NSCSCC lucky-MIPS 设计报告

中山大学
daydayganhuo 队
黄静雯　郭家怡　周沁茗

Monday 16th August, 2021

目录

1 设计简介

我们的 CPU 架构为单发射五级流水，分为取指、译码、执行、访存以及写回五个流水级。整体的流水线结构见图4*。CPU 对外封装为 AXI 接口，未使用 AXI 接口转换桥，使用自己设计的 AXI 接口。

CPU 为哈佛结构，包括指令 Cache 和数据 Cache。指令 Cache 设计为 16KB，四路组相连，Cache 行的大小为 64 字节。实现 receive_buffer 信号缓冲，单周期内返回指令数据。数据 Cache 设计为 16KB，四路组相连，cache 行的大小为 32 字节。

实现了 32 项，4KB 固定页大小的 TLB，命中时可当拍返回物理地址，实现了 4 条 TLB 指令和 3 种 TLB 异常，通过官方的 10 项 TLB 测试。

共实现了 69 条 MIPS32 release1 中的指令，指令集见附录B。

实现了 15 个 CP0 寄存器，CP0 寄存器见附录支持以下几种中断和例外：

1. 中断：2 个软件中断、6 个硬件中断、1 个计时器中断。
2. 例外：
   (a) 地址错例外：取指或读数据（AdEL）、写数据（AdES）。
   (b) 整型溢出例外（OV）。
   (c) 系统调用例外（SYS）。
   (d) 断点例外（BP）。
   (e) 保留指令例外（RI）。
   (f) TLB 缺失（TLB Refill）。
   (g) TLB 无效（TLB Invaild）。
   (h) 修改 TLB 只读页（TLB Modified）。

*为简化图片复杂度，没有画出数据前推通路以及流水线寄存器

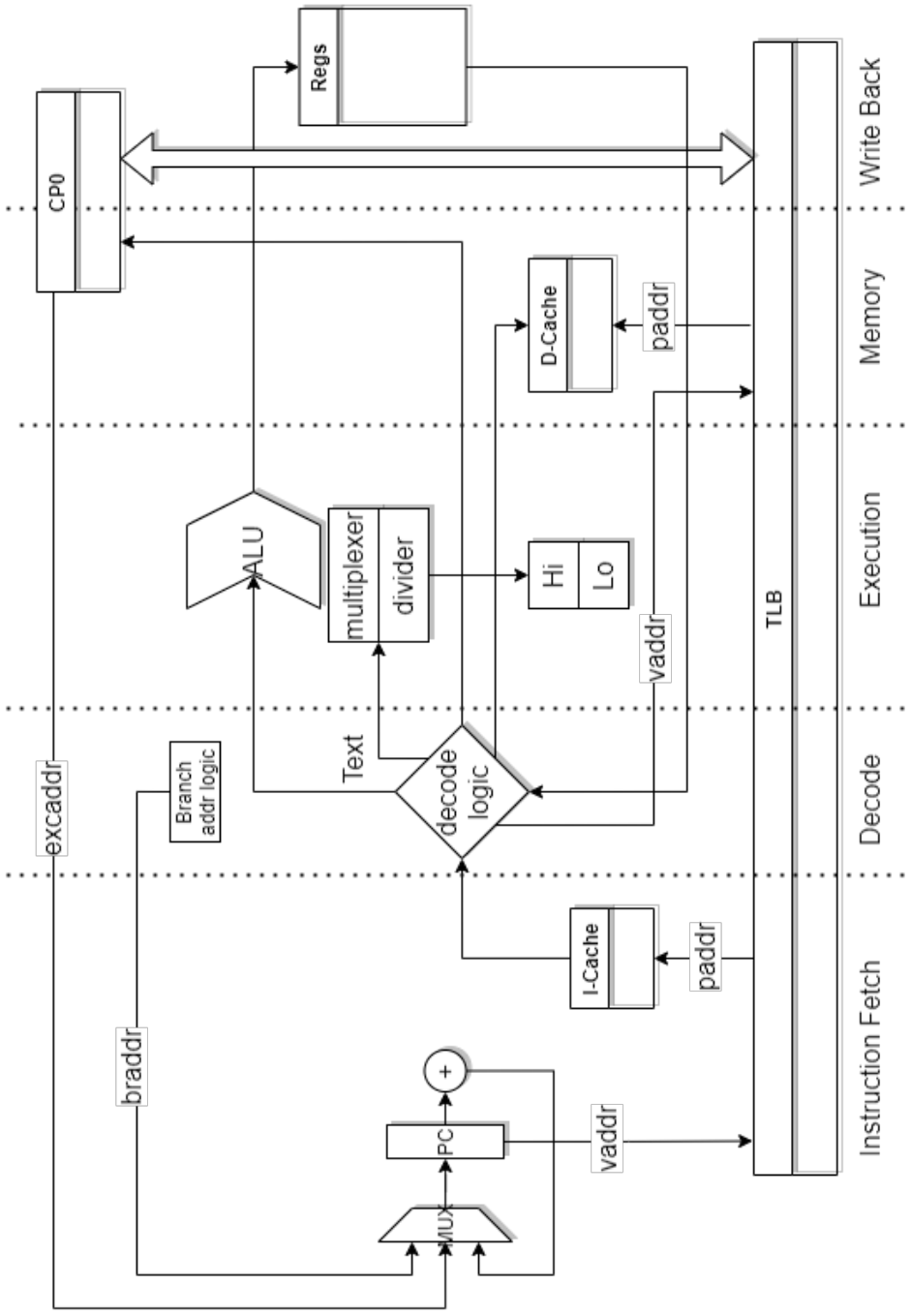
Regs
CP0
Write Back
D-Cache
paddr
Memory
ALU
multiplexer
divider
Hi
Lo
vaddr
TLB
Execution
Text
decode
logic
excaddr
Branch
addr logic
Decode
I-Cache
paddr
braddr
+
PC
vaddr
MUX
Instruction Fetch

