来说，其逻辑值为 1 的情况有且只有一种，且关联指令到来时信号逻辑值为 1，因此不在表中列出。该控制信号表的框架如表 2.3 所示。

表 2.2 主控制器控制信号框架

指令	ALU_OP	MemToReg	ALU_Src	RegWrite	Ecall	S_Type
ADD	5		1	1		
SUB	6			1		
AND	7			1		
OR	8			1		
SLT	11			1		
SLTU	12			1		
ADDI	5		1	1		
ANDI	7		1	1		
ORI	8		1	1		
XORI	9		1	1		
SLTI	11		1	1		
SLLI	0		1	1		
SRLI	2		1	1		
SRAI	1		1	1		
LW		1	1	1		
SW			1			1
ECALL					1	
BEQ	6					
BNE	6					
JAL						
JALR			1			
URET					1	
XOR	9				1	
SLTIU	12			1	1	
LB		1		1	1	
BGE	6					

2.2 中断机制设计

2.2.1 总体设计

中断类型大体可以分为内部中断和外部中断，本次实验设计的中断为可屏蔽的外部中断。本次实验中完成了支持单级中断的单周期 CPU 以及支持多级中断的单周期 CPU。 对于所有的外部中断，跳转到中断指令的方法一般为：保存下一条指令地址，保护现场的寄存器，然后跳到中断源执行中断。从中断返回的方法一般为：根据 ecall 指令恢复中断现场的寄存器，指令跳转到存储的下一条指令的地址。所需要的中断的构成模块相同包括：中断控制器、中断使能信号产生模块、中断请求信号产生模块、中断清零信号产生逻辑、中断入口逻辑模块、保护现场模块等。中断的执行基本逻辑如图 2-2 所示。

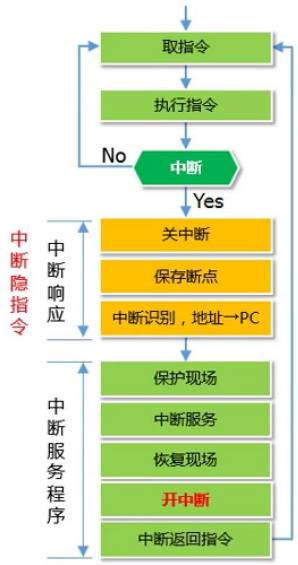


图 2-2 中断处理机制

2.2.2 硬件设计

本次实验设计实现的中断方法为硬件保存，即对于每一种类、每一等级的中断，都单独使用一套完整的寄存器进行现场的保存。具体而言，一组中断寄存器由 EPC 和 IRID 分别保存中断返回地址和当前执行的中断号。此外还用 IE 寄存器保存当前是否允许执行更高等级的中断的 IE 信号，高电平有效。

对于两种中断，均需要支持一个中断请求队列，即在中断执行过程中接受到的中断在当前中断周期结束后转而去执行队列中的请求。这里只需要对每个中断请求使用

一个寄存器保存是否需要执行该中断的信号，若执行完整则激活对应清除请求的 reset 信号，即可将执行完整的中断从队列中弹出。

对于单级中断，由于在执行中断程序时不允许被中断去执行其他中断程序，因而实现较为简单，仅需使用一组中断寄存器保存当前中断的信息即可。利用中断使能寄存器存储使能信号 IE 和中断信号进行与运算，只有在产生中断信号并且处于使能阶段才能跳转到中断服务程序中。同时要保护现场，要保护的现场即为单周期 CPU 的 PC 寄存器输入端的数据，寄存器的使能端为中断请求信号，当产生中断请求时就会将 PC 寄存器的输入端的值存到寄存器中。处理完中断服务程序后，当产生 URET 信号时则要进行中断返回，以 URET 作为多路选择的信号将断点传回 PC，利用中断号来产生相应中断的清零信号。

对于多级中断，由于支持了中断程序继续被中断，因而需要三套中断寄存器保存中断信息。使用优先编码器来对当前中断信号进行选择，若正在执行中断程序且此时允许被打断则利用中断屏蔽字和中断号的比较器判断当前中断是否允许打断当前终端，这样可以动态的改变中断优先级。

2.2.3 软件设计

不需要软件部分支撑。

2.3 流水 CPU 设计

2.3.1 总体设计

本次实验实现的是五段流水线 CPU，即将一条指令分成取指 IF、译码 ID、执行 EX、访存 MEM、写回 WB 五个阶段，通过四组寄存器保存每一段所需要的信息来实现每条指令五个阶段执行的相对独立性，这也是理想流水线的基本设计原理。但是在实际执行过程中，由于存在无条件和有条件跳转指令，因而有些指令的具体效果需要等待上条指令的结果，破坏了指令之间的隔离关系。因而在理想流水线上根据消除影响逻辑和方式不同，分成了气泡流水线和重定向流水线这两大类。此外，利用动态分支预测可以进一步加速了重定向流水线。

总体结构图如图 2-3 所示。

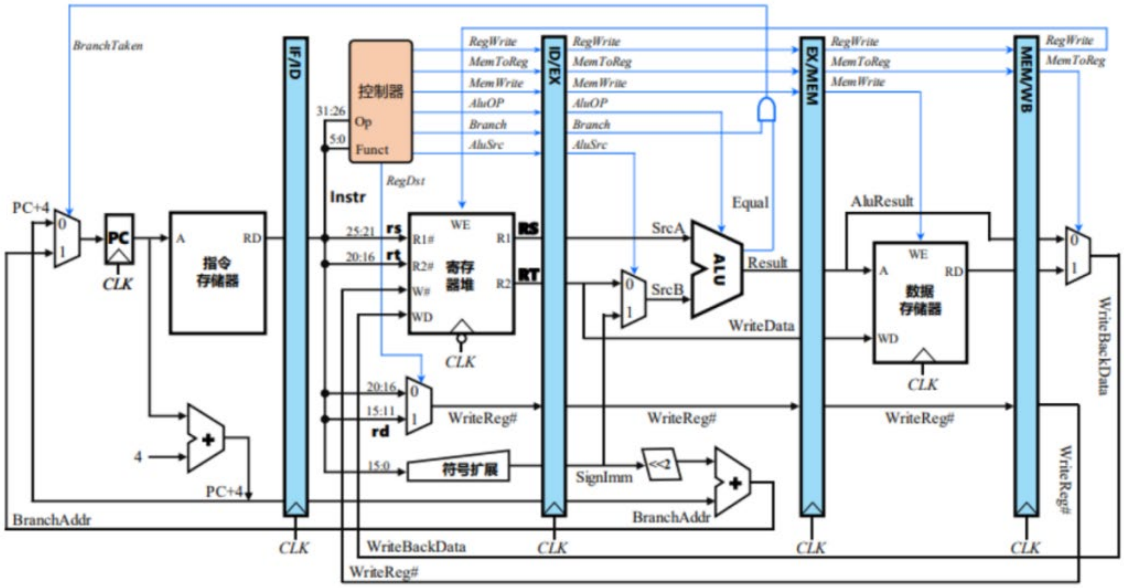


图 2-3 总体结构图

2.3.2 流水接口部件设计

四个流水寄存器 IF/ID，ID/EX，EX/MEM，MEM/WB 分别存储相邻两个阶段之间需要传送的数据与控制信号，流水寄存器上升沿触发存储，并且需要有复位端（高电平有效）和使能端（低电平有效），便于支持后续气泡流水线和重定向流水线的设计，各个流水寄存器的存储数据与信号如表 2-3 所示。

表 2-3 不同阶段所需流水线接口

阶段	所需流水线接口
IF/ID	IR、PC
ID/EX	BEQ、BNE、MemToReg、MemWrite、AluSrcB、RegWrite、S_Type、Ecall、JAL、JALR、LB、BGE、URET、ALUOP、R1、R2、IMM、PC、IR、R1_Forward、R2_Forward
EX/MEM	JAL、JALR、MemToReg、MemWrite、LB、RegWrite、Halt、ALUResult、Data、IR、PC、LB
MEM/WB	JAL、JALR、MemToReg、RegWrite、Halt、ALUResult、ReadData、IR、PC、LB

说明：上面的某些接口可能并不是在所有的流水线设计中都需要用到，例如 R1_Forward 只在重定向流水线中才需要用到，但为了完整，也列在表中。

2.3.3 理想流水线设计

理想流水线由于不涉及同时进入流水线的指令之间的相互作用，因而只需要将单周期的 CPU 按照上述五个步骤直接进行拆分，使用四组流水线接口即可。对于停机指令，在 EX 段处理完停机逻辑后，需要经过两个周期传送到 WB 段再最终执行停机操作，以保证全部指令都执行完毕再停机。

同时理想流水线并不能支持分支指令，存在误取指令的现象，同时由于数据冲突的存在也会导致一些指令无法正确得执行。

2.4 气泡式流水线设计

为了支持同在流水线的指令的相互作用关系，在理想流水线的基础之上需要增加由当前的指令操作信号控制的 DataHazzard 组合逻辑以判定是否需要插入气泡，插入气泡等效于清空流水线接口内寄存器的值。DataHazzard 的主要判定逻辑是根据源寄存器的使用状态和数据相关检测来判定是否存在冲突。

其中 DataHazzard 信号对应的逻辑如下：

```
DataHazzard = R1Used & (R1Adr≠0) & EX.RegWrite & (R1Adr==EX.rd) + R2Used & (R2Adr≠0) & EX.RegWrite & (R2Adr==EX.rd) + R1Used & (R1Adr≠0) & MEM.RegWrite & (R1Adr==MEM.rd) + R2Used & (R2Adr≠0) & MEM.RegWrite & (R2Adr==MEM.rd)
```

