哈爾濱工業大學(深圳)

HARBIN INSTITUTE OF TECHNOLOGY, SHENZHEN

第八届全国大学生计算机系统能力培养大赛

“龙芯杯”竞赛总结报告（个人赛）

指令集： LoongArch

姓　名： 马奕斌

2024 年 08 月 21 日

目　录

一、概述……1
1.1 初赛/决赛指标 ……1
二、系统详细设计……5
2.1 系统整体架构设计……5
2.2 流水线……6
2.3 取指扩展……6
2.4 总线……7
三、关键技术及特色/亮点……9
四、调试及优化……10
4.1 测试方法……10
4.2 调试和优化过程……11
五、复赛编程题设计……12
六、关键问题及解决方案……14
七、总结与展望……15
7.1 竞赛总结……15
7.2 未来展望……15
7.3 备赛经验及建议……15

一、概述

1.1 初赛/决赛指标

初赛三级功能测试：

100

lab1 在 FPGA 板 06 上的结果

```
Test pass
```

100

lab2 在 FPGA 板 06 上的结果

```
Arguments: 0x55555555 0x228 0xf5340af9
Fib info: 'Fib Finish.'
RandomTest info: 'All PASS!'
Fib Pass!
RandomTest Results: 0xbd908d1 0x228 0xf5340af9
RandomTest Pass!
```

100

lab3 在 FPGA 板 06 上的结果

```
Boot message: 'MONITOR for Loongarch32 - initialized.'
User program written
  Program Readback:
  0c0480020d04800204800015858080028e351000ac018002cd0180028e0080298
Program memory content verified
Program elapsed time: 0.000s
  R1 = 800023b0
  R2 = 00000000
  R3 = 807f0000
  R4 = 80400020
  R5 = 80400020
  R6 = 00000000
  R7 = 00000000
  R8 = 00000000
  R9 = 00000000
  R10 = 00000000
  R11 = 00000000
  R12 = 00000022
  R13 = 00000037
  R14 = 00000037
  R15 = 00000000
  R16 = 00000000
  R17 = 00000000
  R18 = 00000000
  R19 = 00000000
  R20 = 00000000
  R21 = 80100000
  R22 = 807f0000
  R23 = 00000000
  R24 = 00000000
  R25 = 00000000
  R26 = 00000000
  R27 = 00000000
  R28 = 00000000
  R29 = 00000000
  R30 = 00000000
  R31 = 00000000
Register value verified
  Data Readback:
  02000000030000000500000008000000d0000001500000022000000370000000
Data memory content verified
```

性能测试结果：

100

perf 在 FPGA 板 10 09 07 上的结果

```
=== Test STREAM ===
Boot message: 'MONITOR for Loongarch32 - initialized.'
User program written
  Program Readback:
  042000150580001506600014861810008c008028ac00802984108002a51080028
Program memory content verified
Data memory content verified
Test STREAM run for 0.033s
```

100

perf 在 FPGA 板 10 09 07 上的结果

```
=== Test MATRIX ===
Boot message: 'MONITOR for Loongarch32 - initialized.'
User program written
  Program Readback:
  048000150582001506840015078081031400150087 6e00588c8a40008ea640008
Program memory content verified
Data memory content verified
Test MATRIX run for 0.066s
```

100

perf 在 FPGA 板 10 09 07 上的结果

```
=== Test CRYPTONIGHT ===
Boot message: 'MONITOR for Loongarch32 - initialized.'
User program written
  Program Readback:
  048000156 55bbd15a5bcbb03669df515c6308003072000141010150 00f0015000
Program memory content verified
Data memory content verified
Test CRYPTONIGHT run for 0.159s
```

决赛测试结果：

100

final 在 FPGA 板 8 上的结果

```
Boot message: 'MONITOR for Loongarch32 - initialized.'
User program written
  Program Readback:
  0000400305800015a400802806600014c68844000c048003862100649e894000b
Program memory content verified
Elapsed time: 0.052s
Data memory content verified
```

1.2 设计概述

本设计为高度模块化的架构，模块前采用半互锁式通信，代码详见[我的仓库]。

- 时钟频率：初赛 220M，决赛 195M
- 流水线：IF-ID-EXE 三大模块，IF 阶段外接 Fetch Plugin 再连总线，EXE 阶段直连总线。EXE

支持乱序执行，即在当前的多周期指令执行期间，无数据相关的指令可调度执行。

- Fetch Plugin：含 144B，15 位 tag，每行 4 个字，共 8 行 的直接映射型 Icache，同时负责对送出的指令进行 BTFNT 分支预测。
- 总线：外接平台 Baseram、Extram 及串口。参数化多周期访存，支持无资源冲突式并行数据访存，即 EXE 阶段可在访问 Baseram 的同时访问 Extram。