图 1: CPU 架构

2 详细设计方案

2.1 流水级

取指 首先判断 PC 位于 mapped 段还是 unmapped 段，通过地址转换获得取指的物理地址，取指级根据此物理地址取出指令。取指级使用状态机判定当前与指令 Cache 交互情况。取指状态机见图2[†]。

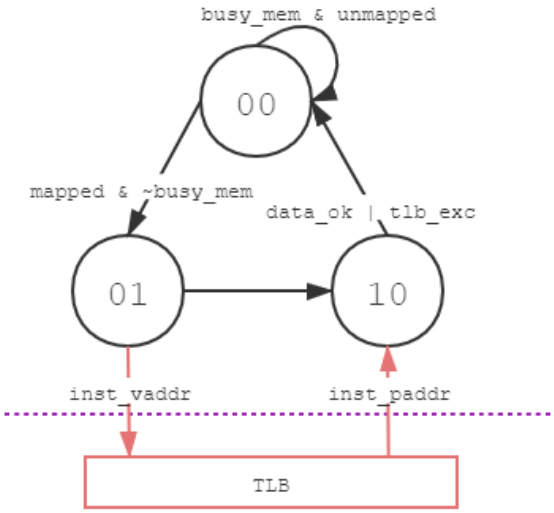

图 2: 取指状态机

取指级可能出现的特殊情况：

1. 重复取指：当上一条取指地址与当前取指地址相同时（相邻两条指令），不发出取指请求，直接将内部存储指令码的变量维持原值，作为当周期的指令码输出。

2. 当拍无法返回指令数据（取指忙碌）：由取指级发出的暂停流水线信号，当取指级忙碌时，向后续各级发送暂停请求信号。

   若取指忙碌时，未暂停后续各级流水线，可能出现当此次取指未命中，需要多周期时，之前的指令在流水时出现异常，需要清空流水线，则 pc_next（取指忙碌指令的下一条指令的 PC）会被覆盖，若此时 addr_ok 已经返回（state=10），那么无法发出新一轮的取指请求，造成新的 pc_next 无法进行取指。等到 data_ok 时，并非 excaddr 对应的指令，但却被当作是 excaddr 对应的指令，导致无法正常进入异常处理服务程序。

