

基于 MIPS 指令集的六级流水线 CPU 设计报告

哈尔滨工业大学（深圳）1队

胡博涵、陈泓佚、施杨

一、设计简介

该设计是基于 MIPS32 的 57 条核心指令集的 CPU，不同于传统的五级流水线结果，该设计使用六级流水（IF1、IF2、ID、EX、MEM、WB）实现了基本的整型算术运算，逻辑运算，访存，跳转，软中断处理，异常处理等功能。

使用 Block RAM 实现了 1 级 16KB 的二路组相连指令 Cache 和直接映射的数据 Cache。

使用基于 2 位饱和计数器和分支历史记录表动态分支预测方案[1]以进一步提升性能。

CPU 运行频率为 109.091MHz。

通过全部功能测试点，记忆游戏（需按官方论坛修改 confreg 模块）和全部性能测试点，性能得分约 54 分。

二、设计方案

（一）总体设计思路

作品采用 MIPS 指令集架构，支持大赛官方提供的 57 条指令，中断和 6 种异常。使用 AXI 接口进行封装。使用流水线、Cache 及分支预测等优化方法进行性能的提升。

总体结构为单发射六级流水线，分为取指 1(IF1)、取指 2(IF2，或称 ICache)、译码和寄存器访问(ID)、执行(EX)、访存(MEM)、写回(WB)六个阶段。采用旁路和暂停流水线的方法解决数据相关问题，采用 2 路组相联指令 Cache 和直接映射数据 Cache，采用 2 位饱和计数器和分支历史信息表实现分支预测。

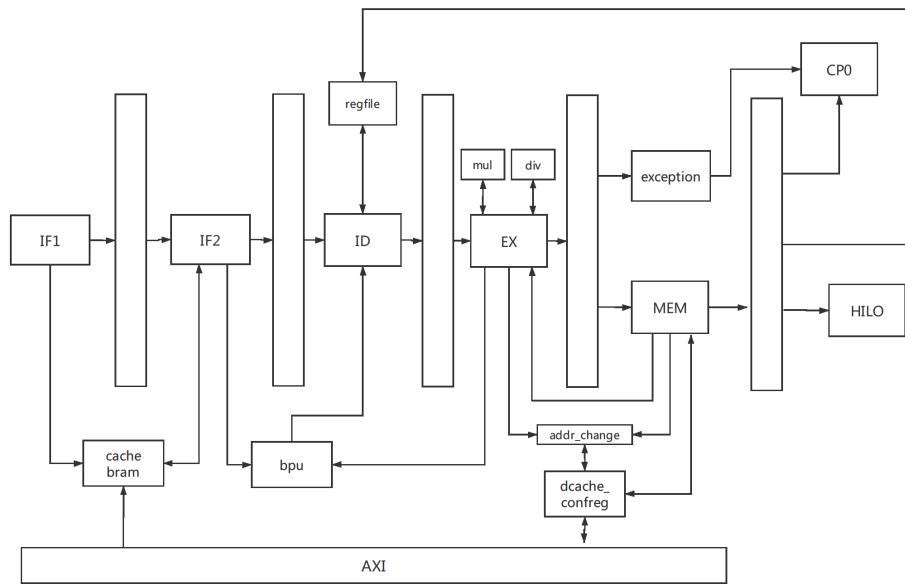


图 1: 总体设计框图

(二) 部分模块设计方案

1. ID 模块设计[2]

(1) 输入输出信号说明:

表 1: ID 模块输入信号说明

输入信号	信号来源	信号意义
pc_i	IF2/ID	当前指令地址
inst_i	IF2/ID	当前指令
ex_opcode_i	EX	ex 级当前指令的操作码 判断 ex 级当前是否是访存指令 用于处理数据相关
ex_we_i/ex_wdata_i/ ex_waddr_i	EX	ex 级写寄存器堆相关信号 用于处理数据相关
mem_we_i/mem_wdata_i/ mem_waddr_i	MEM	mem 级写寄存器堆相关信号 用于处理数据相关
operand1_i/operand2_i	Regfile	从寄存器堆读取的 操作数 1 和操作数 2
this_is_delayslot_i	ID/EX	ID 级是延迟槽指令

