基于 logisim 的 32 位多周期微程序控制 MIPSCPU 数据通路和微程序控制器设计

学　　号	E12214052	专　　业	计算机科学与技术	姓　　名	赵宸宇
实验日期	2024 年 9 月 10 日	教师签字	徐晨初	成　　绩	

摘要

本次实验设计和实现一个 **32 位多周期微程序控制**的 MIPS CPU，为学生深入理解 MIPS CPU 的设计原理和实现细节提供了良好的实验平台。

该实验的主要内容

1. 实验**准备**
2. **数据通路**设计
3. **控制器**设计
4. 8 条指令的 **ISA 调通和联调**
5. 实验**总结与改进**

该实验的主要产出

1. MarkDown 上课**笔记**和慕课线上**自学笔记**，请见附件
2. 完整的**实验报告**材料，包括 **LaTeX 文档**和生成的 **PDF** 文档及其附件，请见**附件和 git 仓库**
3. 该实验过程的 **git 仓库**，包括 circ,xlsx 等支持材料的变更记录（每期实验延期一周发布）；位置：`https://gitee.com/cslearnerer/AHU-CSHT`

目录

一、【实验目的】

1.1 几大主要任务

本次实验需要学生完成一个 MIPsCPU 设计。这 CPU 是指 **32 位 MIPsCPU**，采用**多周期微程序**控制机制。具体来说，该设计主要包含以下几个方面：

1. 首先，**通过 mooc** 对实验设计的具体概念**进行预习**，熟悉实验环境。
2. 然后，需要**设计数据通路**，这是该 MIPsCPU 得以运行的基础框架，也是该后面设计所需要依赖的基础。
3. 再者，需要**完成控制器的设计**。由于本次实验采用的硬件实现是多周期＆微指令的，所以需要先后完成：1. **指令译码逻辑**；2. 微指令**存储器**和微程序**地址转移**逻辑（指令翻译逻辑）；3.**ALU 控制**器逻辑。
4. 最后，需要完成的就是 **ISA 的调通**，所谓 ISA 的调通和联调就是指本次实验涉及的 **8 条指令**的调试成功，以及`sort.hex`**程序的调试成功**。只有这两个方面都成功了，才

能从软件层面说明本次设计的 CPU 基本可以运行。

5. 作为一个附加项目，再完成 CPU 的设计工作后，接下来就可以对 CPU 的设计**进行完善**。例如**优化数据通路**的呈现模式、**优化微指令存储器**的内容设计等等。这些工作大多处于技巧性层面，故而可以通过**逐步渗透到后面的所有实验中迭代优化工作**。
6. 疑难解答

二、【实验原理】

本次实验的实验原理有：

1. 首先是**数字逻辑电路和 logisim**，数字逻辑电路是本次实验得以进行的关键，是本次实验的底层知识体系。而 logisim 则负责在软件层面实现＆模拟＆仿真这个设计。虚拟实验平台 logisim 为本次实验提供了良好的支持，有利于学生理解实验设计，加速设计。
2. 然后是 **MIPs32 位 CPU** 的相关内容，包括**指令集架构、硬件实现、微程序设计等**。对于各个方面，主要的参考内容有百科页面、MIPs 指令集手册等等。需要注意的是，MIPsCPU 在底层设计和实现上和 x86CPU 有着很大的不同，例如 reg 的数量，例如访存的习惯，例如指令的长度和定长特性等。
3. 最后，就是 MIPs 指令集下的**汇编知识**，这里主要通过迁移前面所学的 MIPs 指令集基础知识和 x86 调试经验来完成这方面的理解和联调运行工作。
4. 其他方面，还有**硬布线和微程序控制的区别**，这里我选择了后者，这造成的影响将在下文展开谈论。

