
# 第八届全国大学生计算机系统能力培养大赛

# 龙芯杯团体赛 决赛设计报告

大江大河队

储溦 胡揚嘉 王翔辉 杨译博

cw1@mail.ustc.edu.cn

指导教师：卢建良 赵雅楠

2024 年 8 月 13 日

# 目录

# 第一章　引言

本设计报告旨在全面介绍我们团队为第八届全国大学生计算机系统能力大赛龙芯杯团体赛所精心设计的 CPU 架构。我们的目标不仅是开发一款基于龙芯架构的处理器，这款处理器全面支持龙芯 32 位精简版（LA32R）指令集中的基础整数指令的所有功能和除 TLB 以外的中断、例外功能，更重要的是，它能够无缝集成并高效运行大赛提供的功能测试程序和性能测试程序，展现团队的协作与创新能力。

在现代计算技术日新月异的背景下，CPU 的性能与效率成为衡量计算机系统整体表现的关键因素。为此，我们团队采用了顺序双发射且精心优化的八级流水线设计，包括取指（IF1，IF1-IF2）、预译码及译码（IF2-ID，ID1-ID2）、分派（Issue-EX）、执行（EX）、访存（MEM）及写回（WB）等关键阶段。通过细致的流水线规划与优化，我们的处理器在确保满足竞赛的功能要求的同时，实现了指令执行的高效与流畅。

此外，我们设计的 CPU 还集成了指令高速缓存、数据高速缓存及分支预测器，能够最大限度的发挥性能优势，同时实现了 AXI 接口的封装，展现了良好的可扩展性和灵活性，为不同应用场景下的系统整合与性能提升提供了坚实基础。报告中，我们将详细阐述处理器的整体架构设计、各功能模块的具体实现细节以及在实际功能测试和性能测试中的表现，全面展现团队在 CPU 设计领域的深厚功底与创新精神。

通过本次团体赛的设计与实现过程，我们团队不仅深化了对 CPU 架构复杂性的理解，更在团队协作、问题解决及创新实践等方面获得了宝贵经验。这些经历无疑将为我们未来的计算机系统设计与研发之路奠定坚实的基础，激励我们在技术的征途中不断前行，追求卓越。

# 第二章　说明

**1. 参考设计说明**

本次的处理器设计除了提供 AXI-4 转接功能的 AXI-interconnect 模块是借用 Github 的 AXI 官方 AXI 转接桥源码之外，[1] 其余部分均由我们小组独立完成。

**2. 上板操作说明**

性能测试烧板后，多次复位后可能出现所有测试均不通过的情况，这时需要重新烧板，即可正常测试。

# 第三章　CPU 基本流水线设计

## 第一节　流水线总述

本次 2024 龙芯杯我们队伍设计了一个基于 Loongarch32-Reduced 指令集的顺序双发射八级流水线超标量处理器，总计实现了初赛所需要的条指令，支持基础整数的所有功能和除 TLB 以外的中断、例外功能，最终通过了大赛提供的决赛功能测试与性能测试。本次我们设计的 CPU 架构如图 3.1所示。