表 2: ID 模块输出信号说明

输出信号	信号去向	信号意义
operand1_re_o/operand2_re_o	regfile	读寄存器堆使能
operand1_addr_o/operand2_addr_o	regfile	读寄存器号
branch_condition	ID/EX	跳转条件类型
branch_addr_type	ID/EX	跳转地址类型
branch_return_addr_o	ID/EX	“跳转并链接”指令返回地址
opcode_o	ID/EX	操作码
alu_type_o	ID/EX	计算类型选择
operand1_o/operand2_o	ID/EX	操作数 1/操作数 2
waddr_o	ID/EX	写寄存器号
we_o	ID/EX	写寄存器堆使能
next_is_delayslot_o	ID/EX	下一条指令位于延迟槽
this_is_delayslot_o	ID/EX	这是一条延迟槽指令
exception_o	ID/EX	异常流输出
pause	mycpu_top	暂停流水线

(2) ID 模块总体设计思路:

根据 MIPS 指令集, 指令可大致分为 3 类: R 型、I 型和 J 型指令, 除部分特殊指令需要单独制定译码规则外, 对于常规的算术指令、跳转指令和存储器访问指令, 不同类型指令之间可通过指令的 31-26 位的 OPCode 进行区分。

R 型指令的 31-26 位均为 0, 通过 funct6 字段进行唯一的区分,MUL 指令、寄存器移动指令产生的译码信号需要进行单独的处理。

表 3: R 型指令的字段分布

31:26	25:21	20:16	15:11	10:6	5:0
OPCode 000000	rs	rt	rd	00000 / shamt	funct6

I型指令可以通过 31-26 位进行唯一的区分（除 bltz/bltzal/bgez/bgezal/bal 指令需要通过 rt 和 rs 字段进一步区分外）

表 4: I型指令的字段分布

31:26	25:21	20:16	15:0
OPCode	rs	rt	imm

表 5: J型指令的字段分布

31:26	25:0
OPCode	IMM

此外，异常相关的指令和协处理器访问指令，需要制定特定的译码规则。

此阶段除指令译码外，还负责取回或生成指令操作数，供下一阶段 EX 使用。同时在此阶段，需要解决数据相关问题。若存在非 Load 类数据相关问题，采用旁路策略进行解决，若为 Load 类数据相关，则采用暂停流水线的方式进行解决。

2. EX 模块设计

(1) 输入输出信号说明：

表 6: EX 模块输入信号说明

输入信号	信号来源	信号意义
inst_addr	ID/EX	当前指令地址
inst_i	ID/EX	当前指令
opcode_i / alu_type	ID/EX	用于指明当前 ALU 的操作和操作类型
operand1 / operand2	ID/EX	ID 阶段读出的操作数
we_i	ID/EX	该指令是否需要写回寄存器
operand1_i / operand2_i	regfile	从寄存器堆读取的操作数 1 和操作数 2
hi_i / lo_i	hi_lo	hi_lo 寄存器的输入
mul_result_i / mul_ready	mul	与乘法模块的连接 乘法结果及乘法结束信号
wb_hi_i / wb_lo_i / wb_whilo_i	WB	来自 WB 级的 hi, lo 旁路输入
mem_hi_i / mem_lo_i / mem_whilo_i	MEM	来自 MEM 级的 hi, lo 旁路输入
wait_dcache / dcache_addr	dcache	用于保存造成 dcache 缺失的地址
mem_cp0_we / mem_cp0_w_addr /	MEM	来自 MEM 级的 CP0 旁路

mem_cp0_wdata		
wb_cp0_we / wb_cp0_w_addr / wb_cp0_wdata	WB	来自 WB 级的 CP0 旁路
cp0_data_i	CP0	来自 CP0 的数据
div_result_i / div_valid	div	来自除法模块的结果
branch_condition	ID/EX	跳转条件类型
branch_addr_type	ID/EX	跳转目标地址类型
branch_return_addr	ID/EX	“跳转并链接”指令返回地址

表 7: EX 模块输出信号说明

输出信号	信号去向	信号含义
mul_sel/mul_A/mul_B	mul	mul 模块使能/被乘数/乘数
cp0_r_addr_o / cp0_we_o / cp0_w_addr_o / cp0_data_o	EX/MEM	cp0 读写相关信号
we_o/wdata_o	EX/MEM	写回阶段相关信号
hi_o/lo_o/whilo_o	EX/MEM	写 HILO 相关信号
div_sel,div_sign	DIV	除法模块使能/是否是有符号 除法
mem_addr_o	EX/MEM	访存地址
exception_o	EX/MEM	异常流
pause	mycpu_top	暂停信号
ex_is_branch	BPU	处在 ex 阶段指令是否为跳转 指令（更新 BTB 所用）
branch	IF1	分支预测错误标记
branch_addr	IF1	实际分支地址