   因此当取指级忙碌时，后续所有流水线都暂停，不会出现取指级成功请求后还有其他级发来的暂停请求，即其他级发出暂停请求时，一定是在刚更新流水的一个时钟周期内发出，首轮请求就能被取消。

3. 地址错例外：取指（AdEL）

4. TLB 缺失例外（TLB Refill）。

5. TLB 无效例外（TLB Invaild）。

[†]busy_mem：当前正在访问 Cache/Memory ； unmapped：当前取指虚地址不需要 TLB 映射；mapped：当前取指虚地址需要 TLB 映射；data_ok：返回指令数据；tlb_exc：访问 TLB 出现异常。

译码 译码级对指令进行译码，生成控制信号，接收旁路数据并准备好源操作数，产生跳转目标地址并送到取指级更新 PC。译码级为纯组合逻辑。

译码级可能出现的特殊情况：

1. 系统调用例外（SYS）。
2. 断点例外（BP）。
3. 保留指令例外（RI）。
4. 地址错例外：读数据（AdEL）、读数据（AdEL）。由于我们在译码级产生访存虚地址，因此可能出现地址错例外。

执行 执行级计算算术运算指令、逻辑运算指令和移位指令的结果。核心模块为 ALU、乘除法器和 HILO 寄存器。乘除法器我们调用了 Xillinx 的 IP 核实现，乘法可在 5 个时钟周期内返回结果，除法在 10 个时钟周期内返回结果。

执行级可能出现的特殊情况：

1. 乘除法多周期指令需要请求流水线暂停。
2. 整型溢出例外（OV）。

访存 访存级访问存储器、处理异常。访存级根据前面的流水级传过来的异常码判断有无异常，有异常则在下一拍清空流水线并进入异常处理程序，无异常则正常执行当前指令的访存操作。与取指相同，进行地址转换后，访存级使用状态机判定当前与数据 Cache 交互情况。使用组合逻辑完成地址固定映射。

访存状态机见图3[‡]。

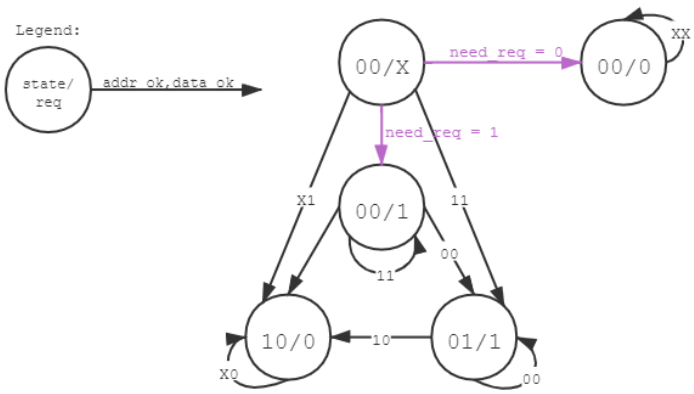

图 3: 取指状态机

访存级的特殊情况：访存忙碌时，需要请求暂停取指、译码、执行级。访存暂停优先级最高，因此当该级访存忙碌多个时钟周期时，数据返回前，都经过流水线寄存器向写回级发送空指令信号。

协处理器 0 访存级根据前面的流水级传过来的异常相关信号更新 CP0 寄存器，CP0 寄存器均能在一拍内完成写回，因此在处理异常时不需要暂停，当拍写回 CP0，下一拍可以进入异常处理程序。

[‡]state：当前状态；req：访存请求；need_req：下一拍需要发出访存请求；

写回 写回级将数据写入通用寄存器堆。

2.2 数据通路

我们在访存级进行异常处理,这样可以消除流水线和 CP0 的数据冲突。我们在执行级就写回 HILO,这样可以消除 HILO 的数据冲突。尽可能的简化 bypass 通路可以提升性能和主频，也降低了设计的复杂度，不容易出现错误。为处理数据冲突问题，CPU 引入了旁路机制，各个阶段的运行结果都会被前推到译码阶段。当通过前推无法解决数据冲突时 (如 load-use 冲突)，流水线会暂停一拍等待结果返回。