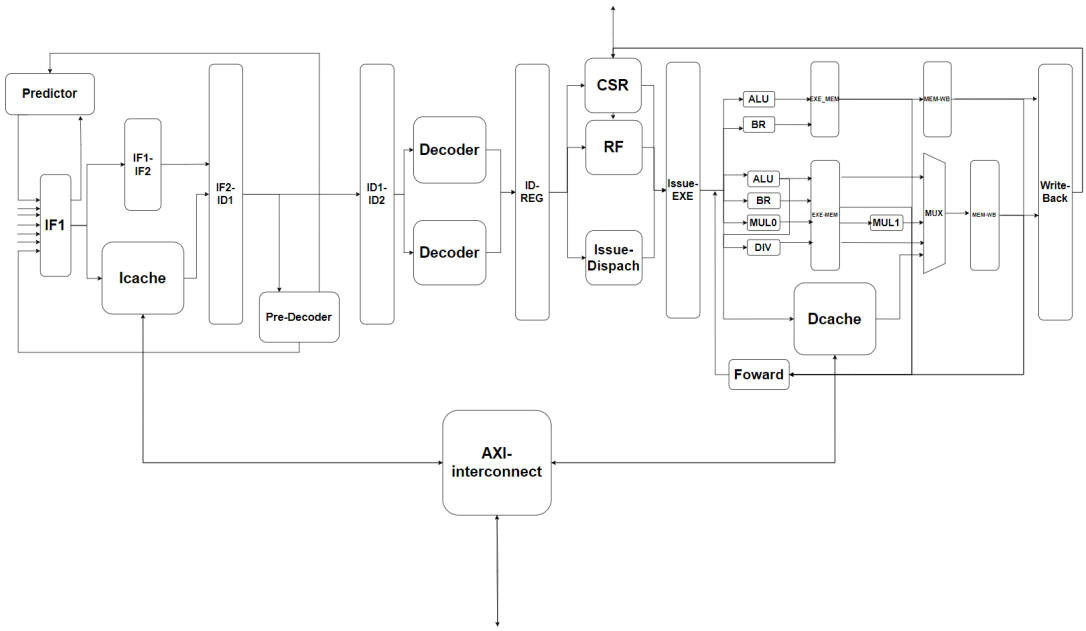


图 3.1　CPU 总体结构设计

## 第二节　发射阶段设计

## 一、IF1 模块

由各路地址和选择地址信号抉择出当周期使用的 PC 值。分发给分支预测和 ICache 取指令，同时保存 PC 于段间寄存器中。

## 二、IF1-IF2 模块

等待 ICache 取指完毕，同 IF1_IF2 段间寄存器保存的 PC 一同保存在 IF2_ID1 段间寄存器。

## 三、IF2-ID1 模块

取出 PC 和 IR，进行预译码。对于有要调整的 PC 及时发送跳转信号给 IF1 阶段 ID1_ID2 段间寄存器是 Instruction Buffer，主要作用是第一次缓冲流水，保存 PC 及其对应的信号。

## 四、ID1-ID2 模块

正式的译码阶段，吗每条指令进行译码，同时处理例外以及异常时间。

## 五、Buffer 模块

Buffer 的设计的核心动机在于处理处理器在执行指令过程中遇到的多种延迟和冲突问题，以提高整体性能。现代处理器采用双发射架构，理想情况下每个时钟周期能够同时取指两次，但实际上，由于诸如指令缓存（ICache）未命中、流水线清空等问题，导致每周期实际取指令数远少于理想值。

在执行阶段，处理器也面临着诸如数据相关性（RAW）、加载延迟（LD+x）、数据缓存未命中（DCache MISS）、乘除运算阻塞等挑战，这些因素都导致处理器在每个周期内无法同时执行两条指令。

因此，为了最大程度地利用处理器资源并减少性能损失，需要引入一个缓冲机制。该设计采用循环数组实现的 FIFO（先进先出）队列，具备双端口读写功能。每个时钟周期，根据指令的有效性，将指令放入缓冲区中。然后，在下一个周期，处理器根据缓冲区中剩余指令的数量，尽可能多地发射指令到流水线中。

在每个周期结束时，根据实际发射的指令数量（考虑到相关性、阻塞、停顿等因素），更新缓冲区的头部（head）和尾部（tail）指针。这种设计能有效地平衡指令的获取和执行之间的时序差异，从而提高整体处理器的效率和性能。

通过 Buffer 的引入，处理器能够更灵活地处理指令执行过程中的各种复杂情况，如数据依赖和资源竞争，使得处理器在面对实际工作负载时能够更加高效地运行，从而提升系统的整体吞吐量和响应速度。

### 六、Issue-Dispatch 模块

完成发射阶段的最后检测。主要考虑如下：同时发射的 RAW 相关，例外和跳转不和存储类指令同时发射，ALU 和 BR 类外所有指令只能在下面发射的问题，LD/乘除 + 指令的互锁问题。同时完成 Reg File 和 CSR 寄存器的读取任务。