(2) EX 模块总体设计

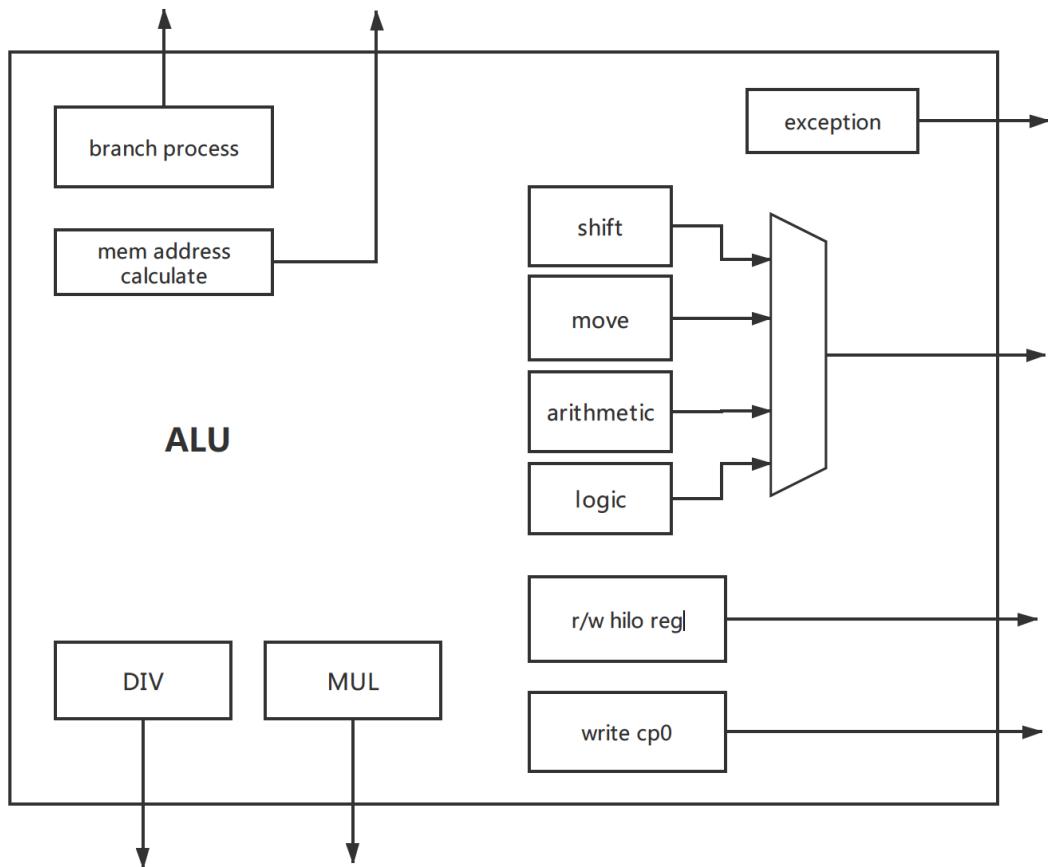


图 2: ex 模块框图

在 ex 中有多个模块，分别执行不同的操作(在代码中未具体划分模块，仅为设计思想):

跳转处理单元；

运算单元：四种类型的运算 (shift,move,arithmetic,logic) 分开进行，最后根据译码段输入的运算类型选择输出的结果；

读写 HILO 寄存器单元：包含数据相关处理；

写 cp0 单元；

乘除法单元：使能外部乘除法模块，产生暂停流水线信号，其中乘法采用片上 DSP 实现；

异常单元：产生溢出异常，加入到异常流中输出到下一级；

访存地址计算单元；

3. 指令 Cache 及 IF1、IF2 的设计

(1) 设计动机

由于访问主存延迟过大，严重影响 CPU 的性能，需要指令 Cache。而指令 Cache 采用 Block RAM 实现。在给出地址的情况下，Block RAM 需要经过一个时钟周期的延迟，才能返回数据，在 Cache 连续命中的情况下，需要 2 个周期才能取出一条指令，严重影响了取指效率。为解决这个问题，将 IF 阶段拆分为 2 级流水：取指 1 (IF1) 和取指 2 (IF2, ICache 访问)。