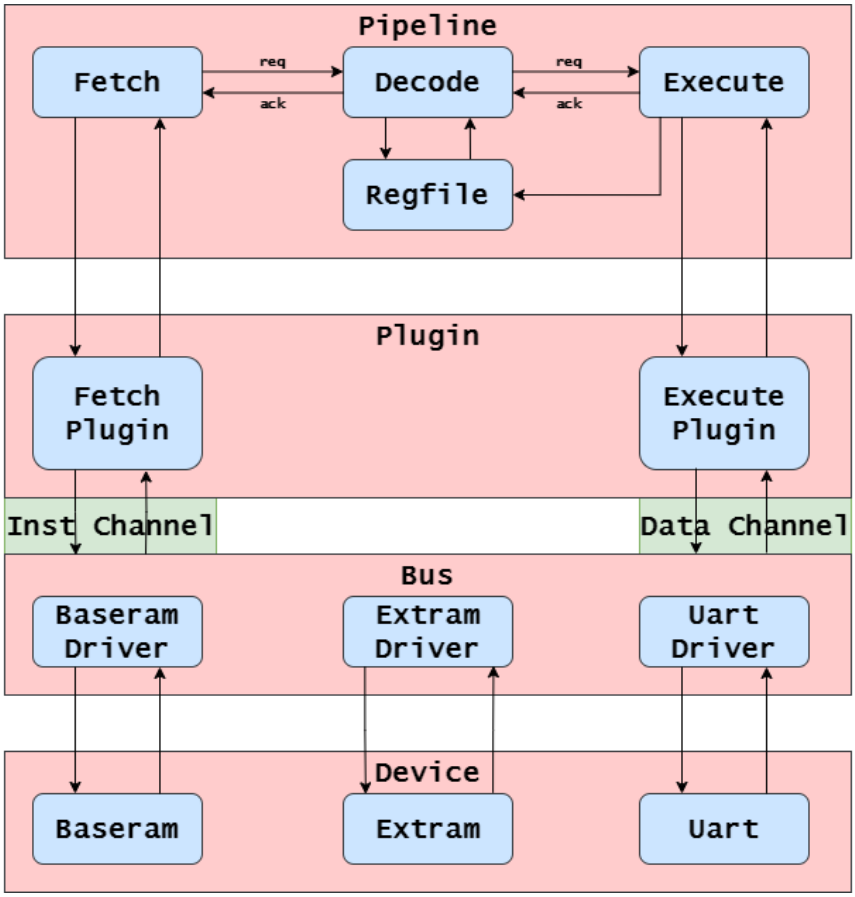
下面给出冒险的主要处理方式：

- 常规指令 RAW/WAW 冒险：在寄存器堆内直接通过转发即可解决。模块间采用半互锁式通信，且 Decode 模块在 req 未被 ack 时会不断向寄存器堆发出读请求，更新送出的数据。
- Load-to-Use 冒险：执行阶段维护 Load 的写控制信息，在 Decode 发出 req 时检查有无数据相关，仅在 Load 结束后（EXE 阶段进入 IDLE 状态）或无数据相关时 ack。
- 总线处结构冒险：Baseram 的读写为 EXE 阶段优先，即数据访存优先，指令访存仅在无数据访存时可发出。Extram 和 Uart 仅含数据访存，但存在 读时写/写时读 冒险，采用状态机序列化操作。

二、系统详细设计

2.1 系统整体架构设计

Seniorious 共分三大模块，分别为 CPU core(pipeline)、Plugin、Bus。Pipeline 负责实现指令集的主要功能，Plugin 负责加速耗时较长的操作，Bus 负责访问各个外设(baseram/extram/uart)。总体架构如下图所示：