## 第三节　执行阶段设计

整体采用顺序双发射的设计，分为 A、B 两通路。同时发射两条指令时，次序较先的指令发射到 A 通路，次序较后的指令发射到 B 通路。A 通路允许发射 ALU 指令和 BR 指令，B 通路允许发射 ALU 指令、BR 指令、LD/ST 指令、乘除法指令、CSR 读写等特权指令、定时器读指令。各流水段包含的模块或实现的功能如下：EX 段：Forward 前递模块、FU_ALU 计算单元、FU_BR 分支跳转单元、DIV 除法器、MUL1 乘法器第一部分，FU_CSR 对 ALE 例外和 CSR 读写的处理单元，Stable_Counter 计数器，能够处理并向 DCache 发送读写信号。MEM 段：MUL2 乘法器第二部分，RF 写回数据多选器，FU_CSR2 例外中断最终处理单元，中断产生后直接当周期送至 MEM 段，能够记录中断信号直至中断可以开始处理，能够记录 DCache 给出的读数据直到被实际使用。WB 段：发送 RF 写、CSR 写、例外中断处理相关信号，如果有因任何原因（CSRWR/CSRXCHG/例外/中断/ERTN）导致的 CSR 内容变更，还会同时发出 WB_flush_csr 信号清空整条流水线并使 PC 跳转到合适位置。执行段流水如图 3.2所示

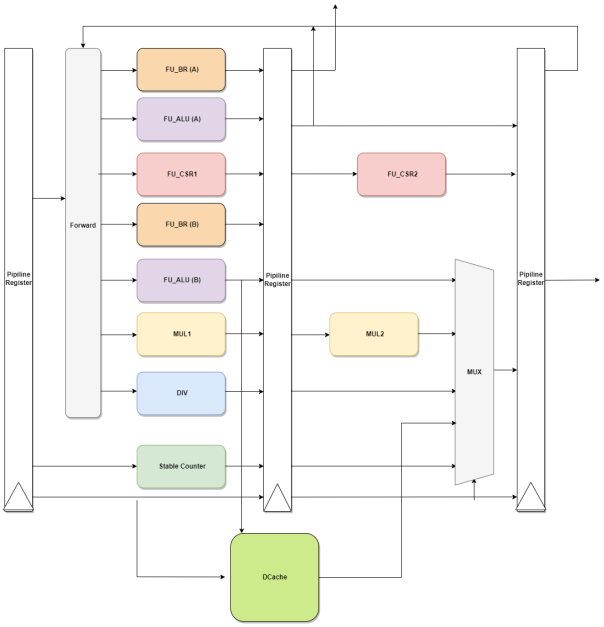


图 3.2　执行段流水架构

## 一、DIV 模块

共 33 个周期，1 个 EX 初始准备，32 个 EX 计算，然后流入 MEM 输出结果。除法器实际计算中使用的是被除数和除数的绝对值，得到商和余数的绝对值，最后根据被除数和除数的符号再对商和余数做处理。除法器中的 64 位寄存器 dvd-rmd 高位为被除数，低位为余数，初始被除数在低位，逐步左移，并在最低位填入余数的下一位。第一个 EX 周期将被除数的绝对值填入 dvd-rmd 的低 32 位，此后 32 个 EX 计算周期，计算每轮将 dvd-rmd[62:31] 与 32 位除数绝对值 dvs 做比较，初始 dvd-rmd[31] 与 dvs[0] 对齐，最后一轮计算前 dvd-rmd[62:31] 与 dvs[31:0] 对齐，最后一轮计算后，dvd-rmd[63:32] 为余数绝对值，dvd-rmd[31:0] 为商绝对值，再根据被除数和除数的符号做正负的处理。33 个 EX 周期中，前 32 个周期给出 stall-div 信号停顿自 ISSUE/EX 段间寄存器之后的流水线。计算完成后流入 MEM 段。

## 二、MUL 模块

采用 2 位 Booth 编码加华莱士树生成两个 64 位临时结果，再用 64 位全加器实现最后的加法。其中 2 位 Booth 编码和华莱士树均在 EX 段实现，EX 段阻塞一个周期，2 位 Booth 编码和华莱士树各使用一个周期完成计算。EX 段乘法器收到使能信号后，给