具体而言，若判定需要插入气泡，则 ID/EX 接口和 IF/ID 接口均需要清空，以跳转到正确的指令处重新执行。由于指令的相互作用，气泡被插入了流水线，导致流水线的效率下降。

2.5 数据转发流水线设计

数据转发流水线（即重定向流水线）以气泡流水线的基础之上进行了一定的优化，结构类似，但是减少了气泡的插入。重定向将数据可能的全部路径进行汇总，然后根据指令的执行情况获取数据，选择一条分支。但是若相邻两条指令存在数据相关，且前一条是访存指令，此时就会出现冲突（称为 Load_Use），此时仍需要使用气泡来避免冲突。

其中 Load_Use 信号对应的逻辑如下：

Load_Use = R1Used & (R1Adr≠0) & EX.MemToReg & (R1Adr==EX.rd) + R2Used & (R2Adr≠0) & EX.MemToReg & (R2Adr==EX.rd)

同时，选择多路分支时需要的多路选择信号 R1_F、R2_F 也要根据指令的情况进行生成，以确保使用的 R1 和 R2 的正确性。

其中 R1_F、R2_F 信号对应的逻辑如下：

R1_F = (R1Used & (R1Adr≠0) & EX.RegWrite & (R1Adr==EX.rd)) ? 2 : (R1Used & (R1Adr≠0) & MEM.RegWrite & (R1Adr==MEM.rd))?1:0

R2_F = (R2Used & (R2Adr≠0) & EX.RegWrite & (R2Adr==EX.rd)) ? 2 : (R2Used & (R2Adr≠0) & MEM.RegWrite & (R2Adr==MEM.rd))?1:0

具体而言，在对应不同流水寄存器的使能端和复位端，以及 PC 寄存器的停机信号生成逻辑可以根据 Load_Use 生成。

2.6 动态分支预测机制

2.6.1 设计原理

动态分支预测是在重定向流水线上进一步的优化。重定向只是在气泡流水线上减少了气泡的产生，但是并未减少分支误取得代价，导致了周期的浪费。动态分支预测使用一个 BHT 表来记忆过去的跳转指令，使用 LRU 算法来动态更新 Cache 槽的存储内容，同时使用双预测位的四状态的状态机来判定是否需要跳转，如所示。不断根据当前实际跳转情况来进行状态机的更新。具体而言，电路在 IF 段通过状态判断出预测结果，在 EX 段根据实际结果通过硬件来更新状态机。通过这两个方法来实现预取得正确的指令，减少分支误取的代价。

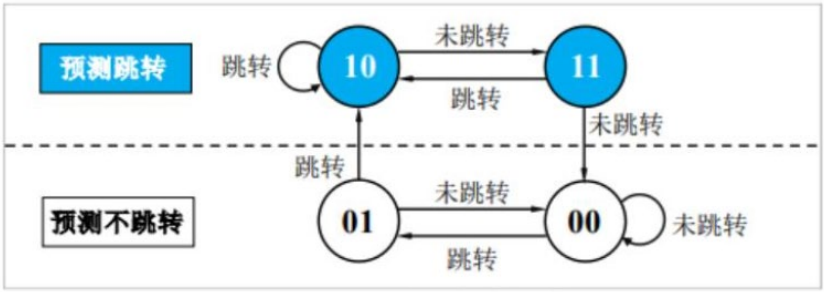


图 2-4 分支预测历史位状态转移图

2.6.2 BHT 表的设计

BHT 表全程分支预测分支历史表，存放了每次分支指令的地址、分治目标地址、历史跳转位、valid 位，硬件设计为 8 位的 Cache 槽。在具体插入地址时，优先插入空行，其次是最古老的跳转地址（即 LRU 算法）。生成 IF 段分支指令命中 L 和 EX 段分支指令命中的信号 M，在 IF 段命中时清空相应 cache 行的淘汰计数，采用 D 触发器存储以避免毛刺，EX 段命中时更新相应 cache 行的分支预测历史位，在 cache 行未满时按照 cache 行编号从大到小的顺序写入，否则则替换掉淘汰计数最大的那个 cache 行。

2.6.3 动态分支预测的流水线设计

动态分治预测在重定向流水线的基础上，增加了利用 BHT 进行预测功能。有以下三种情况：

1. BHT 未命中，则与正常重定向流水线行为一致。

2. BHT 命中但预测失败。此时需要插入气泡，将预取指令取出，并且更新 BHT 表。此时与正常重定向流水线行为大致一致。

3. BHT 命中且预测成功。此时减少了气泡的插入，且 IF 指令预取正确，电路继续指令。

整体而言，预测成功可以减少平均误取深度和气泡插入数目，因而效率相对于重定向流水线又有了进一步的提升。

2.7 流水中断机制

本次实验主要完成了流水中断的多级中断。支持多级中断的流水中断设计与单周期支持多级中断的 CPU 很相似，但是又有很大的不同，涉及到流水线 5 个阶段的分别处理，本次实验选择在 EX 段响应中断。

在中断处理过程中，五个流水阶段要进行不同的处理：MEM 和 WB 的两条指令要执行完，IF 和 ID 的两条指令要清空，EX 段的指令不执行，保存断点并重启流水线，将中断服务程序的地址传入 PC 执行中断服务程序，URET 指令执行后返回断点处继续执行指令。中断服务程序的处理与单周期多级中断的处理完全相同。

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）

① Logism 实现：

使用一个 32 位寄存器实现程序计数器 PC，触发方式为下降沿触发，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。Halt 为停机信号，当需要进行停机时，Halt 控制信号为 1，经过非门之后为 0，接入计数器使能端，忽略时钟信号，使整个电路停机。如图 3.1 所示。

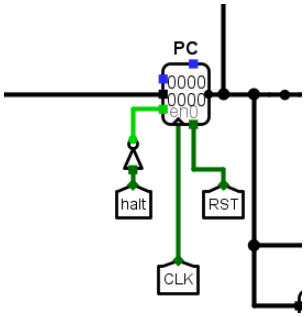


图 3.1 程序计数器（PC）

② FPGA 实现：

程序计数器 PC 的 Verilog 代码如下：

```
always @(posedge clk)
 begin
      if (rst) begin
            temp <= 0;
            end
      else begin
            if (load) begin
                  temp <= D;
                  end
            else begin
```

```
                temp <= temp;
                end
        end
    end
    assign Q = temp;
```

2) 指令存储器（IM）

① 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。

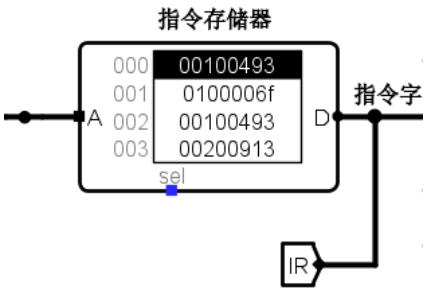


图 3.2 指令存储器（IM）

② FPGA 实现：

使用一个可读写存储器 RAM 实现指令存储器（IM）。选择 RAM 的数据位宽为 32 位，因为该 RAM 的地址位宽为 10 位，所以选择 RAM 的大小选择为 1024。

指令存储器 IM 的 Verilog 代码如下：

```
    initial
    begin
        data=0;
        $readmemh("I://Desktop//2C.txt", ram);
    end


    always@(addr)
    begin
```

```
        data <= ram[addr];
    end
```

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

3) 内存（MEM）

① Logisim 实现

电路中的内存和指令存储器一样，是由十位地址线（2-11 位）寻字，低两位寻字节的存储器，如图 3-3 所示。

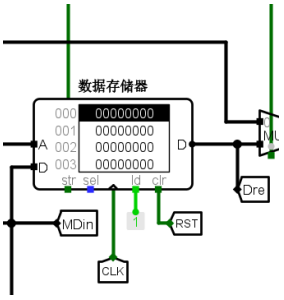


图 3-3 内存（MEM）

② FGPA 实现

内存（MEM）的 Verilog 代码如下：

```
    always @(posedge clk) begin
        if (str) ramm[addr] <= din;
    end


    always @(rst or ld or addr) begin    // note: read memory don't need clk signal
        if (rst) begin
            for (i = 0; i < 2**11; i = i + 1) begin
                ramm[i] = 32'b0;
            end
        end
        if (ld) data = ramm[addr];
    end
```

4) 寄存器（RegFile）

① Logisim 实现

RegFile 由 32 个 32 位的寄存器构成，其主要接口和功能都已经在总体设计方案中阐明。在设计包中 Logisim 的 RegFile 电路已经封装完成，因而只需要直接调用即可。Losigim 实现如图 3.4 所示：

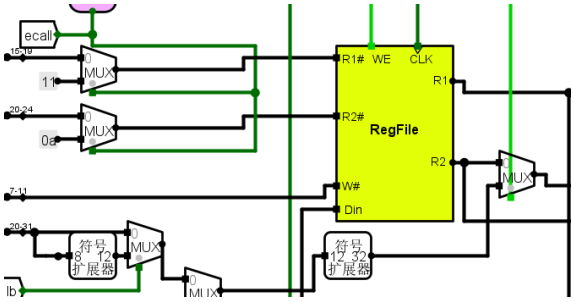


图 3.3 寄存器文件（RegFile）的 Logisim 实现

② FGPA 实现

Verilog 中将其与 ram 的设置相似，开辟了一块 ram 存储空间，初始化为全 0 状态。

```
initial begin
        for(i = 0; i <= DATA_WIDTH-1; i = i + 1) begin
            regfile[i] = 0;
        end
    end

    assign r1 = regfile[r1_addr];
    assign r2 = regfile[r2_addr];

    always @(posedge clk) begin
        if (write_en && w_addr != 0) regfile[w_addr] <= din;
    end
```