按照 取指 --- 译码 --- 执行 顺序，一般指令的执行流程为：

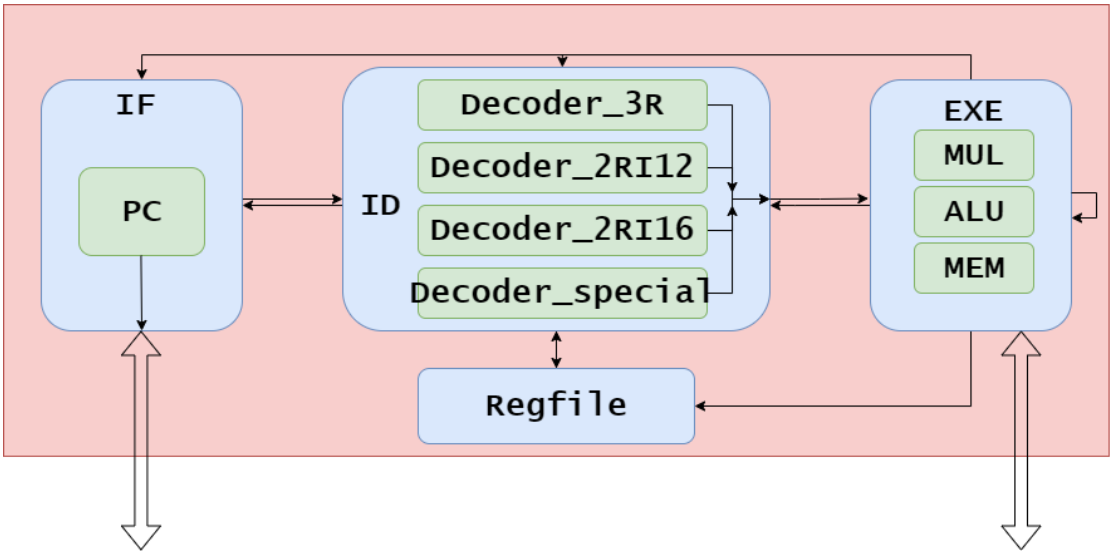
- Fetch 模块向 Pipeline 外发送取指请求后进入阻塞状态，等待 rvalid 信号拉高。Fetch Plugin 接收请求，Fetch Plugin 中的 Icache 检查指令是否已缓存，若否，向 Bus 发出请求，等待 Bus 的 rvalid 信号后将取得的数据写入缓存，而后将所需指令及指令的附加信息(分支预测等)送回 Fetch 模块
- Fetch 模块取得指令，拉高 req 信号，等待 Decode 模块接收后拉低 req，进行下一轮取指
- Decode 模块对输入的指令进行译码后从 Regfile 模块读出所需数据，拉高 req，等待 Execute 模块接收后拉低 req，进行下一轮译码
- Execute 模块根据输入的操作码及操作数进行相应的功能执行。若为访存指令，等待 Pipeline 外的输入信号(asideIn)中相应的 rrdy/wrdy 信号拉高后发出请求，进入阻塞状态，等待(rvalid/wdone)。Execute Plugin(无具体实现)负责 Execute ---

Bus 间的信号转发，从 Bus 存取所需的数据

- Execute 功能执行结束后，根据译码模块给出的写信息，将所需数据写到 Regfile 中

2.2 流水线

Pipeline 分取指-译码-执行三大模块：

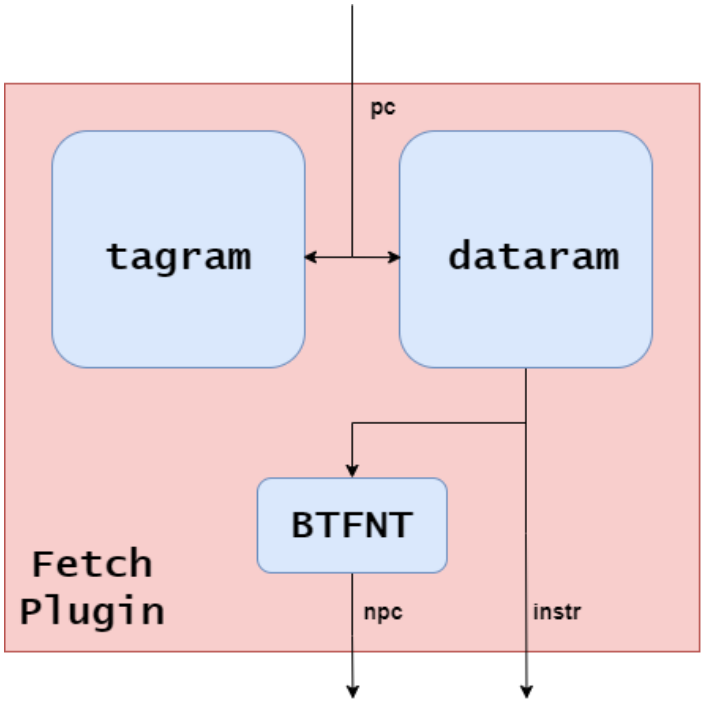


- 取指模块：向 Fetch Plugin 发出取指请求，向译码模块送出指令。取指模块的 PC 更新由来自 Fetch Plugin 的 npc 和来自 EXE 阶段的 br_pc 决定，当 EXE 阶段判明分支预测失败时，选择 br_pc，其他情况下选择 npc。
- 译码模块：主要参考 2023 年团队赛 LA 赛道 invalid-cpu 进行设计。将指令大致分为 3R, 2RI12, 2RI16, special 几类进行并行译码，通过各子译码模块的 match 信号是否有效选择相应的输出。同时，在 req 未被接收时不断向寄存器堆发出读请求以更新操作数。
- 执行模块：支持乱序执行。对于多周期指令如 Load，Mul 等，缓存对应的写控制信号，执行期间若后续指令无数据相关则调度执行，否则模块内部阻塞。此次设计较为糟糕，对 Load 和 Mul 都用专有的 Buffer 进行维护，导致后续其他指令基本没办法进行多周期扩展，卡死了频率。实际上设置一个向量寄存器，当某一个写操作未发出时记录对应的寄存器位为 1 即可。理论上如果普通运算，如加减移位等都进行拉长的话，频率能一直往上提。

