大实验：五级流水线 RISC-V 处理器

蒋昊迪　邢竞择

一、实验目的

1. 深入理解流水线结构计算机指令的执行方式，掌握流水处理器的基本设计方法。
2. 深入理解计算机的各部件组成及内部工作原理。
3. 加深对于 RV32I 指令集的理解。
4. 掌握计算机外部输入输出的设计。
5. 提高硬件设计和调试的能力。

二、实验内容

在大实验中，本组完成了 RV32I 的所有基本指令、基本的中断异常机制及 M,S,U 态切换、页表及虚拟地址转换、分支预测机制、指令缓存、TLB。能够运行三个版本的监控程序以及 uCore 操作系统。

实验分工为：蒋昊迪完成基本的中断异常机制及 M,S,U 态切换、页表及虚拟地址转换、TLB；邢竞择完成 RV32I 指令和额外指令、指令缓存、分支预测机制。

实验基于蒋昊迪同学实验六的流水线处理器框架。下面逐项介绍在大实验中完成的工作。

1. 基本指令扩充

绝大多数的基本指令扩充是容易的，只需指定相应的信号。对部分特殊指令，需要扩充 ALU 操作，通过添加新的 opcode 来让 ALU 进行相应操作。对于较为复杂的 CTZ 指令，使用 casex 语句建立多路选择器完成。

2. 中断异常机制设计

中断异常机制的设计主要需要解决控制状态寄存器（CSR）、中断异常信号数据通路、特权态切换等问题。由于发生中断异常并切换特权态时需要修改大量的 CSR，因此这一部分的控制逻辑可作为一个模块进行处理（后文称为 `CSR-Trap` 模块）。

`CSR-Trap` 模块的主要接口包括 CSR 的常规读写接口（类似于寄存器堆）、各类中断异常的信号入口以及必要的上下文（如 PC 地址、`mtval/stval` 的值）、是否发生 trap 以及跳转的 PC 等输出、全局的特权态及页表信息输出。

（1）CSR 读写指令

为了实现 uCore，我们实现了实验指导书中所列出的 CSR（其中，m/stvec 仅支持使用 Direct 模式）。对于 Zicsr 指令集中的 CSR 读写操作，采用和常规寄存器类似的数据通路，也即在 ID 阶段读取需要的 CSR，在 EXE 阶段进行相应的计算，并在 WB 阶段将数据写回 CSR 中。

由于对部分 CSR 的读写（如 `satp`, `m/sstatus`）可能对后续指令的读取译码产生较大影响，因此在 ID 段到 WB 阶段有 CSR 读写指令时，暂停后续流水线执行[1]（见冲突处理表）。

在 CSR-Trap 模块内部，CSR 读采用组合逻辑，CSR 写采用时序逻辑，在未发生中断异常的情况下，进行相应 CSR 的读写。

对 Zicsr 指令集所涉及的数据通路详见图 1，相应的控制信号见表 1。

（2）中断异常信号设计

所有的中断和异常将会在对应的指令运行到 WB 阶段时传入 CSR-Trap 模块进行判定和跳转。

对于异常信号，主要的任务在于收集各个阶段产生的异常，将相关的信号以及 PC，trap value 等值传递到 WB 阶段。具体来说，有以下异常：

[1] 由于绝大多数场景下不需要操作 CSR，因此我们认为这一操作对性能影响不是很大。

表 1: Zicsr 指令集主要控制信号表

指令	imm_type	alu_sel_a	alu_sel_b	alu_op
CSRRW	/	rs1	csr	sel_a
CSRRS	/	rs1	csr	or
CSRRC	/	rs1	csr	(not a) and b
CSRRWI	CSR	csr_imm	csr	sel_a
CSRRSI	CSR	csr_imm	csr	or
CSRRCI	CSR	csr_imm	csr	(not a) and b

所有指令的 `rd_sel` 信号均选择 csr, `rd_we, csr_we` 均为 1

- 在 IF 阶段（两阶段，具体见页表设计部分），需要判断指令页错误、指令地址非法、指令地址未对齐等异常；
- 在 ID 阶段，需要判断非法指令、ecall/ebreak 等异常，同时需要生成 sret/mret 的信号；
- 在 MEM 阶段（两阶段），需要判断数据页错误、地址非法、地址未对齐等异常。