三、【实验内容】

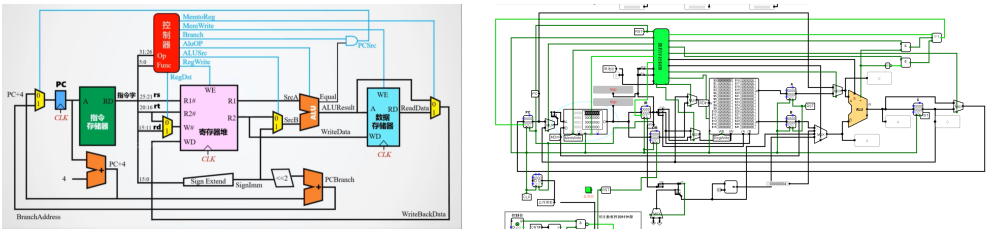
接下来，本文逐步骤讲解实验方案和本次学生完成实验的主要流程。

根据上文【实验目的】提出的实验流程，这里首先需要**完成数据通路的设计工作**。

3.1 数据通路设计

对于**数据通路**，本文参考了 HUST 提出的 MIPsCPU 数据通路设计原理图，如下图所示。对于该原理图，本文参考其设计原理完成了如右图所示的数据通路设计。对于原理图没有提及的如 PC 位宽等等的问题，本文采用了一定的优化方案来辅助实现。例如，本文的 PC 采用字节地址存储，当需要使用 PC 访存时，会有专门电路（PC 右侧的分线机构）对 PC 进行右移操作，同时截断出 MM 所需的 16 位地址位宽。

其次，本文对电路细节做了优化，例如：本文使用了 cs3410 提供的寄存器堆以方便修改和观察 CPU 运行情况；本文采用了一系列探针机构来显示各个锁存寄存器的内容



(a) 数据通路设计原理图　　**(b)** 数据通路具体实现

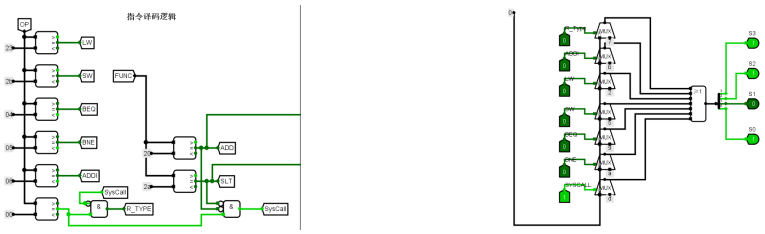
图 **1**　数据通路

进而方便调试；本文还对鼠标的操作模式进行了微调，通过将常用按键绑定到鼠标侧键和上下滚轮键，将本地的鼠标进化为了一个 logisim 专用的调试器。这在之后的调试过程中极大地加速了调试进度。

鉴于是初次实现数据通路，本次实验的数据通路也有不足的地方，其中一个主要点就是：略显凌乱，这将成为今后实验设计技巧渗透中的一大改进方向。

3.2 控制器的设计

如下组图所示，本文使用了一种较简单的方式即选择-或门的方式来完成指令译码逻辑和地址转移逻辑的设计工作。其次对于微程序控制器设计，通过设计—联调——设



(a) 指令译码逻辑　　**(b)** 地址转移逻辑

图 **2**　地址逻辑电路图

计反复迭代流程，本文得到的目前可以稳定运行的一种设计方案如右图所示。这种设计方案对于微命令的设计参考了 HUST 提出的设计原理图，如下图所示。本文结合此设计原理图，同时还对微程序设计进行了优化。

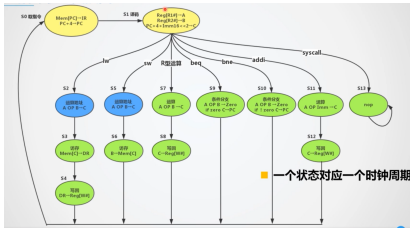
对于 ALU 控制逻辑，这里使用选择器配合指令来完成设计工作。鉴于本次所需要实现的指令较少，这才得以使用这一种设计方案，今后在设计大批量指令时，可能需要设计地址转移逻辑那样的专门电路，来优化 ALU 控制器的设计工作。