5) ALU

① Logisim 实现

ALU 的设计较为简单，按照 ALUOP 对应的计算指令进行设计即可。

② FPGA 实现

Verilog 中仅需要将每个 ALUOP 对应的计算指令进行转化即可，代码相对简单，

因而不再报告中赘述。

3.1.2 数据通路的实现

数据通路基本由取指令，指令译码、执行指令、访存取数、结果写回五大流程构成，根据每条指令的不同，控制信号会选择具体数据通路。

具体数据通路的实现如图 3-4 所示。

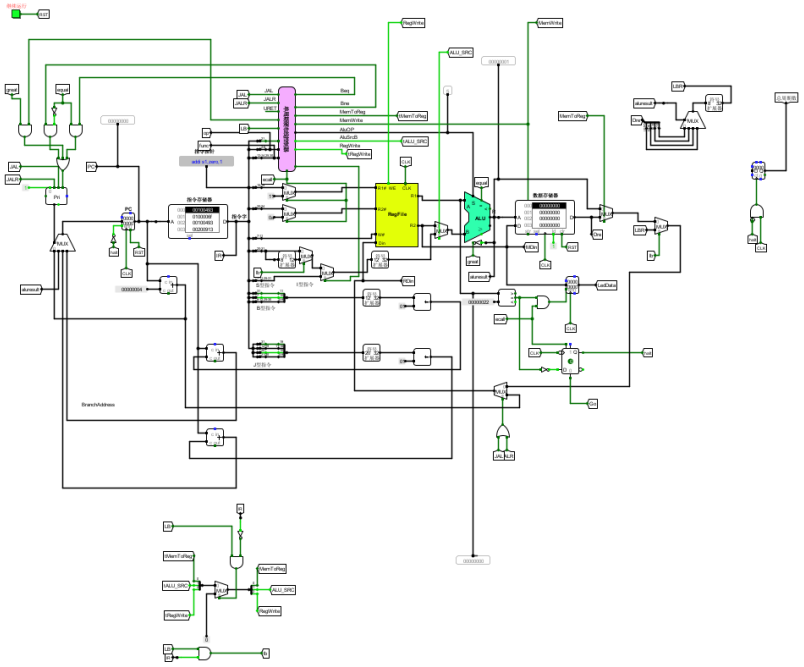


图 3-4 数据通路

3.1.3 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 上进行主控制器的具体实现。

① Logisim 实现

可以通过代码逻辑自动生成，在这里不进行赘述。

② 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，故只取生成信号的整体逻辑图演示如下图 3.5 所示。

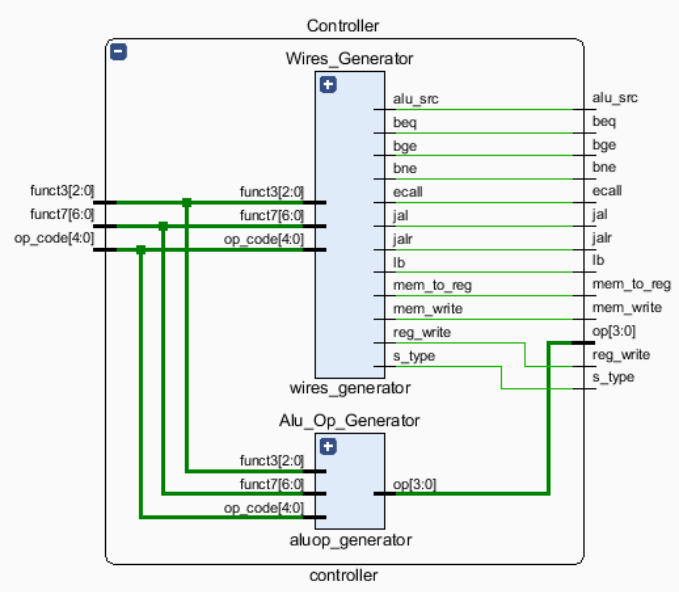


图 3.5 主控制器原理图

3.1.4 单周期 CPU 上板

根据上面的实现原理，重写 Verilog 代码，实现单周期 CPU 上板，上板的原理图如图 3-6 所示。

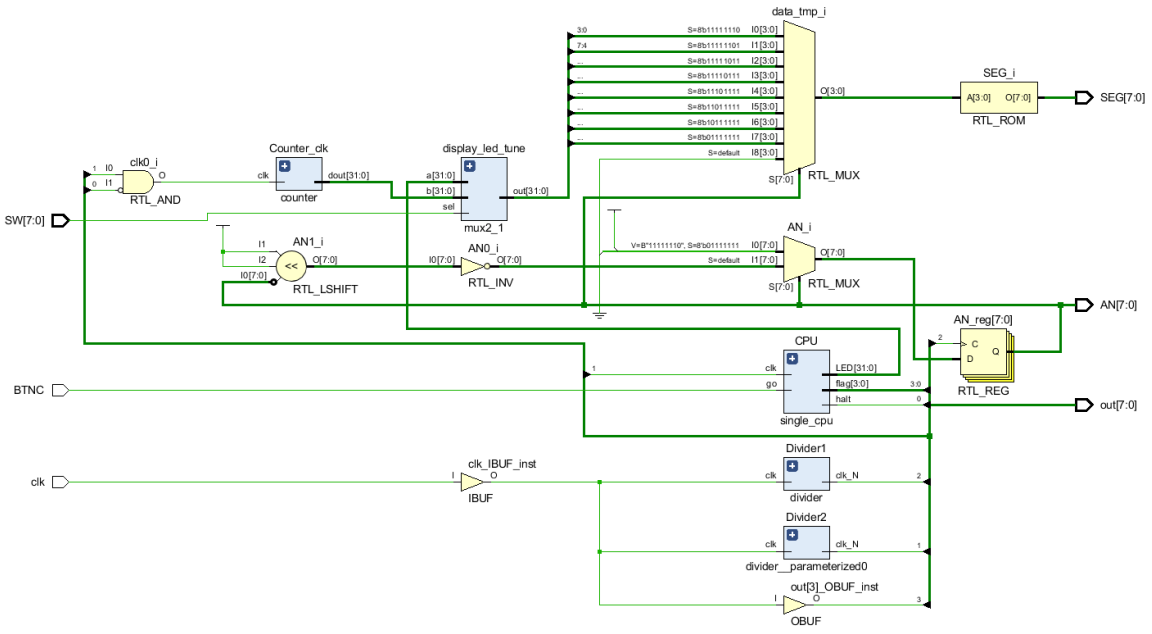


图 3-6 单周期上板 Verilog 原理

3.2 中断机制实现

3.2.1 单级中断

需要实现中断控制器，在产生中断的下一个上升沿输出中断请求与中断号， 这就需要两个寄存器，第一个寄存器在中断源产生脉冲时立刻载入 1，第二个中断再 下一个上升沿时载入 1 并情况第一个寄存器，中断号借助优先编码器输出，中断信号通过三个寄存器的或运算得到，具体中断控制逻辑如图 3-7 所示。

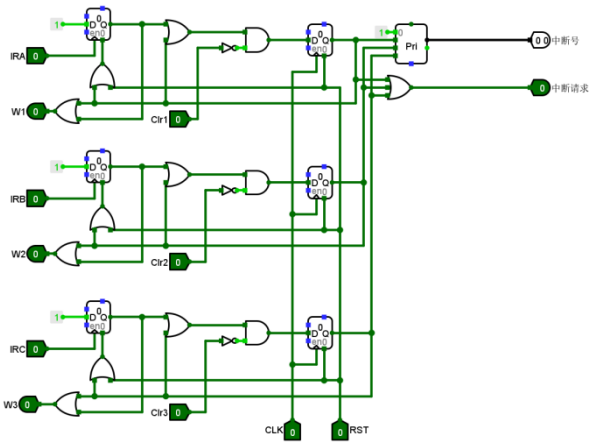


图 3-7 中断控制逻辑

硬件实现上需要通过多路选择器生成中断入口逻辑，中断源地址可以通过 RARS 仿真器得到具体地址，通过使用 EPC 和 IE 寄存器来保存当前中断的状态和返回地址，通过 URET 和中断信号决定生成中断清零信号，清空中断请求队列。具体的硬件实现如图 3-8 所示。

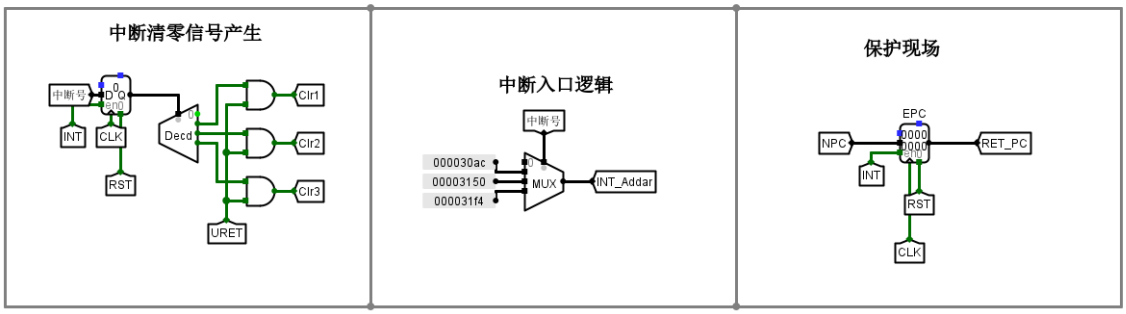


图 3-8 中断信号产生、中断入口逻辑和保护现场

中断处理同时也需要更改部分数据通路以实现中断服务，具体如图 3-9 所示，主要更改的便是 PC 寄存器相关的数据通路，具体如所示。其中 NPC 代表原数据通路中

的 PC 输入，N_PC 代表支持中断服务后的 PC 输入。

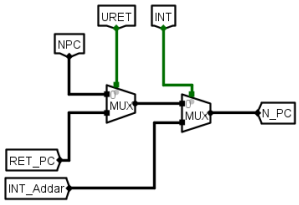


图 3-9 支持中断服务的数据通路更改

3.2.2 多级中断

多级中断中利用硬件保存的实现方法使用三套 EPC 和 IR 寄存器来保存中断号和返回地址，中断控制逻辑与单周期相同，并同时采用 CSRRCI 和 CSRRSI 来控制中断使能信号，具体而言是利用这两条指令来控制 IE 寄存器的信号，IE 寄存器初始为 1，高电平表示开中断。控制逻辑如图 3-10 所示。