(2) 总体设计方案

采用的方案是二路组相联方案，块大小为 4 个字，替换上一次没有访问的块的替换策略。Cache 存储体使用 Block RAM 进行实现。

表 8: ICache Line 各个字段分布

valid	主存地址标记	word1	word2	word3	word4
147	146: 128	127: 96	95: 64	63: 32	31: 0

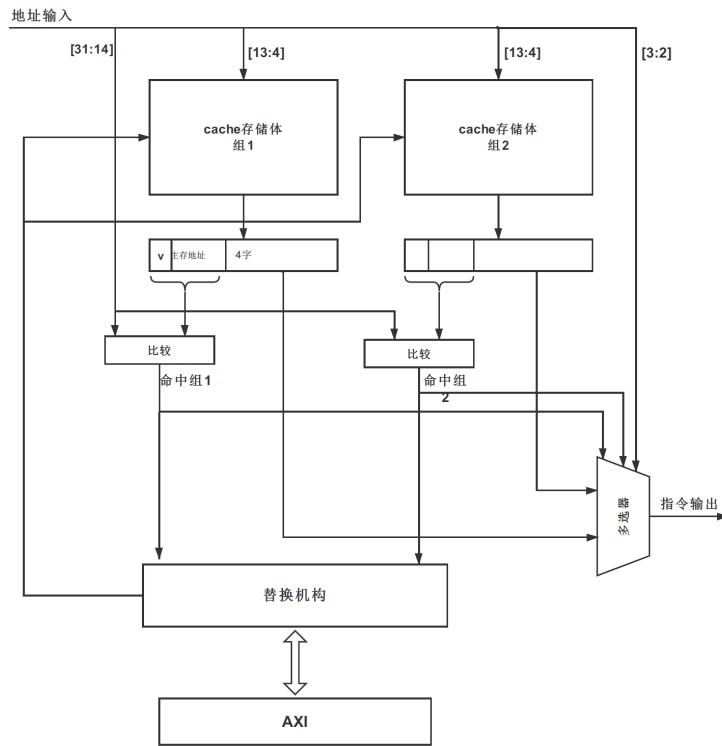


图 3: icache 内部结构

IF1 阶段生成下一条 PC 地址。输入信号包括流水线跳转、暂停、清空信号和来自 IF2 级的 Cache 缺失信号。IF1 通过一系列组合逻辑，计算出下一条指令的地址。

为避免将地址先送入流水线寄存器，再送到 ICache 造成的额外周期开销，地址由 IF1 流

水线寄存器直接送入 ICache，而不经过 IF1/IF2 寄存器。当前指令的 PC 地址则送至 IF1/IF2 流水寄存器。在下一个时钟上升沿，PC 被 IF1/IF2 寄存器锁存，Cache 读出的 Cache Line 也同时到达 IF2。

IF2 阶段对 Cache Line 中的一系列信息进行处理。首先比较输出的 Tag，判断 Cache 是否命中。若命中，则不暂停流水线；若缺失，则暂停流水线，并向 IF1 发出 ICache 缺失信号。需要注意的是，需要将 IF2 阶段的地址重新送回 IF1。

在命中时，IF1 通过 IF2 送来的 PC 计算出 PC+4 并送至 ICache，或选择例外地址/跳转地址；在未命中时，IF1 接收流水线暂停信号，选择直接将 IF2 送回的造成缺失的 PC 作为 ICache 的访问地址，并通过 AXI 总线对 RAM 发起长度为 4 的突发读请求（读取 4 个字到缓存中，再一次性写入到 Block RAM），数据写入 ICache 后，撤销流水线暂停信号。