对于中断，由于其发生与正在运行的代码关系不大，但是又必须指定一个确定的 PC 作为后续返回时使用的地址，且不受气泡影响，同时也为了支持 uCore 中出现的 `mret` 返回后立刻触发 S 态中断的情况，我们将中断与发生中断时处在 IF1 阶段的指令绑定。运行在其之前的指令，而该指令及其之后的指令不执行，这样 mepc/sepc 所填写的值恰好为该条指令的 PC，同时 `mret` 切换特权级后的第一条语句也会立即触发中断。

（3）特权级切换设计

所有的中断异常信号会在 WB 阶段进入 CSR-Trap 模块，在模块中根据优先级确定触发的 trap 的类型，并根据委托情况确认切换到 M 或 S 态（或者是 mret/sret），并根据这些信息完成对应 CSR 的修改以及下一条 PC 的计算。

特权级切换的冲突处理相较于 branch 指令来说更为复杂，因为它需要保证 MEM 阶段不会发生额外的写操作，因此在发生中断异常指令运行到 MEM2 阶段时（此时所有中断异常均已生成）就应该阻止下一条指令进入。同时它也需要考虑 IF 阶段忙时的阻塞操作。详细的设计见表 2.

3. 页表机制设计（含 TLB）

页表机制的设计主要在于设计 MMU 模块。

MMU 模块为一个状态机。首先判断直接映射和 TLB 是否存在表项，如果存在则输出，否则进入第一级页表的读取，读取完毕后并判断 page fault 后进行第二级页表的读取，最后判断 page fault，输出结果并将结果写入 TLB。

TLB 仅设计了一项，但是每页 4KB 的大小也可以保证在连续读写时的高命中率。

在数据通路的设计上，为了更好地解耦虚拟地址转换和实际数据的读取，同时提升效率，我们将 5 段流水线升级为 7 段流水线，将 IF 和 MEM 阶段拆成两部分，第一阶段进行虚拟地址转换，第二级段进行实际的数据读取。这样子需要重新设计部分数据通路和冲突处理，详见图 1 和表 2。

4. 性能优化设计

（1）指令缓存

指令缓存使用寄存器实现，采用 32 路直接映射，每条表项包括

- 1 个 VALID 位
- TAG 项，长度为 26
- 指令项，长度为 32

当 icache hit 发生时，直接从缓存中读取指令作为 IF2 的输出信号。当 icache 遇到 miss 情况时，便从内存中读取指令进行更新，且在等待读取时发出 wait 信号供 hazard 使用。

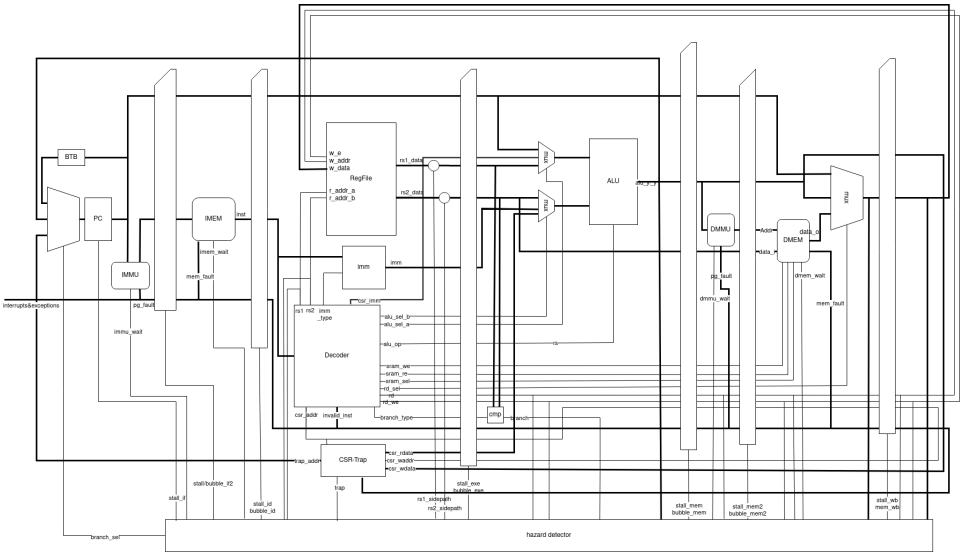

图 1: 数据通路图

当 ID 阶段发现指令是 `FENCE.I` 时，下一阶段在 ID 产生气泡（清除可能是未刷新的指令），对 IF2 进行 stall，并且将 icache 的 VALID 位清空。