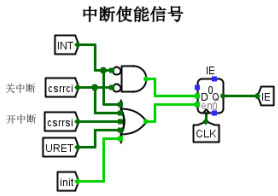


图 3-10 中断使能信号发生逻辑

硬件实现上生成中断入口逻辑方式和生成中断清零信号的方式与单级中断相同，也同样通过使用 EPC 和 IE 寄存器来保存当前中断的状态和返回地址，但是由于中断可以嵌套，所以需要多个 EPC 和 IE 寄存器进行中断保护。同时由于中断存在优先级，因此需要中断比较逻辑确定当前需要执行的中断服务，与单级中断差异化的硬件实现如图 3-11 所示。

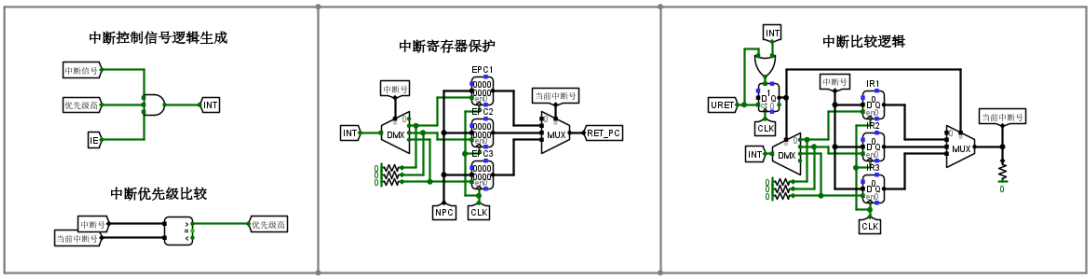


图 3-11 中断寄存器保护、中断优先比较、中断优先逻辑

中断处理更改部分数据通路以实现中断服务与单级中断相同，在这里不再赘述。

3.3 流水 CPU 实现

3.3.1 流水接口部件实现

流水接口部件所要传递的信号已经在上文列出，由此我们可以设计 IF/ID、ID/EX、EX/MEM、MEM/WB 四个流水寄存器，以 MEM/WB 为例，各个信号通过寄存器存储，各个寄存器上升沿触发，RST 信号可以同步复位，如图 3-12 所示。

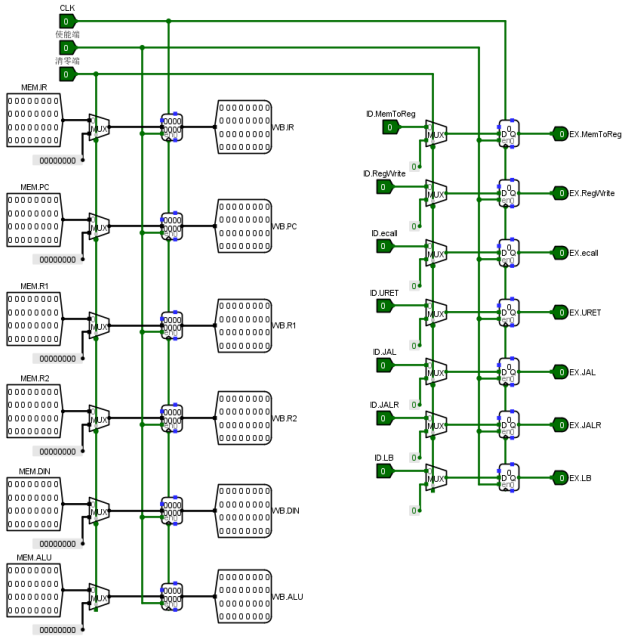


图 3-12 流水接口部件

3.3.2 理想流水线实现

理想流水线是在单周期 CPU 的基础上，将各部件根据五段的分法分成五个部分，每个部分使用流水线接口进行拼接。主要模块的分布为：（1）IF 段：程序计数器 PC、指令存储器 IM；（2）ID 段：寄存器堆 RF；（3）EX 段：运算器 ALU；（4）MEM 段：数据存储器 MEM；（5）WB 段：写回数据。具体的理想流水线如图 3-13 所示。

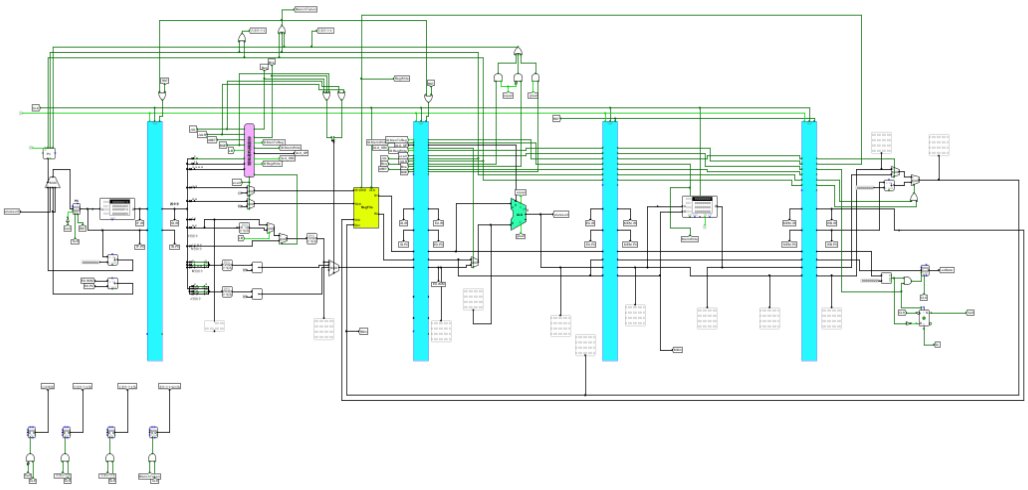


图 3-13 理想流水线的结构图

3.4 气泡式流水线实现

气泡流水线就是在理想流水线的基础之上，根据当前电路的状态（RS1 和 RS2 是否被使用，WriteReg、RegWrite）增加 DataHazzard 逻辑判断，其具体组合电路如图 3.14 所示。

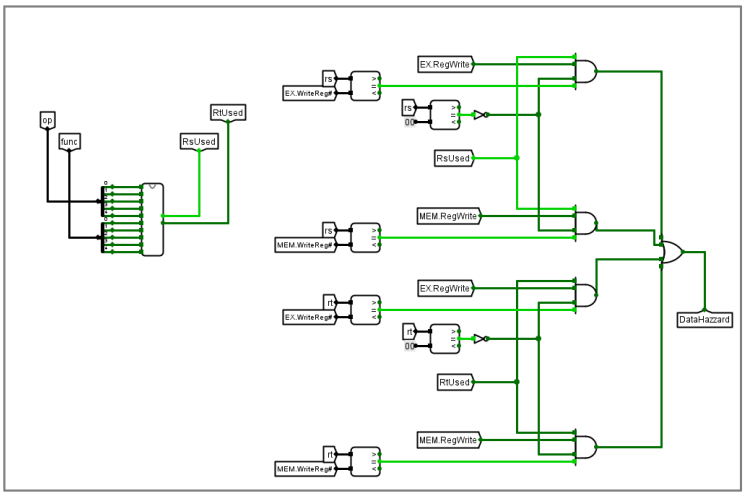
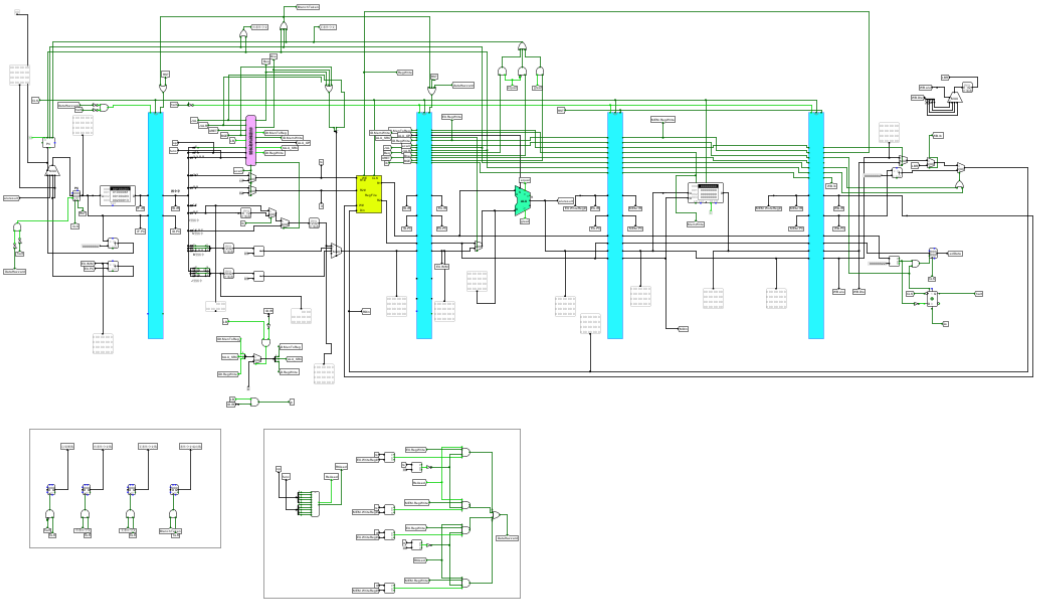