出一个周期的 stall-mul 信号，停顿自 ISSUE/EX 段间寄存器之后的流水线。MEM 段进行 64 位全加器的计算。

## 三、Branch 模块

输入：寄存器读数据、立即数、指令对应 PC、分支指令类型（独热码）、分支预测器的预测跳转地址。

输出：是否需要修正分支预测的结果（即再重新进行跳转）、修正时应跳转到地址、指令原本是否应跳转、指令原本的跳转地址。

分支跳转模块的输出发给 EX/MEM 段间寄存器，最后在 MEM 段起始输出跳转修正信号、跳转修正地址、发给分支预测器的反馈信号。我们设计跳转修正信号的实际发出时间为 MEM 段起始，从而防止 EX 段前递模块->EX 段分支跳转模块-> 此前译码/发射/分支预测模块的过长通路。

## 四、Forwarding 模块

EX/MEM、MEM//WB 段间寄存器输出的数据，直接前递到 ISSUE/EX 段间寄存器后的 Forward 模块。Forward 模块根据寄存器写地址、寄存器写使能、写回数据多选器的比较，输出修正后的源寄存器读数据，作为 EX 段各类计算模块的数据来源。由于我们顺序双发射的设计将 EX/MEM/WB 段分为有序的 AB 两路，前递模块在 MEM 段优先于 WB 段的原则之后，遵循 A 路优先于 B 路的原则。

为了优化时序，防止 MEM 段模块-> 前递模块->EX 计算模块的过长通路，我们的设计保证：MEM 段向 EX 段的前递只前递写回数据多选器选择 ALU 结果的情况；对于 LOAD/MUL/DIV，由发射及之前的模块保证 RAW 相关的情况下插入 Bubble 空指令；对于 CSRRD/CSRWR/CSRXCHG，在 WB 段给出 WB-flush-csr 信号冲刷整条流水线并跳转到 CSR 指令之后重新执行。

## 五、异常处理设计

每段流水线判断本段是否产生例外，若本段出现例外且此前没有记录到本条指令出现过例外，那么置起例外记录信号（ecode-we，也是向 CSR 的写入信号），并记录

产生的例外的例外码（ecode-in）。例外记录信号和例外码随指令在流水线中流动，直到 WB 段产生实际的处理信号发给 CSR。中断产生时，当周期直接尝试插入 MEM 段（MEM-interrupt），此时 MEM 段存在有效指令且 WB 段不产生其他例外导致的 WB-flush-csr 信号，则直接附着在有效指令上，并不影响该指令的实际写回。若无有效指令或有其他例外已在处理，则会在 MEM 段记录一个不断滞留 MEM 段的中断插入信号（MEM-interrupt-buf），直到这条中断信号被实际处理产生写入 CSR 的数据。

# 第四章　高速缓存设计

我们设计了支持 AXI-4 协议的指令高速缓存和数据高速缓存，以此来加速 CPU 的运行，在访存命中 cache 时可以大大提高运行速度，从而提升 CPU 的性能。同时为了适配测试环境，利用板上的外设，我们对 Icache 和 Dcache 增加了 uncache 功能，我们秉持着服用数据通路和状态机的原则，在收到 uncache 的信号后直接进入缺失状态访问主存，同时控制访问宽度和长度，来达到预期的效果，不仅如此，我们还为 cache 实现了 cacop 指令，提高了我们 cache 的可用性，更好的适配了大赛的测试环境。

## 第一节　指令高速缓存设计

### 一、存储参数配置

我们采用**两路组相联**的方式组织缓存，标签 (Tag) 与数据块 (Way) 都使用 **Xilinx 简单单端口写优先块式存储器**作为存储单元，其主要配置参数如表 4.1所示。

表 4.1　ICache 配置参数

| 参数 | 配置 |
|---|---|
| 路数 | 两路组相连 |
| 数据宽度 | 32 位 |
| 地址宽度 | 32 位 |
| 数据块大小 | 16 字节 |
| 缓存行数 | 256 行 |
| 替换策略 | LRU |

## 二、Icache 替换策略

这里采用的替换策略是**最近最少被使用换出 (Least Recent Used)** 的策略，即 LRU 替换策略,[3]LRU 寄存器记录着每路最近使用的情况。每个块的地址对应一个一位的寄存器。之所以设计成一位，是因为这里采用的是两路组相联的组织方式，对于某个地址的 LRU，要么是第一路最近被使用，要么是第二路最近被使用，比较结果恰好只有两个状态，可以被一位进行表示 (例如 LRU 为 0 时表示第一路最近被使用，1 为第二路最近被使用)。考虑到四路及以上的 LRU 的开销比较大，且二路组相联的命中率已经足够了，因此采用两路组相联，并使用一位的 LRU 寄存器记录最近使用的一路。