在命中或突发传输结束后，IF2 根据 PC 的地址，选择对应的 Block，将指令送入下一级流水线寄存器。

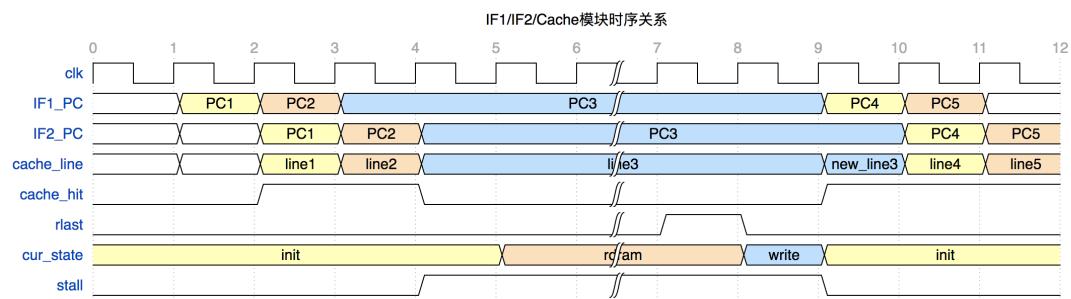


图 4: icache 时序关系

IF1 和 if1_if2 在 if1.v 文件中实现，IF2 和 icache 在 icache.v 文件中实现。

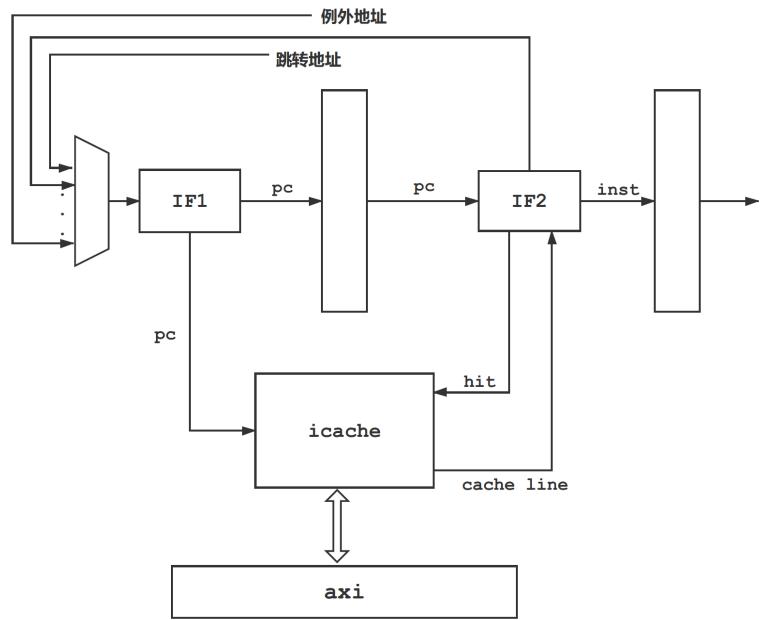


图 5: 取指阶段数据通路

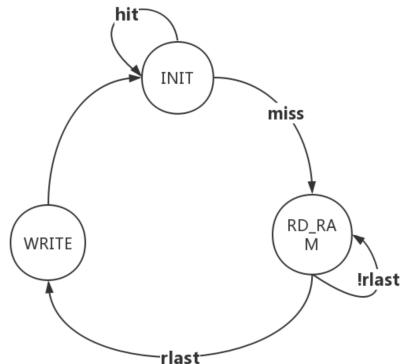


图 6: ICache 状态机

表 9: ICache 状态机各个状态描述

状态	含义
INIT	Cache 就绪，可连续读取
RDRAM	由于 Cache 缺失，由 AXI 总线发起突发传输
write	将突发传输的数据回填 Cache

4. 访存模块 dcache_confreg 和 mem 流水级设计

由于访问主存延迟过大，严重影响 CPU 的性能，需要数据 Cache。因为 cpu 使用统一编址而且 mips 地址空间中有 uncache 段，所以对不同地址空间的访问应该区别开来。我们设

计了一个 dcache_confreg 模块用于区别对 cache 和 uncache 的访问。使用 rwconreg 模块访问 uncache 段，每次只读写一个字，不经过 cache。