图 3-14 DataHazzard 组合逻辑

具体的气泡流水线如所示。



3.5 数据转发流水线实现

重定向流水线（数据转发流水线）就是在理想流水线的基础之上，根据当前电路的状态（RS1 和 RS2 是否被使用，WriteReg、RegWrite）增加 Load_Use、R1_F、R2_F 逻辑判断，其具体组合电路如图 3-15 所示。

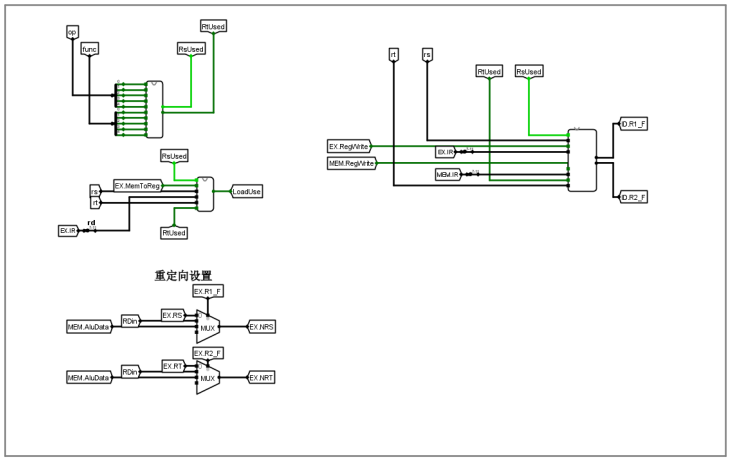


图 3-15 Load_Use、R1_F、R2_F 逻辑电路

3.6 动态分支预测机制实现

动态分支预测需要在重定向流水线的基础上增补 BHT 表，利用寄存器设计 8 位 Cache 槽，实现一个有限状态机用于支持分支预测历史位的状态转移，状态机的转移过程，BHT 表的设计实现如图 3-16 所示。

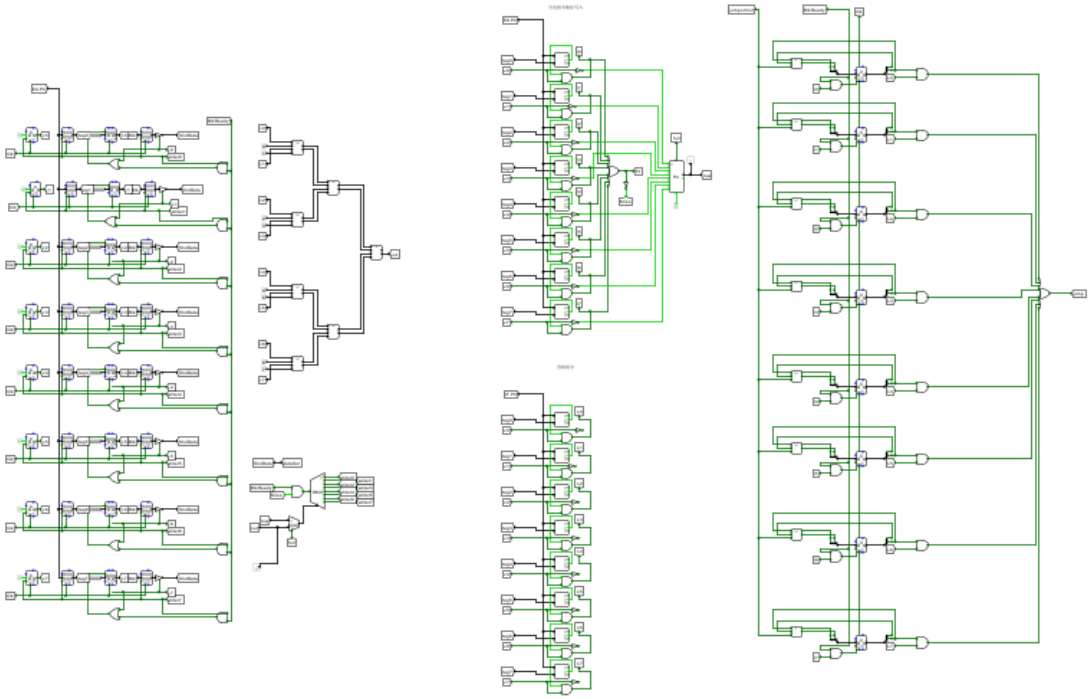


图 3-16 BTH 表的结构图

与此同时，PC 端的入口逻辑也要进行大量的修改，通过分支预测的正确性和预测结果进行多路选择，以确保实现动态分支预测的入口地址修改和纠错等，需要同时修改流水器接口，增加分支预测的判断结果，和 EX 段的真实分支预测结果进行比较，确定转换逻辑是否正确，修改流水器接口的方法比较简单，在这里不列出其具体的修改结构图。PC 端入口具体逻辑如图 3-17 所示。