（2）数据旁路

数据旁路的实现相对简单，只需将可能写回寄存器的信号 (ALU 输出，内存读输出，CSR 输出，PC + 4) 接入冲突处理模块，并根据选择信号判断应选择哪一路，以及该路数据是否已生成。在判定时，应该按照指令从新到老的优先级选择。

另外，由于 Wishbone 的限制，在加入指令缓存前，至多只有两条指令在流水线中运行，数据旁路无法体现其性能。

（3）分支预测

我们在实验中使用寄存器实现了动态预测的 BTB。它的内部实现了 8 路组相联的缓存，每一条表项包括：

- 1 个 VALID 位
- 2 个历史记录位
- TAG 项，长度为 31
- target 项，长度为 31

另外维护一组 `logic [2:0] rank[8]` 寄存器，用来支持 LRU 策略，`logic[x]==0` 表示 `x` 是最近使用过的一项。

BTB 预测时，若能在缓存中找到有效的表项，则根据历史记录决定预测结果，否则直接预测为 `pc+4`。

在 IF1 阶段，若没有收到分支错误或者某些 trap 相关的信号，则自动取用 BTB 的预测结果作为下一条指令的地址。

在 EXE 阶段，若发现是一个分支指令，则检查此刻 ID 中的指令地址是否符合预测结果，并将 `branch_taken` 信号传回 IF1 阶段。BTB 接着根据 `branch_taken` 信号更新历史信息（或者发现缓存不命中并进行一次替换），附带修改 `rank` 来刷新最新的访问顺序。

表 2: 冲突处理顺序表

冲突	IF1	IF2	ID	EXE	MEM1	MEM2	WB
MEM1 Wait + TrapWB	S	S	S	S	S	B	S
IF2 Wait + TrapWB	S	S	B	B	B	B	S
IF1 Wait + TrapWB	S	B	B	B	B	B	S
Trap in WB	Trap	B	B	B	B	B	B
MEM2 Wait	S	S	S	S	S	S	B
Trap in MEM2	S	S	B	B	B	B	
MEM1 Wait	S	S	S	S	S	B	
IF2 Wait + Branch	S	S	B	S	B		
IF1 Wait + Branch	S	B	B	S	B		
Branch	Branch	B	B	B			
fence.i	S	S	B				
Read After Write	S	S	S	B			
IF2 Wait/CSR Write	S	S	B				
sfence.vma (IF2)	S	B					
IF1 Wait	S	B					