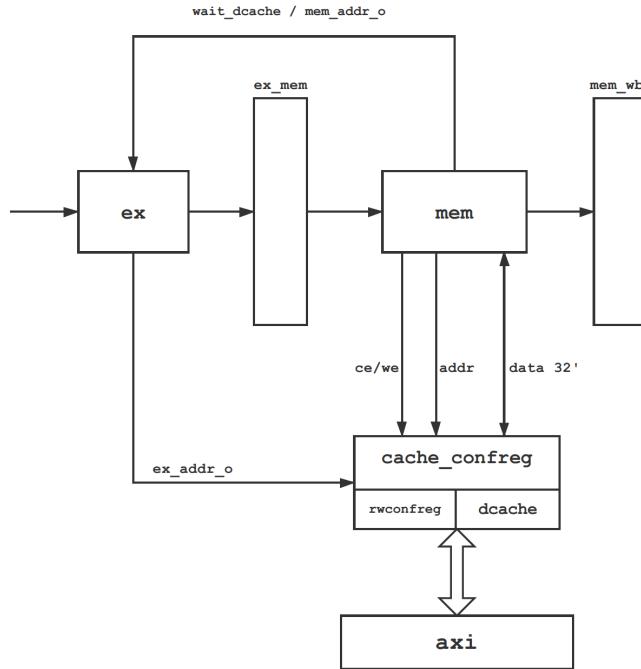


图 7：数据访存通路

(1) dcache

dcache 设计使用直接映射，块大小 4 个字，采用写回法和写分配法。在命中时，读操作不阻塞流水线，写操作需要两拍完成。由于 dcache 采用 Block RAM 实现，读操作需要耗费一拍，为避免阻塞流水线，故绕开 ex_mem 的流水线寄存器，在 ex 段直接将访存地址发送至 dcache 存储体，在下一拍时，cache line 和访存地址 mem_addr_o 同时到达 dcache 比较机构，产生 hit/miss 信号，并且接受 mem 级发出的读使能或写使能。

如果读命中则不阻塞流水线，读数据结果直接输出，如果写命中，因为要向 dcache 写入数据所以要保持地址不变，所以我们采取的方案是阻塞流水线一拍使写地址保持不变。

若读缺失，需要注意的是，此时，虽然此时暂停流水线信号已经生效，但是处在 EX 级的指令已经是下一条指令。为避免访存地址丢失，故总是将上一次访问 cache 的地址送回 EX 阶段并在 ex_mem 寄存器保存，并使用标志寄存器(cache_wait)对 cache 访问地址进行选择 (EX 段给出的地址 / ex_mem 寄存器保存的地址)。

若写缺失，由于在写阶段已经置高流水线暂停信号，故 EX 阶段仍保持原有指令，并不用处理访存地址丢失的问题。

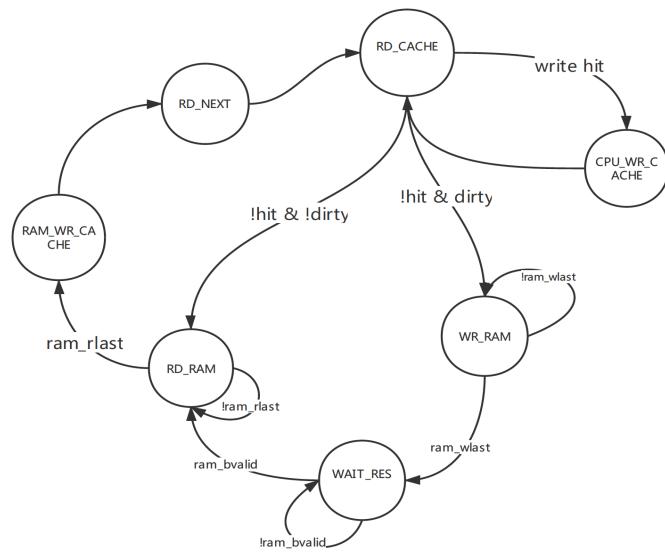


图 8: DCache 状态机

表 9 DCache 状态机各个状态描述

状态	含义
RD_CACHE	初始状态，从 bram 读出数据
CPU_WR_CACHE	把来自 CPU 的数据写入 cache 中
WR_RAM	把 cache 中数据写到 RAM 中
WAIT_RES	等待写响应
RD_RAM	从 ram 中读数据到缓存中
RAM_WR_CACHE	把读回的数据一次性写入到 cache 中
RD_NEXT	从 ram 读回数据并写入到 cache 中后需要一个周期从 cache 中读出数据