2.3 Cache

指令 Cache

特性

1. 指令 Cache 的容量为 16KB，四路组相连映射，cache 行的大小为 64 字节。
2. 指令 Cache 采用并行访问 Tag 和 data 的形式，根据产生的 hit 信号实现对命中数据的选择。
3. 指令 Cache 采用虚拟地址 Index，实际物理地址 Tag 的访问模式
4. 指令 Cache 采用伪随机替换算法。
5. 指令 Cache 对外采用 AXI 接口，支持猝发传输，一次传输请求可以处理 64 字节的数据，提高总线利用率。

访问过程 判断 cpu 是否发出访存请求信号，若是则根据 Index 字段以及 hit_line_num 从 cache 指定行中传输指定数据，并赋给 CPU 端口数据输入，实现单周期访问命中并返回。若没有命中，则向 Memory 发起地址访问请求，并进入第二状态。在第二状态中，检测到 Memory 传来的地址握手信号，将 mem_req 信号拉低，等待地址传输完成信号 line_data_ok，未传输完成，继续在该信号等待，传输完成，根据替换信号和 Offset 信号从指令 Cache 中选取访问数据并传向 CPU。

数据 Cache

特性

1. 数据 Cache 的容量为 16KB，四路组相连，Cache 行的大小为 32 字节。
2. 数据 Cache 采用并行访问 Tag_ram 和 data_ram 形式，同时获取 Tag 和 data，通过判断器确定是否命中，命中则在当周期将数据直接返回。
3. 数据 Cache 采用虚拟地址 Index，实际物理地址 tag 的访问模式。
4. 数据 Cache 采用伪随机替换算法，利用计数器实现未命中替换行号加 1 的处理。
5. 数据 Cache 对外采用 AXI 接口，支持猝发传输，一次传输请求可以处理 32 字节的数据，提高总线利用率。

访问过程

1. 读数据 Cache

   (a) 第一拍：将地址送至 Cache RAM 端口处，等待一拍才进行时钟周期判断。

   (b) 第二拍：从 Cache 中读出相应的 Tag 字段，与物理地址中的 Tag 字段进行对比，如果相等，并且该 Cache 行的有效位为 1，表示此次读请求命中，根据物理地址中的 Offset 字段可以读取 Cache 中的某一个字并传向 CPU。若访问不命中，则进入第三拍，根据伪随机替换算法得到 Cache 中的替换行 ID，判断其中的 dirty 是否为高电平，若为高电平，则进入第四拍，否则直接跳转进第三拍，进行 MEM 的数据传输。

   (c) 第三拍：向 Memory 发起访问请求，传输地址是缺失 Cache 行的起始地址，并且一次访问一整行数据，也即一次访问传输 8 个字（不采取关键字优先策略，故一次传输需要若干个时钟周期才能将数据全部返回）。将生成的 Cache 行信息填入 Cache 对应的 Index 位置，完成后进入第二拍，进行命中数据的访问，并根据物理地址中 Offset 字段查询 CPU 所访问的具体数据，并将该数据返回给 CPU。

   (d) 第四拍：向 Memory 发起写访问请求，传输地址是被替换行的 ID，一次写回一整行数据，也即一次传输 8 个字（需要若干个周期才能将数据全部返回）。完成后进入第三拍，进行总线读访问请求。

2. 写数据 Cache

   (a) 第一拍：将地址送至 Cache RAM 端口处，等待一拍才进行时钟周期判断。

   (b) 第二拍：从 Cache 中读出相应的 Tag 字段，与物理地址中的 Tag 字段进行对比，如果相等，并且该 Cache 行的有效位为 1，表示此次写请求命中 Cache，根据物理地址中的 Offset 字段和 size 字段可以确定写字节使能信号。若访问不命中，则进入第三拍，根据伪随机替换算法得到 Cache 中的替换行 ID，判断其中的 dirty 是否为高电平，若为高电平，则进入第四拍，否则直接跳转进第三拍，进行 Memory 的数据传输。