2.3 取指扩展

Fetch Plugin 在 Seniorious 中负责提供 Fetch 模块所需的指令及相关附加信息，包括下一条指令的地址及分支预测结果。该模块的主要组成部分为 Icache 和 BTFNT 分支预测器。

下面分别进行说明：



- Icache：通过对性能测试程序的指令数进行统计及综合考虑时序优化等因素，最终确定 Icache 的配置为 144B，15 位 tag，每行 4 个字，共 8 行 的直接映射缓存。指令缓存采用 Block Memory 实现，分为 tagram 和 dataram，同时进行了流水化，使得缓存命中情况时可连续送出指令。
- 分支预测：BTFNT(Backward Taken Forward Not Taken)为静态分支预测，对计数 for 循环的预测准确率很高。由于三个性能程序均含有大量的 for 循环，故使用 BTFNT 也能有极高的预测命中率。此外，BTFNT 的实现简单，有利于简化布线及节省时间成本。

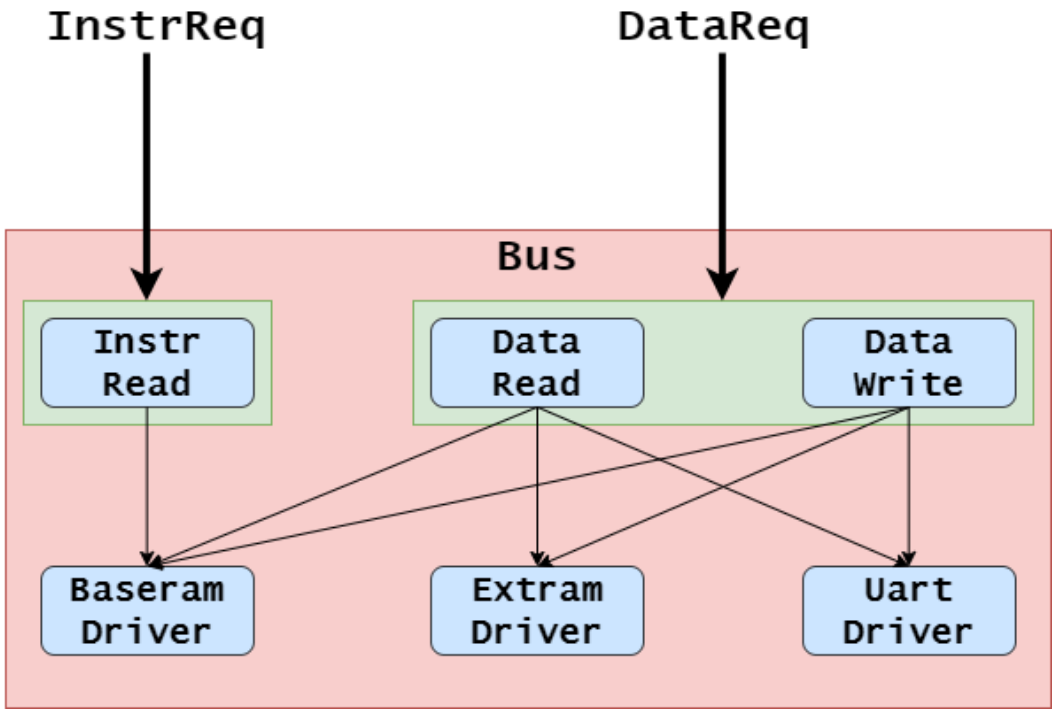
2.4 总线

Bus 模块负责处理取指请求与数据访存请求。在本次比赛的设计中，为充分利用外设资源，Bus 模块应在保证正确处理结构冒险的情况下，提供以下特性支持：

- 访存周期参数化。在 SRAM 芯片手册的规范下，为充分提高 CPU 主频，同时获得稳定的访存结果，Bus 需支持多周期访存。
- 支持连续访存。在一次 Icache 重填事件中需要连续读取多个字，Bus 需在开始处理事件后周期性递增读地址实现连续访存
- 无资源冲突性并发处理。在 Inst Channel 和 Data Channel 所访问的资源不同时，或者 Data Channel 的不同请求所访问的资源不同时，Bus 应能够同时处理，以最大化减少访存开销。

访存周期参数化和连续访存可通过设置计数器进行实现，此处不过多赘述。为支持第三个特性，Data Channel 需有分离的读写通道，因此，Data Channel 的状态机又可进一步拆分为读状态机和写状态机。