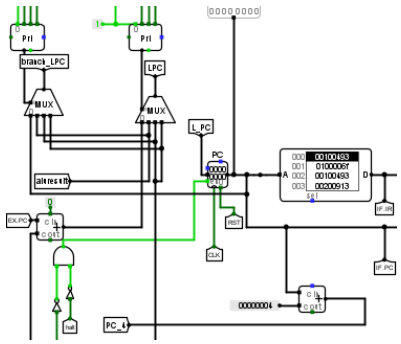


图 3-17 PC 端入口具体逻辑结构图

3.7 流水中断机制实现

在重定向流水线的基础上添加多级中断机制，相关实现方法在上文中已经描述过，下面给出实现的结果如图 3-18 所示。

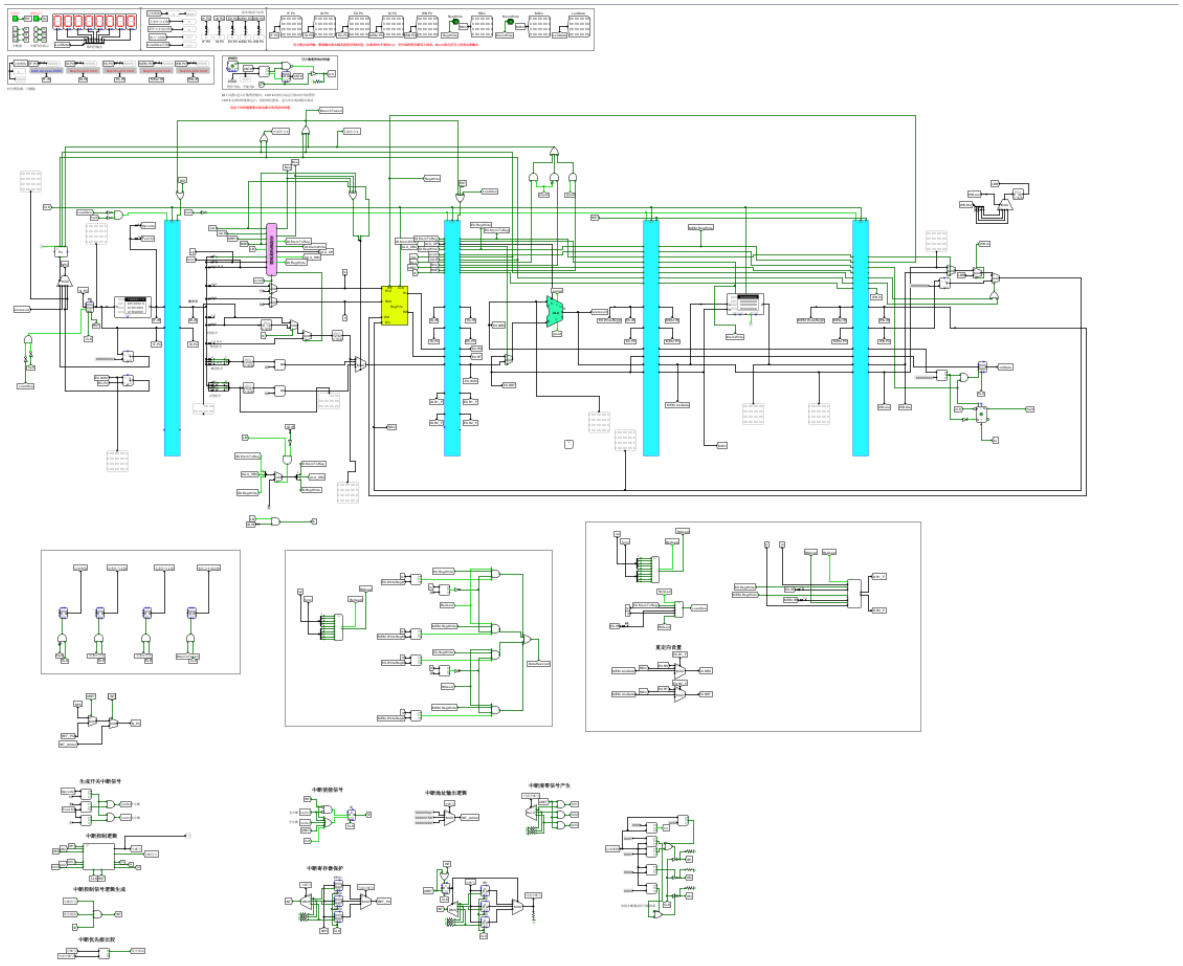


图 3-18 流水多级中断

可以保证执行结果和单周期多级中断相同。

4 实验过程与调试

4.1 测试用例和功能测试

本次实验中，单周期 CPU、气泡流水线 CPU、重定向流水线 CPU、单级和多级中断均在头歌平台通过仿真测试。如图 4-1 所示。

关卡	任务名称	开启时间	代码修改行数	评测次数	完成时间	实训耗时	是否查看答案	经验值	关卡得分	调分
1	RISC-V单周期CPU(24条指令)	2023-08-29 15:58	13666	11	2023-08-31 14:27	1小时 7分钟 0秒	否	1000/1000	11.11/11.11	11.11
2	理想流水线设计	2023-08-29 16:06	21458	6	2023-09-01 18:14	1小时 55分钟 51秒	否	300/300	11.11/11.11	11.11
3	气泡流水线设计(EX段分支3624版本)	2023-08-29 16:07	70677	10	2023-09-03 16:46	49分钟 41秒	否	800/800	11.11/11.11	11.11
4	重定向流水线(EX段分支2298版本)	2023-08-29 16:07	39650	21	2023-09-05 10:23	1小时 5分钟 23秒	否	1000/1000	11.11/11.11	11.11
5	重定向流水线设计(ID段分支2103版本)	2023-08-29 16:07	- -	1	--	--	否	0/1000	0.00/11.11	0.00
6	气泡流水线设计(ID段分支3309版本)	2023-08-29 16:07	- -	0	--	--	否	0/700	0.00/11.11	0.00
7	单周期RISC-V+单级中断	2023-08-29 16:07	37338	6	2023-09-05 23:06	1分钟 11秒	否	700/700	11.11/11.11	11.11
8	多级嵌套中断(EPC硬件堆栈保存)	2023-08-29 16:07	40976	39	2023-09-06 17:33	36分钟 43秒	否	1000/1000	11.11/11.11	11.11
9	多级嵌套中断(EPC内存堆栈保存)	2023-08-29 16:07	- -	1	--	--	否	0/1000	0.00/11.12	0.00

图 4-1 评测记录

具体测试用例有 Benchmark_CCAB.asm 和中断测试.asm 两个测试文件。

4.1.1 测试用例 1 Benchmark_CCAB.asm

以下为 Benchmark_CCAB.asm 在单周期、气泡流水线、重定向流水线、动态分支预测电路的运行结果：

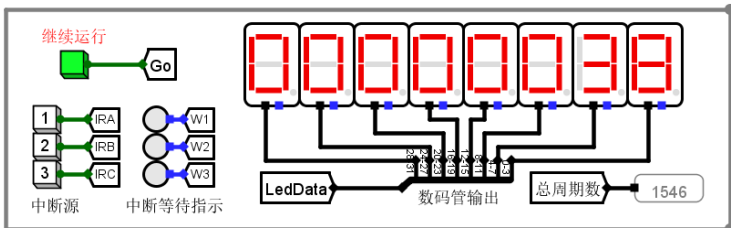


图 4-2 单周期 CPU 运行最终结果

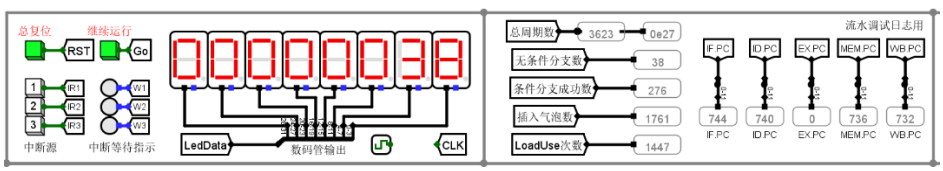


图 4-3 气泡流水线 CPU 运行最终结果

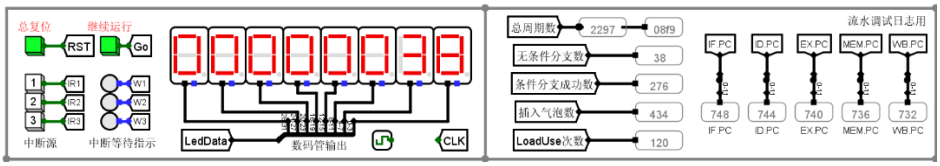


图 4-4 重定向流水线 CPU 运行最终结果

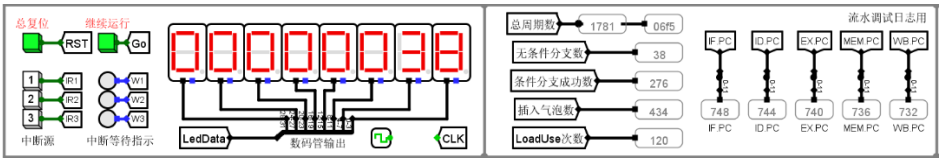


图 4-5 动态分支预测重定向流水线 CPU 运行最终结果

以上各运行结果均与实际结果一致。

4.1.2 测试用例 2 中断测试.asm 测试

对于三种中断，对于中断按键操控的中断源均能在开中断时及时响应，并且在当前中断操作结束完成后及时响应中断请求队列中级别最高的中断请求，并且通过多次中断响应的测试。

4.2 性能分析

在单周期流水线 CPU 中，执行指令条数为 1546，但是每周期长度较长。

对于多周期的流水线，仅估算起见，每个周期长度可认为约为单周期 CPU 的周期长度的 1/5。对于气泡流水线，需要执行 3624 周期，相较于单周期流水线，时间缩短一半。对于重定向流水线，所需执行周期数为 2297，比气泡流水线又减少 1/3。对于动态分支预测，其周期数仅 1781，在重定向流水线的基础之上又减少了 1/4。

动态分支预测执行相同的代码所需的时间周期仅为单周期 CPU 的 1/5，CPU 执行效率得到了空前的提升。

4.3 主要故障与调试

4.3.1 中断返回故障

单周期多级中断： 中断返回问题。

故障现象：在执行多级中断服务时，对于新来的中断服务程序处理后找不到原来的中断服务程序继续执行。

原因分析：在多级中断服务的中断号确认时，没有能够正确保存中断号，导致处理过新的中断服务程序后，没有将中断号进行正确的找回，保存好的现场也就没有正确恢复。

解决方案：增添中断号保存寄存器，将中断号进行保存，在处理过新的中断服务程序后找回正确的中断号继续执行。

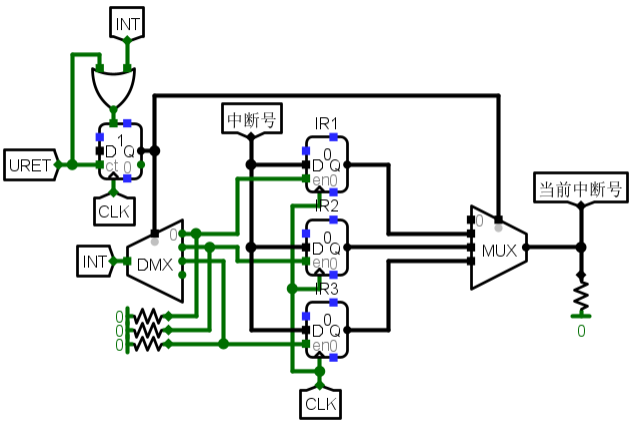


图 4-6 中断保存中断号结果图

4.3.2 LB 指令故障

单周期 CPU 增加支持 CCAB 指令：实现 LB 指令。

故障现象：LB 指令设计需要进行字访问，字读入，需要大幅度更改数据通路，在过程中出现取字不成功，字扩展结果不正确等现象，同时由于 LB 指令与 nop 指令非常接近，会产生混淆。

原因分析：取字时需要采取的方案是先取出整个字节，然后再设计取出，同时可以借助指令本身的特性（nop 指令为全 0），进行进一步区分。

解决方案：使用多路选择器对于字节进行进一步的取出。具体结构图如图 4-7 所示。

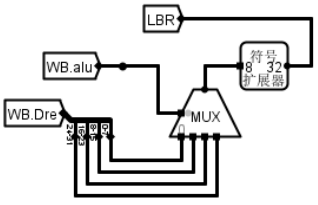


图 4-7 LB 指令取字数据通路

增添对于 LB 指令进行区分的数据通路，更改包括 LB、RegWrite、MemtoReg 等四个信号，用于区分 LB 指令和 Nop 指令，具体的数据通路如图 4-8 所示。

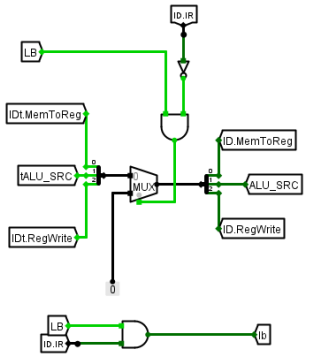


图 4-8 LB 数据通路

4.3.3 动态分支预测故障

实现动态分支预测：预测正确后后续指令未执行。

故障现象：动态分支预测结果发现自己错误后没有能够正确跳转回应该执行的指令。

原因分析：分析地址计算模块发现代码编写错误。分支预测后跳转的指令应该是由多种情况组合而成的，同时需要考虑预测后正确或者错误对应的不同情况，对于没有进行正确跳转的指令逐步调试发现，没有正确理解跳转的含义，在 EX 段预测错误的时候，应当选择 EX 段的 PC+4 后才能继续正确执行指令。