   (c) 第三拍：向 Memory 发起访问请求，传输地址是缺失 Cache 行的起始地址，并且一次访问一整行数据，即一次访问传输 8 个字（不采取关键字优先策略，故一次传输需要若干个时钟周期才能将数据全部返回）。将生成的 Cache 行信息填入 Cache 对应的 Index 位置，完成后进入第二拍，进行命中数据的访问，将生成的 Cache 行信息填入 Cache 对应的 Index 位置，并将 CPU 传来的数据填入对应的 Cache 行中。

   (d) 第四拍：向 Memory 发起写访问请求，传输地址是被替换行的 ID，一次写回一整行数据，即一次传输 8 个字（需要若干个周期才能将数据全部返回）。完成后进入第三拍，进行总线读访问请求。

2.4 内存管理

MIPS32 Virtual Address Space				
kseg3	0xE000.0000	0xFFFF.FFFF	Mapped	Cached
ksseg	0xC000.0000	0xDFFF.FFFF	Mapped	Cached
kseg1	0xA000.0000	0xBFFF.FFFF	Unmapped	Uncached
kseg0	0x8000.0000	0x9FFF.FFFF	Unmapped	Cached
useg	0x0000.0000	0x7FFF.FFFF	Mapped	Cached

图 4: 虚地址空间

Unmapped 段 直接使用组合逻辑将虚地址高三位抹零就得到物理地址。

Mapped 段 根据虚地址查询 TLB 得到物理地址，命中当周期返回物理地址。TLB 查找部分使用纯组合逻辑。若不命中、查询到的 TLB 项无效、访存时修改 TLB 只读页，都会触发异常，进入到异常处理程序。TLB 项的更改由 TLBWI、TLBWR 指令操作。

3 性能提升尝试

详细的性能分数见附录D

3.1 优化取指状态机

为减小关键路径的延时，对取指状态机[§] 重新进行设计。

1. cstate = 00

   (a) 当前访存忙碌

   - nstate=00，保持初态；
   - inst_req=0，不请求取指；
   - busy_if=0，取指不忙碌；

   (b) 若要访问 kseg0/1 区间，且访存不忙碌

   - nstate=00；- inst_req，若取指地址异常 if_exc=1，则不请求取指 inst_req=0，反之，则请求取指 inst_req=1；inst_req=0；
   - busy_if:
     - 若不发出请求，则不忙碌 busy_if=0；
     - 发出请求后，在 inst_data_ok 返回前，一直处于取指忙碌 busy_if=1；inst_data_ok 返回后，不忙碌 busy_if=0；

   (c) 若要访问非 kseg0/1 区间，且访存不忙碌

   - nstate=01，即将进入地址转换状态；
   - inst_req=0，不请求取指；
   - busy_if=1，取指忙碌；

2. cstate = 01

   - 向 TLB 请求地址转换，并将 TLB 返回的 paddr、miss、valid 等有效信息在状态 01 转换为 10 的时刻进行采样保存；
   - nstate=10；
   - inst_req=0；
   - busy_if=1，取指忙碌；

3. cstate = 10

   (a) 若 TLB 地址转换成功，则等待 inst_addr_ok、inst_data_ok 的返回；

   - nstate：inst_data_ok 返回成功，nstate=00；等待中 nstate=10；
   - inst_req ： inst_addr_ok 返回成功，inst_req=0；等待中 inst_req=1；
   - busy_if：inst_data_ok 返回成功，busy_if=0；等待中 busy_if=1；