因此，各状态机间的相互关系可如下图所表示：



由上图可以看出，各个 Driver 均存在请求并发的可能，下面从 Driver 一侧进行分析：

- Extram Driver 和 Uart Driver 来说，由于处理器为顺序单发射型，同时间到达的 Req 至多只有一个，并发的情形只可能是在处理前一个请求的过程中有新的请求出现。此种情况下，只需将新的请求进行缓存后，使对应的状态机阻塞，在当前请求处理结束后检查阻塞状态，将阻塞中的请求进行处理即可。
- 对 Baseram Driver 来说，由于取指和访存请求可能同时到达，Driver 需要对同时到达的请求进行仲裁。由于取指经过指令缓存后大多数时间不会访问总线，因此固定优先级为 DataWrite > DataRead > InstrRead

遗憾的是，当前的 Seniorious 对上述想法的实现过于繁杂，从而限制了 CPU 的主频提升。在未来有机会的情况下，应将 Driver 从 Bus 中解耦，采用更加优雅的方式进行相同的功能实现以提高主频。

三、关键技术及特色/亮点

架构设计理念：

- 保证高度模块化：每一个模块必须尽可能地减少与其他模块的耦合，必须能够轻易地进行替换。模块化的设计有利于
  - 单元测试：对单个模块进行调试比对整个 CPU 核进行调试的难度要低很多。同时可组合若干个模块进行测试，保证在集成之前各个模块的功能正确
  - 针对性优化：对某一模块进行优化时不会影响到其他模块的功能，便于 DIY。以总线优化为例，耦合式的设计会导致修改总线时不得不同时修改取指和执行模块，最后只会是 Totally a mess.
- 保证参数化：Cache 参数，总线多周期访存，乘法多周期等均应留出配置空间，便于后期提高频率时进行修改。以多周期访存为例，一个糟糕的设计不得不修改流水线，添加一个访存级，连接端口等。参数化的设计只需要改动一个数字即可。
- 层次化设计：Indirection 能够提供更多的设计空间。以本架构为例，在 Pipeline 层和 Bus 层中间添加一个 Plugin 层可以留出更多的设计空间。例如在 Exe Plugin 中进行硬件加速，直接接管总线，下面以本次比赛的决赛编程题“找最大值”举例说明：
  - 由于进行硬件加速不需要理会汇编指令，可将 mul 指令进行利用
  - 监控程序启动后，执行用户汇编指令，汇编中仅有一条 mul 用于激活 plugin 的加速功能。
  - 执行阶段检测到操作码为 mul 时，向 Exe Plugin 发出请求，进入阻塞状态，等待 Exe Plugin 发出 done 信号表示执行结束。
  - Exe Plugin 进入加速状态，将所有参数（数组初始地址、结束地址、写入结果的地址等）全部内置，连续向总线发出访存请求并进行比较，到达结束地址时发出写请求后，向 Exe 阶段发送 done 信号。
  - 假设数组共有 N 个数，访存周期为 4 周期，硬件加速的耗时约为 4N 个周期。
  - 假设汇编采用循环实现，每个循环体内部有 循环条件判断-访存-比较-循环变量自增四条指令，一次循环约 $1+4+1+1=7$ 个周期，共约 7N 个周期，其中若有分支指令，还应考虑分支预测失败惩罚。当 N 十分大时（决赛为 0xc0000），运行时间差距很大。
  - 不难看出能否硬件加速的重要性（本人决赛事故现场，纯属意外…）

四、调试及优化

4.1 测试方法

项目采用 sbt 进行构建，结合 gtkwave 进行波形调试，主要步骤如下：

```
# 在 Chisel 工程主目录下：
sbt run 即可生成 sv 文件，移动至 vivado 工程文件夹中即可进行综合
# 指定测试文件
sbt “testOnly YourTest -- -DwriteVcd=1” 而后进入 test_run_dir 下对应的文件夹，运行：
gtkwave YourTest.vcd 查看得到的波形
```

测试可指定代码和数据文件：

```
// in Soc.scala
// modify the path to the test files
val baseRam = Module(new BaseSram("./func/bintests/lab3.txt"))
val extRam = Module(new ExtSram("./func/bintests/data.txt"))
```

下面给出可能的几种测试方式：

- 用 verilog 写 testbench，在 vivado 中进行仿真（不推荐）：Verilog 仿真缺少足够的报错，如组合环路、端口未连接等，调试过程中耗费时间在这些上面是痛苦的。以及丑陋的 vivado，在集成仿真时很麻烦。
- ChiselTest（推荐）：提供充足的报错信息，保证只要能编译通过，仿真中就不会有低级错误出现，节省了大量精力。同时，可以使用 Scala 的语法进行测试文件的编写，敏捷开发大法好。
- Trace：龙芯杯个人赛可使用 golden_trace 测试框架，该框架由往年的参赛者搭建。Trace 的优点是快速定位问题，但无法提供太多的进一步的信息。
- 综合实现比特流后到平台测试（不推荐）：此种方法最为低效，如果出错了则需要重新综合实现，或者 JTAG 调试，较为麻烦。建议本地测试通过后再拉到平台测试，如果平台测试和本地测试不符，则一次性拉取多个信号进行 JTAG 调试，争取一次定位问题。