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。

4.4 实验进度

表 4.1 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 RISC-V 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。

时间	进度
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。

5 团队任务

5.1 简介

在本次课设的小组任务中，我们在 Nexys 4 开发板上实现了手写大写英文字母识别。用户可以在展示系统中上传一张含有大写英文字母的图片。图片经由运行在 PC 端上的展示系统转换成 28 * 28 浮点数矩阵后使用 UART 协议输入开发板。数据输入开发板后，部署在 CPU 上的三层感知机算法（下称 MLP）会计算出对应字母的 ASCII 码并传回展示系统，由展示系统展示结果。我们的系统结构图如图 5-1 所示。

整个任务可以分成 4 个模块：支持 RISC-V IMF 全指令集单周期多级中断的 CPU、遵循 UART 协议的数据传输功能、手写大写字母识别算法以及展示系统。下面将一一介绍这四个模块的设计和实现。

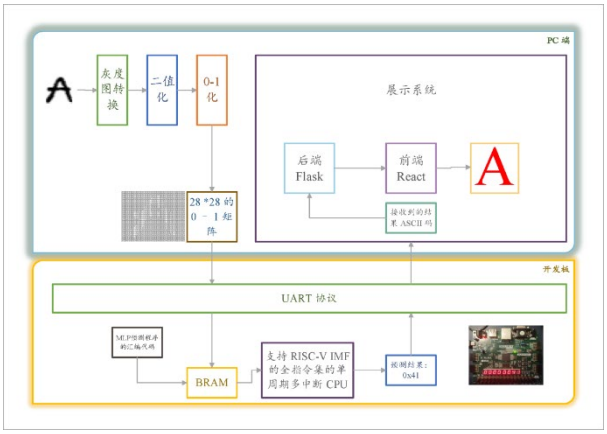


图 5-1 系统结果图

5.2 分工

我们小组由 4 名队员组成，各成员的分工如下（按姓氏拼音首字母排序）

- 崔昊阳：
  - 班级 / 学号：CS2104 / U202115415
  - 分工：MLP 的训练、预测代码的编写、编译和上板部署；讲解 PPT；管理任务的总体进度并主持小组会议。
- 王佳欣
  - 班级 / 学号：CS2102 / U202115378

- 分工：支持 RISC-V IMF 全指令集的单周期多级中断 CPU 的设计与实现；系统联调时的 Bug 解决。

- 杨明欣
  - 班级 / 学号：CS2106 / U202115514
  - 分工：展示系统的设计与实现；理清项目流程；制作展示 PPT
- 周汝凡
  - 班级 / 学号：CS2104 / U202115454
  - 分工：遵循 UART 协议的 FPGA 数据传输功能的设计与实现；系统联调时的 Bug 解决。

5.3 模块设计和实现

5.3.1 支持 RISC-V IMF 全指令集的单周期多级中断 CPU

本次课设我们在个人任务实现的支持 24+4 条指令的单周期 RISCV CPU 的基础上，对 CPU 进行功能扩展，用 verilog 语言编写实现了基于 FPGA 开发板的支持 RISC-V IMF 全指令集的单周期多级中断 CPU ，CPU 整体架构如图 5-2 所示。我们主要增加了浮点数相关的浮点数寄存器与运算器模块，修改了硬布线控制器部分控制信号生成逻辑，增加了 IMF 扩展指令集相关数据通路。

浮点数部分，其寄存器的实现与整数寄存器类似，在此不作赘述。同时我们设计并实现了 FPU 浮点数运算器，支持 IEEE754 浮点数的有符号和无符号加减乘除运算，以及部分特殊浮点数运算指令的结果输出。浮点数运算的具体实现逻辑是将每个 32 位 float 浮点数划分为符号码、阶码和尾码，依据 IEEE754 浮点数的运算标准，进行对阶、移位、舍入等操作，最终获得运算结果。与 ALU 类似，我们将加减乘除各模块进行封装和调用，并为浮点数运算设计了五位的运算方式控制信号，据此选择 FPU 的输出，另外为我们为浮点数运算增加了控制信号 FMemtoReg, FMemWrite, FRegWrite, itof 和 ftoi，完成了浮点数存取、浮点数与整数间的转换的数据通路控制。

IM 指令扩展部分，我们进一步对硬布线控制器进行扩展，增加整数乘法和其他扩展指令的控制信号生成逻辑，并相应地增加相关数据通路控制，实现数据的存取、运算等，完成扩展指令功能。同时该 CPU 支持多级嵌套中断处理机制。尽管本次演示系统中没有使用到相关功能，但在汇编代码中引入中断处理代码后，即可通过

FPGA 板上的按钮进入中断处理程序，进行多级嵌套中断的处理。

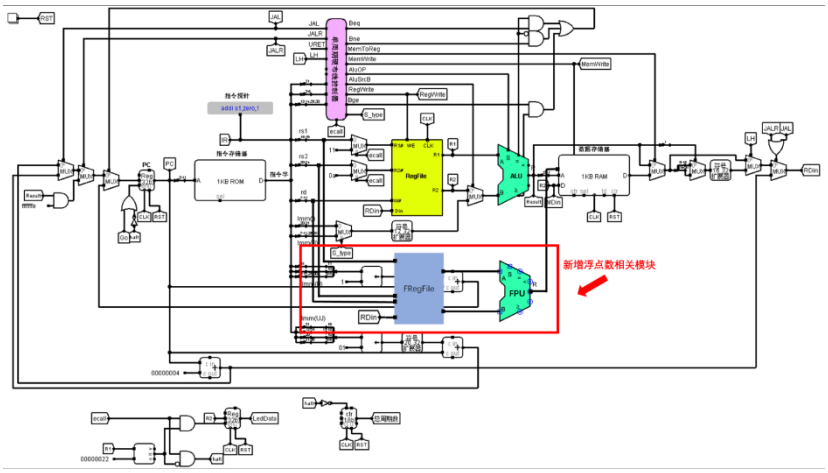


图 5-2 CPU 结构图

5.3.2 UART 通信协议实现展示系统和开发板之间的数据传输

在已实现的 CPU 的基础上，我们还需要实现输入图片的信息 InputData 和输出预测结果 Res 的传输。其中 InputData 由展示系统传输到开发板中，Res 由开发板传回展示系统。在本实验中，我们使用 UART 通信协议来实现这一过程。在具体实现中，与 UART 协议相关的模块有 4 个，分别是：用于（开发板）接收一个字节的模块 UART_Receiver、（开发板）接受指定数量字节的模块 Data_Receiver、（开发板）发送一个字节的模块 UART_Transmitter 和统筹接收和发送指定数量字节数据的顶层模块 UART_Top（前期实现的 CPU 也包含在这一模块中）。

所有模块的实现都采用了有限状态机(FSM)来控制收发数据的状态转移过程。在此我仅解释模块 UART_Top 的 FSM，其余模块的实现与之类似。总体上，收发数据包括了 7 个状态，覆盖了 3 个主要过程：接收数据状态(IDLE)，CPU 计算状态(CPUEn,StartCPU,WaitCPU)，传回结果状态(TxSendEn,StartSend,WaitSend)。当系统复位信号 i_rst_n=1 时，系统状态为 IDLE，此时开始接收指定数量字节的数据；接收完毕时信号 receive_done=1，由此转移到 CPUEn 状态；此时将接收的数据 receivedata 输入 CPU，转移到 StartCPU 状态；此时启动 CPU 信号 Rst_CPU=0(也就是 CPU 输入中的全局复位信号)，转移到 WaitCPU 状态；CPU 开始计算预测结果；当计算完成时信号 CPU_Done=1，转移到 TxSendEn 状态；此时将预测结果 Res 输入传输模块，接着转移到 StartSend 状态；这一状态主要完成控制每次传输 Res 的哪一部分字节（因为一次只能传输一个字节），转到 WaitSend 状态；CPU 发回数据，发送完毕

后回到 StartSend 状态，再次传输下一个字节，由此在这两个状态之间来回切换，当 Res 的全部字节传输完毕后回到 IDLE 状态等待下一次接收数据。更为具体的状态转移过程如图 5-3 所示。

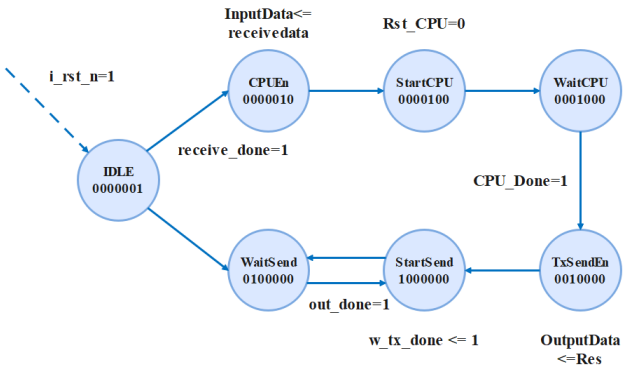


图 5-3 URAT 通信协议

5.3.3 手写大写字母识别算法的设计和实现

在本次课设中，我们采用了多层感知机（MLP）来实现手写大写字母的识别。MLP 是一种前馈神经网络，由多个层级组成，至少包含一个输入层、一个或多个隐藏层和一个输出层。输入层接受原始数据作为输入信号，隐藏层和输出层的神经元通过学习逐渐提取和组合数据特征，并生成最终的输出结果。经过在准确率大小和 Nexys 4 开发板的算力和主存容量之间的权衡，我们最终使用了 3 层 MLP，隐藏层向量长度为 100。