   (b) 若 TLB 地址转换失败

   - nstate=00，回到初态；
   - inst_req=0，不请求取指；
   - busy_if=0，非取指忙碌；

[§]次态 nstate、现态 cstate、取指请求 inst_req、取指忙碌 busy_if

3.2 尝试为数据 Cache 添加 Write Buffer

Write Buffer 能够处理数据 Cache 替换行写回的请求。可从两个方面提升性能：

1. 当数据 Cache 缺失时，无需等待 AXI 总线完成写请求，就能进行缺失行的读取，减少了数据 Cache 缺失造成的延时。

2. 当数据 Cache miss 时，若缺失的行在 Write Buffer 中，则可以在一拍内将缺失行从 Write Buffer 中读出，不需要向 AXI 总线发起缺失行的读取，减少了该情况下的延时。

Write Buffer 本质上是一个处理数据 Cache 缺失行的缓冲模块，该模块实现了 axi 接口的读写通道分离。当替换行需要写回时，只要 Buffer 未满，就可以将替换行在一拍内写入。之后，Buffer 会在 AXI 写通道处于空闲的时候将其中的替换行依次写进内存。

这里需要注意的是写回通道上 uncached 类型的写访问请求和 Write Buffer 写回之前的优先级问题。我们设立了一个暂停信号，在检测到 uncached 类型的写访问请求之后，会将 stall 信号拉高，暂停 Write Buffer 向内存的写回操作，等待 uncached 写请求访问结束之后，再将 stall 信号拉高，重新开始 Buffer 的写回操作。此外，若写回的数据行已经存在于 Buffer 中，则在数据行中回出现新旧两行数据，因此，我们在 Buffer 中的每一行添加一位有效位，在检测到待写入数据行已存在于 Buffer 时，会将该数据行置为无效，等待新的数据行写入。在我们经过尝试之后，我们发现添加了 TLB 之后的 buffer 在性能测试时无法通过，初步认为是写回通道分配问题。由于时间关系，我们最终放弃了调试添加 Write Bufferr 的数据 Cache，选择了初赛提交的数据 Cache 版本。

3.3 静态双发射及分支预测尝试

我们尝试仿照 Arm Cortex-A53 架构设计一个静态双发射并具有分支预测模块的 CPU，但由于时间较紧，我们仅仅只是查看了关于 Arm Cortex-A53 的资料，评估了实现双发射的复杂度和调试难度，最终还是决定改进我们的单发射 CPU 以提升性能。