一个好的测试框架是非常重要的——如果你每次都必须综合实现后拉到平台，然后看到 error 信息之后瞎改一通再综合实现，那就只能祝你能完赛了。

一个好的测试应该：

- 有良好的错误检查：组合环路、端口未连接、未初始化等低级错误不应人工通过波形进行检查，好的测试应该能在编译期就发现这些错误。
- 单元测试方便：你应当能快速地对核上的一个模块，如 Decode、Multiplication 等进行功能测试，确保功能正确后再进行集成。
- 调试方便：你应当能获取足够多的信息用于定位问题。

综合以上考虑，我的建议是使用 Chisel：

```
sbt “testOnly SocSpec -- -DwriteVcd”
```

4.2 调试和优化过程

调试和优化以三个性能测试程序为导向，一步步的进行。

- 分支预测：原始版本为静态预测，总是预测不跳转。由于三个测试程序的主体均为 for 循环，且循环的次数较大，考虑采取 BTFNT(Backward Taken Forward Not Taken)静态预测策略，效果显著。实现起来较为简单，只需要在取指令的时候判断是否为分支指令并且立即数最高位为 1 即可。时序友好，对后续提升频率有益。
- 指令缓存：三个性能测试程序的指令数较少且以 for 循环为主，将会大量重复执行某一段指令。因此，Icache 能对性能提升有很大帮助。Icache 直接映射，存储 32 条指令便足够，不仅效果显著，还对提频率有帮助。本设计采用 Block Memory 作为存储体，故还需对 Cache 进行流水化，使其在 Hit 时能连续送出指令。
- 多周期访存：提高性能最直接的方式就是提高频率。重构总线模块使其支持参数化多周期访存后不断进行调整，在确保三个测试程序执行时间之和有所下降的前提下尽可能提高频率。
- 乱序执行：在多周期访存的情况下，Load 和 Store 指令无法在一个周期内得到结果，后续指令阻塞导致效率低下。因此重构执行阶段使其在 Store 发射后的指令继续执行，Load 发射后无数据相关的指令继续执行。在之后频率提至 200M 时多周期乘法也采取了同样的思路，效果显著。
- 并行外设访问：由于 Stream 测试程序是 Load From Baseram, Store into Extram, 因此如果总线能在将数据存入 Extram 的同时从 Baseram 加载数据，该程序能得到很大的性能提升。然而，由于 Stream 在三个测试中的时间占比较小，Stream 优化显著，但总时长优化程度一般。

下面指出一些陷阱：

- 不需要 Dcache：三个测试程序均无数据的时间局部性，并且 Crypto 是随机访存，Dcache 更有可能降低性能。
- 不需要复杂的分支预测：静态预测即可有很好的效果，复杂的分支预测会导致时序紧张，不利于提高频率。
- 频率并非越高越好：60M 单周期访存，双周期则需要 100M，三周期则需要 150M，四周期不应低于 190M，五周期不应低于 240M。

五、复赛编程题设计

测试说明

1. 题目：数组最大值

   给定 32 位非负整数数组 A，计算最大值，并存到结果目标地址，结果为 4 字节无符号整型。

2. 规定：

   数组 A 地址为 0x80400000 ,数组长度为 0x300000 字节 ,即 A 的范围是 0x80400000-0x806fffff。

   结果目标地址为 0x80700000。

3. 提示：

   测试数据为随机数据，所以结果只能通过计算得到

提交说明

1. 实现汇编程序完成指定要求，汇编程序写在仓库根目录下 asm 文件夹内的 user-sample.s 文件里，最后标记 最终版本。
2. user-sample.s 文件里提供了一个模板汇编程序，这段程序的功能是计算斐波那契数并存入 SRAM
3. 允许更改硬件设计，保证独立完成设计即可。
4. 比赛分数:未通过测试 0 分，通过测试的按执行时间映射到[15,30]，最短时间 30 分，最长时间 15 分

测试要求

1. 虚拟内存空间为 0x80000000～0x807FFFFF，共 8MB，要求整个内存空间均可读可写可执行。
   - 0x80000000～0x803FFFFF 映射到 BaseRAM；
   - 0x80400000～0x807FFFFF 映射到 ExtRAM。
2. CPU 字节序为小端序。
3. CPU 时钟使用外部时钟输入，50MHz / 11MHz 两路时钟输入均可使用。
4. 在复位按钮按下时（高电平）CPU 处于复位状态，松开后解除。
5. CPU 复位后从 0x80000000 开始取指令执行。