注：S 代表 Stall, B 代表 Bubble

三、效果展示

1. uCore 运行效果

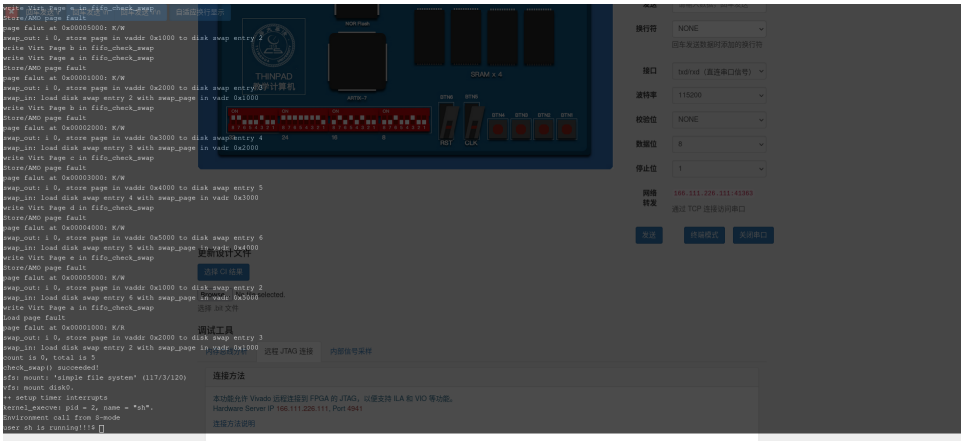

图 2: uCore 运行结果

图 3: uCore 运行结果

图 4: uCore 运行结果

2. 性能优化测试

经过验证，我们的 CPU 可以在 80MHz 的时钟频率下运行。

表 3 为标准测试的运行结果，除最后一列以外，其余测试均在 10MHz 时钟频率下测得，从左向右为递进关系，即右侧的测试中包含左侧的所有优化。

表 3: 性能测试结果（单位：秒）

测试名	基础版本	指令缓存	数据旁路	分支预测	80MHz
1PTB	161.112	60.45	53.689	33.564	4.194
2DCT	80.523	83.884	23.489	18.475	2.307
3CCT	187.891	87.182	87.29	46.931	5.872
4MDCT	181.201	110.73	83.89	73.873	9.228
CRYPTONIGHT	8.189	7.83	4.995	4.362	0.551

四、实验总结

1. 遇到的困难和解决方案

在调试过程中，由于串口输出在仿真过程中速度很慢，因此仿真到可能发生错误的地方时往往需要花费很长的时间，同时向串口中输入相应的字符也不是很方便。因此我们在确认了串口输入输出信息正确后，修改了监控程序的相关代码。其中我们删除了串口输入输出有关内容，

直接跳转到 `G` 指令的执行步骤，并硬编码起始地址；同时修改了 kernel 的编译流程，将待调试的用户态程序一起编入二进制字节流中供 testbench 读取。

在实现 BTB 时，我们尝试使用是否跳转作为 EXE 阶段判断跳转是否失败的依据，但这种方法对于跳转目标不固定的指令会出错。最后我们通过比较 ID 阶段的 pc 与 EXE 的预测结果来判断跳转是否成功，能确保执行正确。

另外，为了方便编写更为复杂的测试程序，我们编写了一个简易的 C 语言接口库，可以同时编译 RISC-V 格式的二进制文件和本地可执行文件，用于比较运行结果；同时整合了 `putchar` 函数的串口接口。另外，为了方便上载二进制文件，我们也为 term 增加了相应的功能。

在调试 uCore 的时候，由于初始化过程耗时很长，且无法像串口输出一样简单地删除了事，因此我们只能采用线上平台 + ILA 的调试模式。首先根据串口输出情况定位 bug 可能存在的地方，然后将 ILA 的触发器设置成该位置附近的 PC 值，并分析附近的关键信号值。

2. 心得体会

“奋战三星期，造台计算机” 是本课程，甚至是贵系最为著名的口号。经过这三星期的努力，我认为我们顺利地完成了造机的任务。

在设计处理器的中断异常的过程中，我设计了多版方案，并对照文档，仔细推敲了各种方案在不同指令组合下的数据通路变化，并分析利弊及实现细节，最终选择了目前采用的数据通路模式。因此中断异常模块自监控程序第二版一直到 uCore 的支持中始终没有产生较大的、影响整体设计的 bug。

在实验的过程中，调试更是令人头痛的问题。虽然在我们的 CPU 中总的 bug 数量不多，但是要定位每一个 bug 还是需要花费很大的精力。无论是仿真还是 ILA 上板，都需要在大量的信号波形中查找出异常的值，这确实需要耐心和细心。

在实验中，疫情也确实给我们带来了不小的影响。一些已经构思好的设想也没有机会去完成，这也是我们实验过程中的一些遗憾。当然，这次实验我们也收获颇丰，不仅体会了硬件的基本设计方法，并且在文档和代码的阅读中更加深刻地理解了课本上的知识，对相关的体系结构有了更进一步的认识。

最后，在整个实验的过程中，我认为助教团队提供的文档很好地帮助我们梳理了设计的目标和需要完成的任务，使得 uCore 的设计也并非想象地那么遥不可及。也感谢老师和助教们的付出。