3.3 ISA 调通-联调

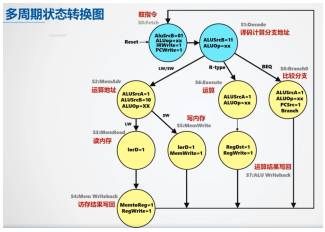
最后就是完成 8 指令调通和 sort 程序联调。

这里我转回到多周期微程序 MIPsCPU 的顶层视图。在这里主要用到的操作有：

1. 使用总复位软按键和鼠标 DPI 键（见上文）进行复位和单步调试工作；



(a) 状态转移图



(b) 状态转移—数据通路图

微指令功能	状态	微指令地址	数据到指令	PcSrc	AluA	AluB	MToR	RDst	写IR	写PC	RWrite	MWrite	MRead	BEQ	BNE	Alu-op	P	下址	微指令	十六进制
取指令	0	0000	0	0	0	01	0	0	1	1	0	0	1	0	0	00	0	0001	00001001100100000001	13201
译码	1	0001	0	0	0	11	0	0	0	0	0	0	0	0	0	00	1	0000	00011000000000010000	30010
LW1	2	0010	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0011	00110000000000000011	60003
LW2	3	0011	1	0	0	00	0	0	0	0	0	0	1	0	0	00	0	0100	10000000001000000100	100204
LW3	4	0100	0	0	0	00	1	0	0	0	1	0	0	0	0	00	0	0000	00000100010000000000	8800
SW1	5	0101	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	0110	00110000000000000110	60006
SW2	6	0110	1	0	0	00	0	0	0	0	0	1	0	0	0	00	0	0000	10000000001000000000	100400
R1	7	0111	0	0	1	00	0	0	0	0	0	0	0	0	0	11	0	1000	00100000000001101000	40068
R2	8	1000	0	0	0	00	0	1	0	0	1	0	0	0	0	00	0	0000	00000100100000000000	4800
BEQ	9	1001	0	1	1	00	0	0	0	0	0	0	0	1	0	10	0	0000	01100000000101000000	C0140
BNE	10	1010	0	1	1	00	0	0	0	0	0	0	0	0	1	10	0	0000	01100000000011000000	C00C0
ADDI1	11	1011	0	0	1	10	0	0	0	0	0	0	0	0	0	00	0	1100	00110000000000001100	6000C
ADDI2	12	1100	0	0	0	00	0	0	0	0	1	0	0	0	0	00	0	0000	00000000010000000000	800
CALL	13	1101	1	0	0	00	0	0	0	0	0	0	0	0	0	11	0	1101	10000000000001101101	10006D

(c) 微命令字段设计案

图 **3**　微程序控制综合设计

2. 使用 MIPS RAM 作为一个接口，将本地 LAPTOP 上的 hex 即编译的二进制文件载入其中，即载入汇编程序（机器码）；
3. 进行单步或多步调试，这里主要是观察两个 MIPs 反汇编命令窗口、观察当前微指令地址（结合状态转换图）、观察 reg 堆、观察各个锁存器的探针、使用鼠标上下滚轮切换组件选择和调试模式来修改电路和查看没有探针的二进制内容——通过反复操作以上步骤，即可得知该指令或者说该程序有没有正常运行。进而对电路做进一步修改。

具体来说，工作细节如下例所示：

3.3.1 单指令调通

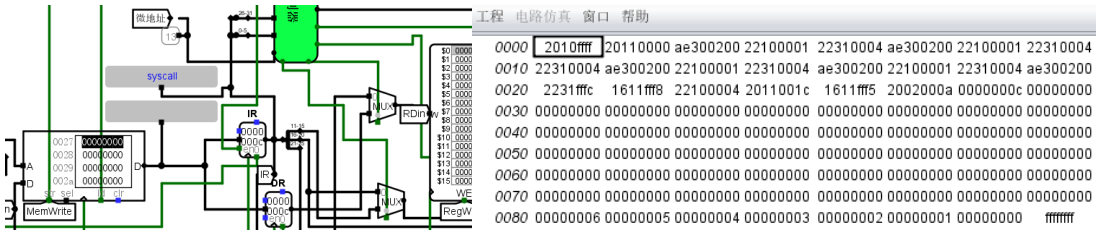
例如对一条 addi 指令，首先需要将 addi.hex 载入到内存，复位 CPU 做调试准备。然后需要对照着状态图和 addi.asm，结合语义特征，观察 addi 指令是否按照设计的状态流程 0 - 1 - 11 - 12 -0 正确运行。如果在中途发现信号或者数据错误，则需要重视错误发生的地点，这可能是一个锁存器、可能是一个微命令位（即控制器的输出端口的状态没有安装既定方式运行）、也可能是一个 reg（这是是 rw 或 w 出现了问题）。