本次复赛的编程题目在完成难度上几乎为 0。遍历数组进行比较即可。

然而必须注意的是，复赛汇编代码如果有 bug，如数组访问越界，平台评测脚本不会给出任何输出。这个问题十分具有误导性，因为理论上讲程序是在监控程序 boot 之后执行 G

命令后才开始跑的，即使程序有问题也应该会有 boot 输出，没有输出则会误导选手认为 CPU 的核出现了问题。而改核调试在决赛现场是具有极大压力的——万一改不出来就爆零了。本人即是因为这个问题，在写硬件加速的途中不得不转移注意力进行调试，最终没有多少时间进行加速，只能提交了一个 naïve 的版本。

尽管如此，一些常见的优化思路还是容易想到的。

首先分析题目的关键点：

- 数据类型为 4 字节无符号整型，即需要做无符号比较
- 数组长度较大

下面给出两个方向的优化思路：

1. 指令级优化：

   本题最直接的方法是实现一条 bgeu 指令，但使用分支进行赋值会导致大量的分支预测失败从而降低效率。因此，可以考虑实现一些特殊功能的指令：

```
Max rd, rj, rk
If $rj >= $rk, $rd = $rj
即将两个寄存器中的最大值写入目的寄存器。
可通过指令 alias 实现，例如，由于程序中不需要使用乘法指令，可将乘法在核中的具体实现改为 max。
（因为汇编代码需要提交平台在线编译，所以自定义指令码不太可行）
```

通过使用 max 指令，汇编代码的指令数可以明显减少，进一步的，可以做如下设计：

```
Max rd, rj, imm
Load a word in (rj + imm), compare it with rd, if greater, write to rd
即合并上一版本的 max 和 load 指令，进一步减少循环体的指令数。
可进一步合并下一个地址的计算。
```

事实上，这种优化算是一种“半硬件半软件”级别的优化，同样需要改核，但是仅是将现有的几个功能进行合并。尽管如此，指令级优化也能取得很好的效果。

2. 硬件加速

另一种方法是通过写一个加速器挂载在总线上，直接过一遍总线数据。这种方法相较指令级的优势是减少了分支指令。

硬件加速有赖于架构的设计，及设计者所设计的架构是否方便直接写一个加速器挂上去而不影响到核本身的功能（监控程序还是要起的）。如果是 AXI 总线，则很容易挂加速器，但 AXI 总线会限制核的频率。如果是自己设计的总线，则在设计时最好考虑给硬件加速留出空间。

另外要说的是，本次比赛有人采取了协处理器和监控程序同时运行的方式，抢在监控程序启动之前（这部分时间不计入性能）执行完需要的操作，最终运行时间为 0.00s。这种方法不知道明年会不会被制裁，最好也留意一下。

六、关键问题及解决方案

下面简述几个主要的问题：

- 架构设计粗糙：算上寒假期间看《自己动手写 CPU》写的 MIPS，前前后后我总共写了三版。中间的一版实际上只是《自己动手写 CPU》的 LA 指令集翻版，删除了不必要的指令和延迟槽且加了点私货。然而该版架构耦合性过强，导致很长一段时间我不想对他进行任何修改。直到暑假开始时重构了一遍，变成现在的模块化设计。建议在编码之前仔细推敲架构，多考察架构的优缺点，比如“某一模块方便替换吗”“如果我想优化这个部分，我需要修改多少东西？”
- 不自信：在 CPU 核经过充分测试后仍保持怀疑，导致在决赛现场由于平台无任何输出时第一时间认为问题在核而不在汇编代码上。浪费了两个多小时后将版本回退至初赛提交版本，发现汇编数组越界。最终只落得个无任何优化，没爆零就庆幸的结果。
- 准备不充分：决赛前仅将架构修改为支持 1-2 个操作数运算（如开平方、除法运算）加速架构，祈祷不会考数组操作的题。结果怕什么来什么。

七、总结与展望

7.1 竞赛总结

龙芯杯可以算是我在系统设计方面的第一个较为复杂的 project。自己 diy 架构的过程也挺有趣的，但是 vivado 编译综合实现的时间太长了，人生苦短，远离 vivado……

今年个人赛可以算是卷出了新高度。LA 赛道一等奖为单发 267M，性能测试 0.222s，MIPS 赛道一等奖为双发 180M，性能测试 0.189s。而二等奖不是双发就是单发 250M 往上。弱小的我在单发 220M 性能测试+决赛差点爆零无优化的情况下拿个三等只能说是意料之中。