——蒋昊迪

- 利用好模块化设计能够显著地简化接口、方便复用。例如 mem_controller 模块，在 wishbone 总线的 mux 之上又进行了一次模块化，提供了 wait 输出信号，使涉及 SRAM 的模块更容易实现。
- 流水线的各个阶段具体需要的周期数不同，对于 mmu、icache、dmem 这些需要访问内存的模块，在 hazard 中需要谨慎处理等待的情况
- 例如 exe 阶段产生分支预测错误的信号时，需要等待 IF1 和 IF2 的内存访问完成后才可以在 IF1 IF2 ID 中加气泡，具体的等待方式就是将 EXE 之前的阶段都进行 stall，这样分支错误的信号在下一个周期仍会保留
- 流水线的控制信号非常复杂，必须要厘清各个信号在何时生效。例如全部的中断和异常信号需要传到 WB 阶段再处理；CSR 寄存器的写操作需要传到 WB 阶段再执行；fence.i 则可以在 ID 阶段立刻处理。
- 全相联缓存比同样大小的直接映射缓存电路复杂程度大很多，我们根据缓存内容的特征，给 btb 和 icache 实现了不同类型、不同大小缓存。
- CSR 寄存器的读写不能实现数据旁路，CSR 的写信号从 ID 阶段产生，则只要 EXE、MEM1、MEM2、WB 阶段存在 CSR 写信号，就需要将 ID 前全部阶段暂停。

- 使用宏代替常数能显著方便开发。

——邢竞择

五、思考题

1. 流水线 CPU 设计与多周期 CPU 设计的异同？插入等待周期（气泡）和数据旁路在处理数据冲突的性能上有什么差异。

   答 异：流水线 CPU（在未发生冲突时）每一时钟周期每一部分均在处理一条指令，多周期每一时钟周期只在处理一条指令；同：每条指令均需要经过多个周期才能完成执行。

   加入数据旁路后，能够解决一部分数据冲突，尽早获得部分寄存器更新后的值，而不需要等到它写回寄存器后才能获得。

2. 如何使用 Flash 作为外存，如果要求 CPU 在启动时，能够将存放在 Flash 上固定位置的监控程序读入内存，CPU 应当做什么样的改动？

   答 首先需要添加一条连接 Flash 的 Wishbone 总线以及相关控制器，并为 Flash 分配对应的读写地址。然后将初始化的 PC 修改为 Flash 对应的地址。

3. 如何将 DVI 作为系统的输出设备，从而在屏幕上显示文字？

   答 在硬件层面，使用 Block RAM 或其它方式为 DVI 分配一块显存，驱动器按照一定的周期读取这些数据并发送到 DVI 设备上。在软件层面，根据字体文件将文字转换为点阵，并向显存的地址写入该图像。

4.（分支预测）对于性能测试中的 3CCT 测例，计算一下你设计的分支预测在理论上的准确率和性能提升效果，和实际测试结果对比一下是否相符。

   答 一轮循环执行的指令有

```
.LC2_0
     bne t0, zero, .LC2_1
     # some instr that won't run
.LC2_1
     j .LC2_2
.LC2_2
     addi t0, t0, -1
     j .LC2_0
```

   分支预测能够正确预测三次跳转，总共占用 4 个周期；无分支预测时，除了目标是 `.LC2_2` 的 J 指令以外，另外两个跳转指令会预测错误并浪费两个周期，总共占用 8 个周期。

   我们测得无 BTB 版本用时 87.3s，有 BTB 版本用时 46.9s，速度大约为 2:1，符合预期。

5.（虚拟内存）考虑支持虚拟内存的监控程序。如果要初始化完成后用 G 命令运行起始物理地址 0x80100000 处的用户程序，可以输入哪些地址？分别描述一下输入这些地址时的地址翻译流程。

   答 0x80100000 或 0x0000000。输入这些地址，并切换到对应 PC 执行时，首先会查询页表入口第 512(0x200) 或第 0 项 (0x000) 的第一级页表，接着查询该页表的第 0x100 或 0x000 项，得到物理地址，并到对应的物理地址处读取指令。

6.（异常与中断）假设第 a 个周期在 ID 阶段发生了 Illegal Instruction 异常，你的 CPU 会在周期 b 从中断处理函数的入口开始取指令执行，在你的设计中，b - a 的值为？

   答 该指令会依次执行 ID,EXE,MEM1,MEM2,WB 阶段之后进入异常处理入口，b-a=5.