4 运行 PMON

尝试运行 PMON 但失败，初步推测是由于入口地址错误导致的。但时间紧迫，我们没有进一步去检查 PMON 运行失败的原因。

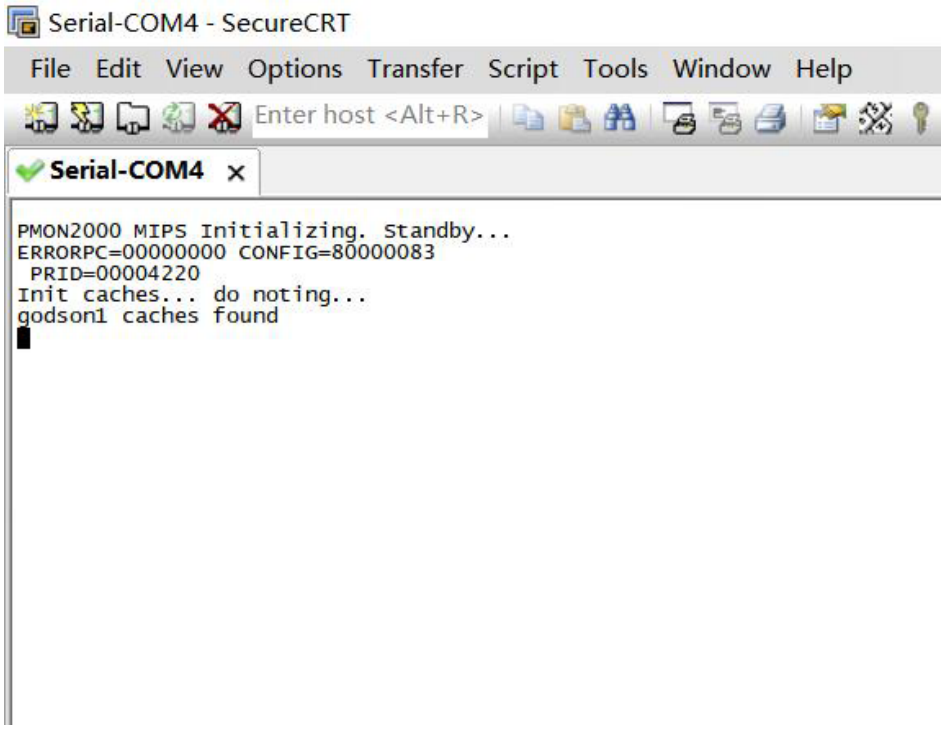

图 5: 运行 PMON 失败

参考文献

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

[2] MIPS Technologies, Inc..MIPS® Architecture For Programmers Volume III: The MIPS32® and microMIPS32™ Privileged Resource Architecture[M].MIPS Technologies, Inc.:955 East Arques Avenue,2011:23-194.

[3] MIPS Technologies, Inc..MIPS® Architecture For Programmers Volume II-A: The MIPS32® Instruction Set[M].MIPS Technologies, Inc.:955 East Arques Avenue,2011:33-286.

[4] MIPS Technologies, Inc..MIPS32® Instruction Set Quick Reference: The MIPS32® Instruction Set[M].MIPS Technologies, Inc.:955 East Arques Avenue,2011:1-2.

[5] Wei, J. and Guo, W., 2019. Ji suan ji xi tong she ji. Beijing: Dian zi gong ye chu ban she.

[6] Patterson, D. and Hennessy, J., 2021. Computer organization and design. Amsterdam: Elsevier.

Appendices

A 声明与致谢

部分代码参考 2020 年复旦大学 2 队vanilla-cpu开源代码以及 2019 年中国科学院大学 2 队nscscc2019ucas开源代码。在此对他们表示感谢。

参赛过程中我们的指导教师粟涛老师给我们提出了许多宝贵的建议。参加过 2020 年龙芯杯比赛的刘嘉琦师兄、何伟亮师兄、林金煌师兄也向我们提供了许多帮助，我们在此表示衷心的感谢。

B CPU 指令集

算术运算指令（17 条）：ADD, ADDI, ADDU, ADDIU, SUB, SUBU, SLT, SLTI, SLTU, SLTIU, MUL, MULT, MULTU, DIV, DIVU, CLO, CLZ

逻辑运算指令（8 条）：AND, ANDI, OR, ORI, XOR, XORI, NOR, LUI

移位指令（6 条）：SLL, SLLV, SRL, SRLV, SRA, SRAV

分支跳转指令（12 条）：BEQ, BNE, BGEZ, BGTZ, BLEZ, BLTZ, BGEZAL, BLTZAL, J, JAL, JR, JALR

数据移动指令（4 条）：MFHI, MFLO, MTHI, MTLO

访存指令（12 条）：SB, SH, SW, LB, LBU, LH, LHU, LW, LWL, LWR, SWL, SWR

自陷指令（2 条）：SYSCALL, BREAK

特权指令（8 条）：ERET, MTC0, MFC0, TLBP, TLBR, TLBWI, TLBWR, CACHE（为空）

C CP0 寄存器列表

表 1: CP0 寄存器列表

Select	Number	Name
0	0	Index
0	1	Random
0	2	EntryLo0
0	3	EntryLo1
0	4	Context
0	6	Wired
0	8	BadVAddr
0	9	Count
0	10	EntryHi
0	11	Compare
0	12	Status
0	13	Cause
0	14	EPC
0	15	PRId
0	16	Config

D lucky-MIPS 性能分数

表 2: 性能提升过程记录

日期	主频	性能分数	IPC 比值	备注
2021/08/03	62MHz	27.633	23.920	添加了指令 Cache 和数据 Cache
2021/08/10	65MHz	31.080	23.880	优化暂停机制
2021/08/11	70MHz	33.356	23.794	添加 Write Buffer
2021/08/15	61MHz	28.835	23.953	添加了 TLB 模块和部分指令，优化关键路径
2021/08/16	66MHz	31.264	23.876	优化关键路径