总之，通过错误—定位错误—修改错误的调试流程，就可以通过指令的单步调试逐渐发现所有的错误进而将硬件和微程序的根源上的错误通通改正。

3.4 排序程序调通

重复以上单指令调试步骤，调试通过后就能确保 8 条指令均已经正确实现了。

然后直接载入运行 sort.hex，即可在 RAM 中查看得到以下运行结果。如下图所示，-1 到 6 共 8 个数字均已被正确排序，这标志着本次实验取得圆满成功。



(a) 运行结果-电路　　**(b)** 运行结果-数据

图 **4**　运行结果

3.5 验证

完成 sort 排序工作后还应当对实验结果进行验证。经过验证，本次实验到 systemcall 指令共运行了 891 个 clk，和标称值 891 一致。

四、 【小结讨论】

4.1 疑难解答

4.1.1 为什么不能合并 LW 和 SW 指令的第三个状态？

两条指令的第三个状态需要完成的任务相同，所以微命令控制字段是一样的。但是两条指令接下来指令的微命令不一样，所以需要不同的下址字段。

故而，不能合并 LW 和 SW 指令的第三个状态。

如果非要合并，则需要增加额外的条件字段以实现不同的微地址转移方式

4.1.2 多周期时钟周期更长？是否性能更差？

根据：

$$CLK_{sum} = CPI_{avg} * I_{cnt} \tag{1}$$

多周期 CPU 的 CPI 更大而指令的总条数不变，因此多周期时钟周期更长。

所以，同样的编译器编译出来的程序，多周期架构的 CPU 必然使用更多时钟周期。

但是多周期实现的 CPU 性能不一定就差。

通过使用多周期架构，时钟频率得以提升，因此单一指令的总的运行时间往往可以减少，这导致多周期实现的 CPU 往往能达到更好的机器性能。

4.2 小结

本次实验通过设计和实现一个 32 位多周期微程序控制的 MIPS CPU，使学生深入理解了 MIPS CPU 的设计原理和实现细节。实验过程中，学生不仅掌握了数据通路和控制器的设计方法，还通过反复调试和优化，提升了对硬件设计和调试的能力。

在数据通路设计方面，我通过合理摆放基础组件，并优化电路细节。完成了数据通路框架的基本设计。

在控制器设计方面，采用了选择-或门的方式完成指令译码逻辑和地址转移逻辑的设计，并通过在后续工作中反复迭代，最终实现了稳定运行的微程序控制器设计。对于ALU 控制逻辑，也是和指令译码逻辑的设计相似地，通过选择器配合指令完成了设计工作。

在 ISA 调通和联调过程中，主要任务是通过单步调试和多步调试，逐步发现并修正硬件和微程序中的潜在错误。在联调成功后最终成功运行了冒泡排序程序，验证了 CPU 设计的正确性。

总结，本次实验让学生掌握了 MIPS CPU 的设计和实现方法，在设计的过程中，我感受到了极大的成就感和获得感。通过课前预习，课中设计的实验模式我最终实现了该 CPU 的设计并对 MIPs 实现细节、微程序控制器、状态机设计、哈佛架构（学习理论基础时用到的单周期硬布线环境下使用的架构）、冯诺依曼架构（本实验）、硬件-ISA 粒度的程序调试有了深刻的理解。这为今后的学习和研究和工作打下了坚实的基础。