## 三、Icache 状态机设计

ICache 内部逻辑控制由状态机完成，其转换图如图 4.1所示：

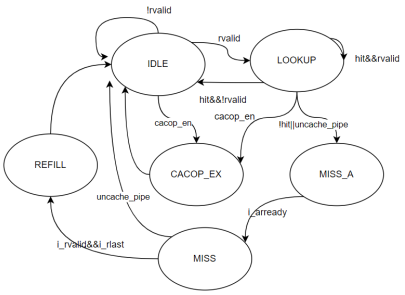


图 4.1　ICache 状态机转换图

其中各个状态的活动如下：

- **IDLE**：流水线流动，Icache 空闲；
- **LOOKUP**：检查是否命中。如果命中则继续流动，并更新 LRU；如果不命中，则进入 MISS—A 状态，开始访问主存；
- **MISS-A**：流水线不动，访问主存，进行地址握手，等待主存的 arready 到来后进入 MISS 状态；
- **MISS**：与主存的数据握手状态，当主存同时返回 rvalid 和 rlast 之后，完成一块的读取，进入 REFILL 状态；

- **REFILL**：将写使能信号置高，让读到的数据写入，并更新 LRU；
- **CACOP_EX**：进行 CACOP 操作，仅需一周期；

### 四、与流水线的交互

对于读未命中的情形，需要根据 Icache 发出的 rready 信号来选择是否暂停流水线，若 rready 为 0，则需要暂停流水线，且由于 CPU 架构是双发射，所以需要一次送出两条指令，由于可能出现数据跨 Cache line 的情况，此时只能发出一条指令，我们使用一位的 flag 信号来指示指令组的高 32 位是否有效。从而后端译码区可选择是否丢弃无效指令。

## 第二节　数据高速缓存设计

由于我们的 CPU 的 EX 和 MEM 段有两级流水，从而 Dcache 可以在 EX 段发起访存，在 MEM 段获得访存结果。

### 一、存储参数配置

与 Icache 类似，我们采用**两路组相联**的方式组织缓存，标签 (Tag) 与数据块 (Way) 都使用 **Xilinx 简单单端口写优先块式存储器**作为存储单元，其主要配置参数如表 4.2所示。

表 4.2　DCache 配置参数

| 参数 | 配置 |
|---|---|
| 路数 | 两路组相连 |
| 数据宽度 | 32 位 |
| 地址宽度 | 32 位 |
| 数据块大小 | 16 字节 |
| 缓存行数 | 256 行 |
| 替换策略 | LRU |

## 二、DCache 替换与写回策略

同 Icache 一样，我们采用了 LRU 替换策略，来提高 Dcache 的命中率。同时，我们使用了写回写分配的写入策略，在没有 L2cache 的情况下，这种策略能够最大程度提高访存性能，且由于写回状态机和读状态机是并行的，所以在触发写回状态机的时候可以并行换行，掩盖了写回的时间。

## 三、DCache 状态控制

### 1. 读状态机设计

DCache 的读逻辑由状态机完成，其转换图如图 4.2所示：

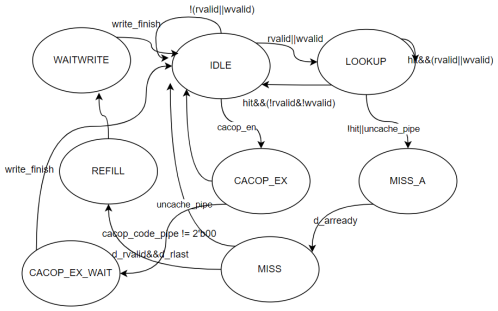


图 4.2　DCache 读状态机转换图

其中各个状态的活动如下：

- **IDLE**：流水线流动，Dcache 空闲；
- **LOOKUP**：检查是否命中。如果命中则继续流动，并更新 LRU；如果不命中，则进入 MISS—A 状态，并触发写回状态机，开始访问主存；
- **MISS-A**：流水线不动，访问主存，进行地址握手，等待主存的 arready 到来后进入 MISS 状态；
- **MISS**：与主存的数据握手状态，当主存同时返回 rvalid 和 rlast 之后，完成一块的读取，进入 REFILL 状态；
- **REFILL**：将写使能信号置高，让读到的数据写入，并更新 LRU；