在设计和实现过程中，我们遇到了难以手工编写汇编代码实现 MLP、开发板主存容量不足以存储参数等问题。对于难以手工编写汇编代码实现 MLP 的问题，我们将汇编 MLP 的实现过程分成训练、预测和部署三个阶段。在训练阶段，我们使用 Python 编写了训练代码并在 PC 端上进行快速训练以得到模型参数。在预测阶段，我们使用 C 语言编写了 MLP 的测试代码并进行了调试。在部署阶段，我们使用 riscv-none-embed-gcc 编译器将预测代码编译成汇编语言。将汇编代码进行修改和加入输入输出后使用 rars 工具汇编成 hex 格式的机器指令部署上板。对于开发板主存容量不足以存储参数的问题，我们一方面在几乎不影响准确率的情况下，将隐藏层向量长度减少，以减少模型参数的大小；另一方面利用开发板提供的 BRAM 来提高 RAM 的容量上限。

5.3.4 前后端展示界面

我们构建了一个 WEB 程序作为展示系统。我们通过前端 react 框架，后端 flask 框架构建了一个简单的展示界面，展示界面包括主页和导航页。

主页主要包括系统的简单介绍和优势展现，如图 5-4 所示。

图 5-4 主页 1

导航页则实现了具体的预测逻辑。用户上传图片后，系统调用后端逻辑将图片进行二值化处理，通过通信协议传入开发板中进行预测，最后向前端返回预测结果。同时，前端还返回了上传的图片和二值化处理后的图片矩阵，便于观察。导航页如图 5-5 所示。

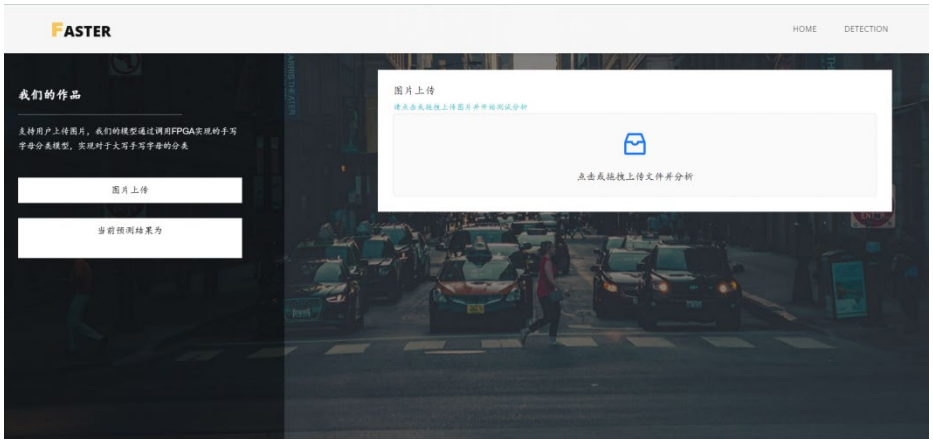


图 5-5 导航页

5.4 结果展示

下面我们进行简要的结果展示。首先，我们上传一张含有大写字母 A 的图片如

图 5-6 所示。

图 5-6 A 图片

上传图片过后，经过计算，FPGA 开发板返回的结果如图 5-7 所示，可以看到计算结果为“A”的 ASCII 码。

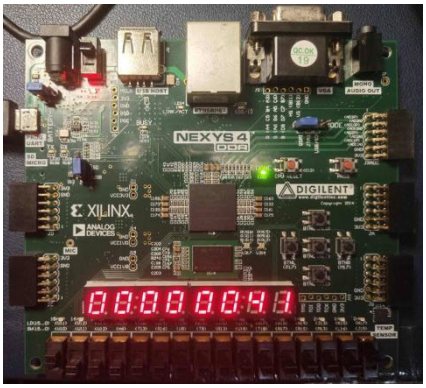


图 5-7 FPGA 预测结果

同时，展示系统也正确返回结果，如图 5-8 所示，预测成功。

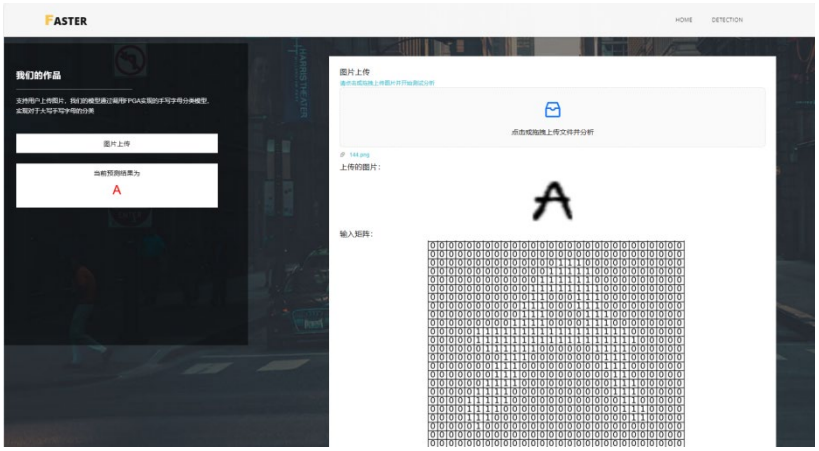


图 5-8 前端返回预测结果

5.5 调试过程和错误解决

（1）UART 传输多字节数据出错

这一步出现的问题主要是无法同时控制单字节的传输和多字节的传输字节选择，实际情况表现为只能接受第一个传输的字节。采用的解决办法是将这两个部分分为两个不同的状态 StartSend 和 WaitSend，并加入新的控制信号 out_done 来实现状态转移，修改后状态的转移过程在 5.3.2 节中已提到，最终实现了传回任意指定数量字节

的功能。

（2）汇编代码执行时出栈入栈出错

对 CPU 进行仿真调试的过程中，我们发现浮点数运算指令可能从未初始化的 RAM 中读取到错误内容，导致后续 CPU 运算出错。经 RARS 模拟检查发现，汇编指令默认整数和浮点数共用一个栈空间，原本整数和浮点数存储器分离的设计导致整数和浮点数的栈空间同样分离，运算时可能出现错误的数据获取。因此我们将浮点数和整数的存储器合并，输入 MemWrite 和 FMemWrite 分别控制整数和浮点数的存储器写入同一栈空间，保证 CPU 的正确运行。

6 设计总结与心得

6.1 课设总结

在本次课程设计中，我完成了如下任务：

1） 基础任务：设计了单周期 CPU、理想流水线 CPU、气泡流水线 CPU、重定向流水线 CPU、支持单级中断和多级中断的单周期 CPU，支持单级中断的流水线 CPU，以及动态分支预测的重定向流水线 CPU。

2） 功能总结：所有的 CPU 除理想流水线外，均支持 24 条基本指令和 4 条扩展 CCAB 指令。功能和效率均达到最佳结果。

3） 团队任务：完成团队任务相关部分，实现了基于自制 CPU 通过 verilog 语言上 FPGA 开发板实现了手写字母识别系统。

6.2 课设心得

本次课程设计可以说是迄今为止做过的最有难度、需要查询的资料最多，需要自己学习的部分最久的一个课程实验，无论是基础任务需要独立设计完成 CPU，同时又要从简到难实现流水线 CPU。现在再来回顾整个课程设计的整个过程，确实收获很大。

课程设计刚刚开始的时候，还没有意识到整个实验的难度，因为有上学期但单总线 CPU 的设计基础，在设计单周期 CPU 时，没有感觉到很大的障碍，所以整个过程还算比较迅速。然而，在完成流水线 CPU 的过程中，我可谓是受尽了折磨，因为自身比较粗心的性格，在画流水接口时，由于几个流水接口比较相似，需要进行复制，因此在过程中常常出现差了一根线没有连，导致要单步调试很久，才能确认错误。在这个实验过后，我一定要努力让自己变得更细心。同时在流水线的动态分支预测部分也很有难度，虽然教材上有讲解动态分支预测的部分，但是如果不细心看很容易弄错，我在设计动态分支预测时遇到的最大的问题就是地址始终跳转的不对，卡到死循环，通过多次仔细阅读书籍，查询资料，我最终才将其改正，整个过程让我体会到了什么叫细心一点事半功倍，粗心一点事倍功半的效果。

紧接着，单周期上板的过程我也遇到了很多问题，其中最大的问题就是行为仿真

通过了但是上板最终错误，真的是很考验我的耐心，最终我的单周期上板 LB 指令不知道为什么仿真通过了，最终上板还是没有结果。同时，通过本次的团队任务，我也深刻感受到了团队力量的强大，我们最终在基于 FPGA 的开发板上实现了手写字母识别系统，同时提供了简单美观的前端展示界面，比较完美地完成了任务，同时也提高了自己的能力。

然而对于本次课程设计，我还有一些小小的建议和改进。本次课程设计在设计单周期 CPU 的时候也可以提高一下模型的封装程度，设计一下按模块完成任务，通过保证每一模块的检查正确，可以有效帮助我们排查错误，通过检查的模块可以确保正确，我们在调试的时候就可以不用过于关注这个模块，不然我个人觉得在整个实验的过程中，系统调试的过程远大于模块设计与实践的时间。

最后在这里也感谢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] 谭志虎，秦磊华，吴非，肖亮.计算机组成原理. 北京：人民邮电出版社，2021 年.

[4] 谭志虎，周军龙，肖亮. 计算机组成原理实验指导与习题解析.北京：人民邮电出版社，2022.

[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：杨明欣