唉唉，遗憾离场。

7.2 未来展望

由于龙芯杯现在能保研加分，以后的比赛只会越来越卷。单就个人赛而言，由于一个赛道一等+二等总共就 3 个名额，想拿二等往后估计都得是单发 250M 或双发了。就目前看来，单发基本上已经是走到尽头了，如果想拿好名次的话，应该不畏困难做双发。

个人感觉个人赛之后应该会加点测试，给分支预测和 Dcache 留出一席之地。就目前来看这两个在 CPU 中具有很重要地位的功能部件在目前的性能测试中不能说是毫无作用，只能说是没啥卵用。

7.3 备赛经验及建议

报名之前可以问问自己参加龙芯杯的动机。下面给出了几种可能的动机及相应的备赛思路。

a) 我不想上小学期，听说打龙芯杯可以不上我就来了。在这种情况下，你不需要做太多，只要报个名，在下个学期开学一两周前提交参赛总结即可。不需要进决赛，也不需要通过初赛的所有测试（或许以后会提高要求？）。理论上你能跑过第一级测试就可以了。

b) 我想保研加分，要求不高，能加分就行。在这种情况下，你的目标就是进决赛。今年进决赛的门槛为单发 60M 的朴素五级流水。按此情况预计 2025 年单发 60M 加个 Icache 就能堪堪入围，最好进行多周期访存拉到 100M 以上，应该是相对比较稳了。进决赛后可以直接摆烂，决赛爆零也没关系，包三等奖的。

c) 我想拿一等奖赚 1w RMB 去提一台 mac。Oh, 这是一条注定艰险的道路，你要做好在备赛期间付出巨大精力的准备。你应当以超频双发射处理器为基本设计并进行你所能想到的各种优化。祝你好运，哥们。

d) 我对系统设计很感兴趣，打打玩，顺带能拿个名次是最好的。我参加的动机偏兴趣多一点，所以整个架构基本都是自己设计的，这个过程也还算有趣。如果你也是这样的话，需要克服的只有和 vivado 搏斗及波形调试的痛苦而已。同样的，祝好运，哥们。

下面给出一些备赛的建议，以下建议从一个对计算机组成原理基本不了解的小白角度出发，从入门到提高进行分点。有基础的同学可以适当略过前面的几点。

- CSAPP：初探计算机系统，lab 是精华，对比赛来讲做到 cache lab 即可。如果链接失效了可以直接 google 15213.
- 雷思磊《自己动手写 CPU》：MIPS 处理器的实现，但是没必要全看，对照龙芯杯 MIPS 赛道三级功能测试所需要的指令，只实现那些指令就可以进入下一步了。
- Chisel(optional): 用 verilog 也可以写出性能很好的核，我校今年 MIPS 的二等奖就是 verilog 写的 250M 单发。尽管如此我还是推荐 chisel，敏捷开发、测试方便、OOP、FP，并且决赛写加速器也可以直接 chisel“高级语言”编程。
- 选择 LA 或 MIPS 赛道，以通过三级测试为目标进行敏捷开发。建议先写个简单的单发五级流水，一开始频率不用太高，10M 就行，一个一个测试通过，了解怎么访问平台的 SRAM 和串口等。在平台开放后的一个星期内最好就已经能跑性能测试。然后做一下 Icache 和多周期访存，了解基本写法，频率拉到 100-200M。估算的话可以按单周期访存极限频率 65M，双周期 130M 来，但一般只会离极限越来越远。不需要去看 SRAM 芯片手册，就算知道必须维持 15ns 也没啥卵用，不是你能控制的。良好的设计应该能做到单周期访存 60M，双周期访存 110M。
- 选择设计方向。如果只想拿个三等的话可以开始摆烂了。想冲一二等的话有两条路可以选：单发超频或双发。选好之后开始重构你的 CPU 核，进入自己设计架构的 DIY 阶段。
- 单发超频的目标应放在 250M 以上，这注定了你后续的开发是极其无趣的，大多数时间将耗费在 vivado 综合实现上，还要祈祷平台的测试板性能还行才能跑过。另外，你应当设计架构使得硬件加速方便，不然决赛基本没有太大的优势。
- 双发是推荐的选项，单发 267M 性能测试也还是 0.222s，双发 180M 就能突破 0.2 大关为 0.198s，无疑是更优的选择。选择双发意味着仍需要付出很大的精力，但设计部分还是相对较多，比一直综合调频有意思的多。
- 决赛要 win 最终仍是逃不了改核，不管是扩展指令还是硬件加速。汇编不会有太大的难度（今年求最大值这个真没绷住）。个人还是推荐 Chisel+硬件加速这条路，具体如何操作的话，让 AI 用 Chisel 写点二分查找啥的就知道了。
- 最后一条建议：不要自己一个人闷着头搞，善用 AI，善问学长学姐，在前人的肩膀上进行开发，祖传代码不丢脸（）。