- **WAITWRITE**：由于写操作可能比读操作先完成，所以需要设置等待写完成的状态；
- **CACOP_EX**：进入 cacop 指令操作，若仅为初始化操作则一周期完成，否则进入 CACOP_EX_WAIT 进入写回状态维护 cache 一致性；
- **CACOP_EX_WAIT**：进入写回状态, 维护 cache 一致性；

**2. 写回状态机设计**

DCache 的写逻辑也由状态机完成，其转换图如图 4.3所示：

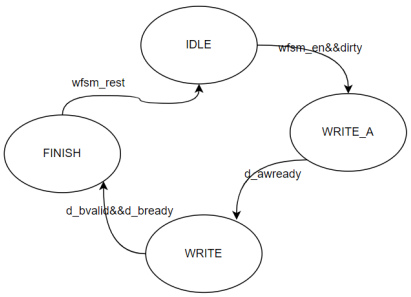


图 4.3　DCache 写状态机转换图

其中各个状态的活动如下：

- **IDLE**：不需要写回，空闲；
- **WRITE-A**：需要写回，触发写回状态机，进入写地址握手状态；
- **WRITE**：进行写数据握手；
- **FINISH**：写操作完成，向读状态机报告；

## 四、与流水线的交互

同样，当 Dcache 处于缺失状态时，我们需要暂停后端流水线，并且流水线需要能够暂存 Dcache 送出的有效数据，从而保证访存的正确性。

# 第五章　分支预测设计

## 第一节　分支预测说明

### 一、预译码部分

1. 预译码在取值完成后立刻进行，组合逻辑，给出指令类型和跳转目标，类型定义如下：

   - 00 非跳转指令
   - 01 除了 bl 和 jirl 的跳转指令
   - 10 bl
   - 11 irl

2. 预译码部分还要在分支预测出错时或缺失时给出预测的目标地址，以及预测的类型：

   - 方案：译码单元可以进行指令的静态预测：
     - 对于 01 型指令，如果分支预测器没有这条指令的信息，那么如果指令是向低地址分支的，则将 PC 置为分支 PC；对于高地址的则不做分支。
     - 对于 10 型指令，如果分支预测器没有其记录，则取指单元在此做出补救，进行分支。
     - 最后，执行单元一旦发现分支指令的预测行为与指令严格执行的结果不一致，则执行单元对于 PC 做出修改且享有最高权限。

### 二、预测部分

#### 1. PHT

1. 如果读出的 btb 表项不是跳转类指令，则不需要 pht 的结果。

2. 大小：1024 项 10 位 index 2 位 counter 20 位 tag 。

3. 每一项是一个 2 位的计数器，00 为强 untaken，01 为弱 untaken，10 为弱 taken，11 为强 taken 根据首位判断是否跳转，初始化为 01。

4. 对于所有跳转指令，都要进行 pht 的修改。

5. PHT 需要两个读口，一个写口，一个读口用于预测，另一个读口用于读出结果后结合流水线的反馈进行更新，写口用于更新。

**2. BTB**

1. 存储 br_type 和 br_target，两路 BTB0 和 BTB1，区别于 pc[2] 的奇偶性，用于同时预测两条指令。

2. 大小：参数化配置。

3. 逻辑：

   - 读取取指 pc。
   - 读取 btb 表项。
   - 判断是否命中。
   - 若命中，取出目标地址和类型。
   - 若未命中，预测顺序。

4. 对于是否跳转，需要 2 位感知机的结果。

5. 预测端输入：取指 pc 输出：br_type, br_target。

6. 反馈端输入：decoder_pc, decoder_type, decoder_target 效果：btb 的更新。

7. btb 表项：valid, tag, target, type。

## 三、反馈修正部分

修正执行流为了减小惩罚，可以在预译码段进行修正，反馈为了减小复杂读和优化时序只在执行段进行修正。

**四、超标量相关部分**

1. 一般可以给出相邻两条指令的预测结果，前一条跳转了则输出由前一条决定，否则由后一条决定。

2. 除此之外，考虑到 cache line 的情况，pc 的 2-3 位如果是 1，预测由前一条决定。

# 第六章　测试结果

## 第一节　功能测试结果

我们设计的 CPU 最终通过了初赛所需要的 1-58 条功能测试并完成了仿真和上板测试，测试结果如图 6.1所示，通过双色 LED 灯全绿并且数码管高位和低位都显示十六进制的 3A 可以知道，我们确实通过了该功能测试，并且在任意随机延迟的情况下均能通过，最终功能测试成绩为 100.00 分。

## 第二节　性能测试结果

我们同时进行了性能测试，并通过了 20 个测试中的 16 个，得到了我们的 CPU 的运行时间（其中上板未通过的时间记为 0），结果如表 6.1所示，最终我们的性能得分为 0.122，时钟频率确定为 80.0MHz

# 第七章　总结与展望

## 第一节　总结

在本次第八届全国大学生系统能力大赛龙芯杯团体赛中，我们团队紧密合作，成功设计并实现了一款基于龙芯架构的高性能 CPU。通过深入研究龙芯 32 位精简版（LA32R）指令集，我们确保了处理器对 LA32R 基准指令集的全面支持，并成功通过了功能测试，展现了卓越的兼容性和可扩展性。[2]

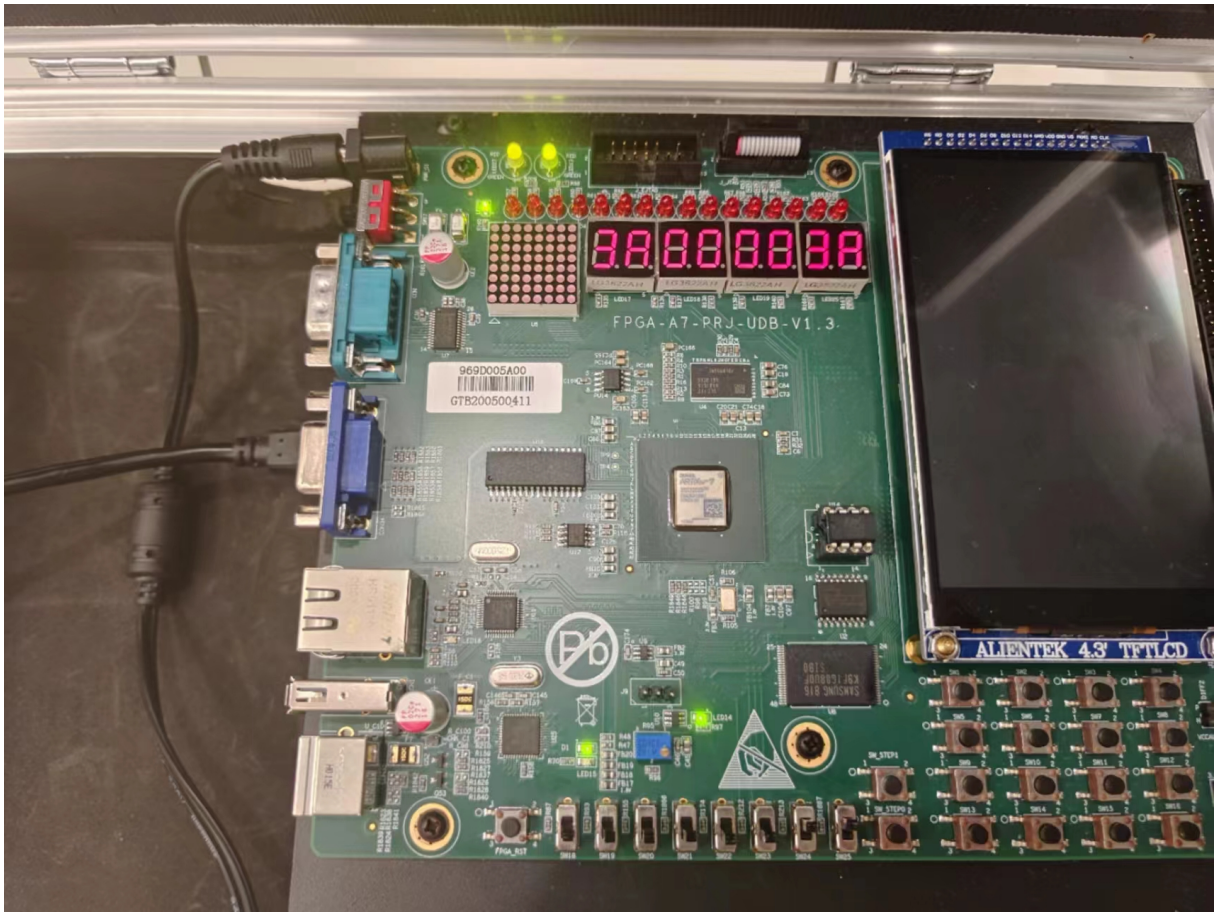


图 6.1　功能测试结果

在设计过程中，我们采用了先进的流水线技术和模块化设计思路，通过精细的流水线划分与优化，实现了指令的高效执行与资源的合理利用。同时，我们注重细节，对控制单元、流水线结构及数据通路等关键模块进行了精心设计与实现，确保了处理器的整体性能与稳定性。

此外，我们团队还克服了诸多技术挑战，如指令集兼容性问题、流水线冲突与冒险等，通过创新的解决方案和不懈的努力，最终实现了设计目标。这一过程不仅锻炼了我们的技术能力，更提升了我们的团队协作与问题解决能力。

我们团队将以本次大赛为契机，不断总结经验教训，深化技术研究与创新实践，为推动我国计算机系统设计领域的发展贡献自己的力量。

表 6.1　性能测试结果

| 测试 | CPU count | SOC count | IPC ratio |
|---|---|---|---|
| bitcount | 5c370 | 73492 | 0.9087 |
| bubblesort | 0 | 0 | 0.0001 |
| coremark | 0 | 0 | 0.0001 |
| crc32 | 0 | 0 | 0.0001 |
| dhrystone | 123481 | 16c1e8 | 0.4774 |
| quicksort | 23e180 | 2cda88 | 0.7384 |
| selectsort | 1e67c0 | 2601f6 | 0.5205 |
| sha | 18c737 | 1ef988 | 1.2091 |
| streamcopy | 19f9c | 20867 | 1.0382 |
| stringsearch | 250008 | 2e408d | 0.5510 |
| fireye_A0 | 32deb1 | 3f96a3 | 0.6447 |
| fireye_B2 | 972b4 | bcfe4 | 0.7640 |
| fireye_C0 | d5b23 | 10b2cf | 0.7365 |
| fireye_D1 | 5da990 | 75150c | 0.6139 |
| fireye_I2 | 45aae9 | 571649 | 0.8382 |
| inner_product | e589ac | 11eecbe | 0.6387 |
| lookup_table | 3a713a | 490e46 | 0.7178 |
| loop_induction | 8c1175 | af167b | 0.7055 |
| my_memcmp | 2e02c1 | 39841a | 0.7772 |
| minmax_sequence | 0 | 0 | 0.0001 |

## 第二节　展望

展望未来，我们将继续深化对 CPU 架构及计算机系统设计的研究，不断探索新技术、新方法，以推动计算机系统性能的进一步提升。具体而言，我们计划从以下几个方面进行拓展：

**1. 性能优化**

针对当前设计的不足之处，我们将进一步优化流水线结构，提升指令执行效率与资源利用率。同时，考虑引入多发射、乱序执行等高级技术，以应对更复杂的应用场景。

**2. 扩展性与灵活性**

为了满足不同用户的需求，我们将继续提升处理器的扩展性与灵活性。通过增加更多的接口与扩展槽位，支持更多的外设与功能模块，为用户提供更加丰富的选择与定制空间。

**3. 功耗与能效比**

随着绿色计算理念的普及，我们将更加注重处理器的功耗与能效比。通过采用先进的低功耗设计技术与优化算法，降低处理器的能耗，提升整体能效比，为可持续发展贡献力量。

# 参考文献


[1] Alex Forencich. *verilog-axi*. `https://github.com/alexforencich/verilog-axi`. 访问日期: 2024-7-15. Dec. 2021.

[2] 姚永斌. 超标量处理器设计. 北京: 清华大学出版社, 2014.

[3] 汪文祥, 邢金璋. *CPU* 设计实战. 北京: 机械工业出版社, 2021.