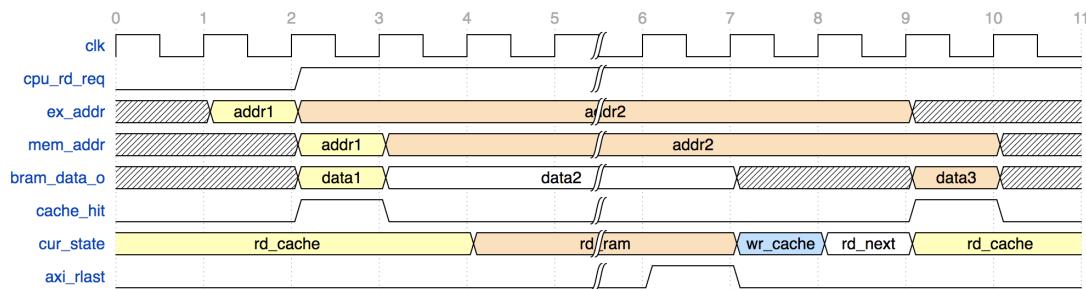


图 9: dcache 读时序

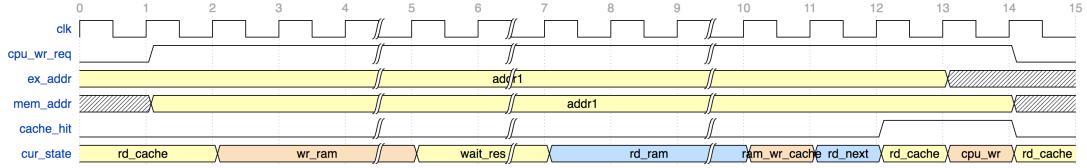


图 10: DCACHE 写时序 (写缺失)

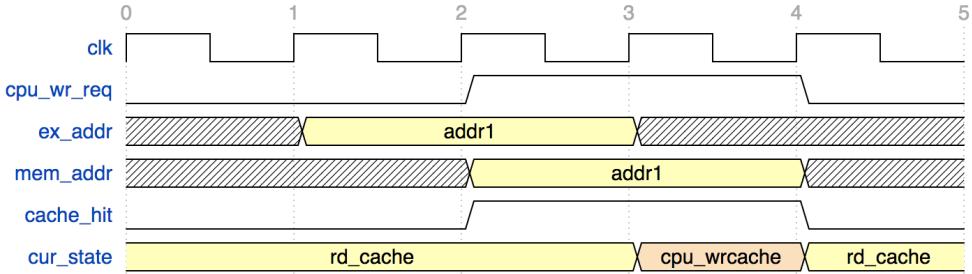


图 11: DCACHE 写时序 (写命中)

(2) mem 流水级

访存指令在 mem 级进行进一步的译码，产生读数据使能信号和具有字节选择功能的数据使能信号。mem 与 dcache_confreg 总是以字形式进行交互：读数据时读回来一个完整的字再进行切割拼接等，写数据时发出一个完整的字搭配字节选通的写使能信号。因为访存而需要暂停流水线时，由 dcache 和 rwconfreg 模块发出暂停信号，mem 模块只进行访存指令译码和收发数据。

5. CP0 模块和 exception 异常处理模块设计

本 CPU 能处理 6 种例外，支持软中断和硬中断。因为目前版本还没有 tlb，所以 cp0 的功能主要为保存 count 寄存器和异常相关寄存器（cause, status, badvaddr, epc）。

目前版本 CPU 处理异常的模式为精确异常，在各个流水级收集异常信息，在访存阶段的 exception 模块集中处理异常。在 exception 模块中，识别输入的异常流；产生 clear 信号清空流水线（清除异常指令后面的指令）；确定指令的下一个地址：异常返回时为 EPC 其他异常统一为 0xBFC00380；以及产生异常类型输出到 cp0 用于写 cp0 的寄存器。

在 WB 级写 cp0 的相关寄存器，在 EX 级读 cp0 寄存器的值，如果有数据相关则从对应的流水级前推数据（如果 WB 级写 cp0 则无论是否有相关都阻塞一拍，否则容易在这里出现关键路径而限制主频）。

6. 分支预测器设计

(1) 设计动机

在最初的 CPU 设计方案中，采用在 ID 阶段判断分支并计算跳转地址的方案。而在实际

测试过程中，ID 阶段判断分支一定程度上加重了 ID 级组合逻辑的复杂性，限制主频，故将分支判断转移至 EX 阶段，并引入分支预测机制。

(2) 设计方案

分支预测器设计采用 2 位饱和计数器的设计方案。其中，分支地址记录表（BTB）地址映射方式为直接映射，使用 Block RAM 实现。经对 CPU 结构的分析和 MIPS 延迟槽指令的分析，分支预测应发生在 IF2 级，在 ID 阶段，ID 阶段给出译码结果，可判断当前指令是否为分支指令，结合译码结果，可确定是否使能分支预测器的输出。若是分支指令，则采用分支预测器的输出。若预测器预测分支将发生，则将预测的分支地址送入 IF1 级进行取指。结合 MIPS 体系结构中延迟槽指令的特性，可以最大程度地降低分支预测失败导致清除流水线的损失。

(3) 分支预测的工作流程

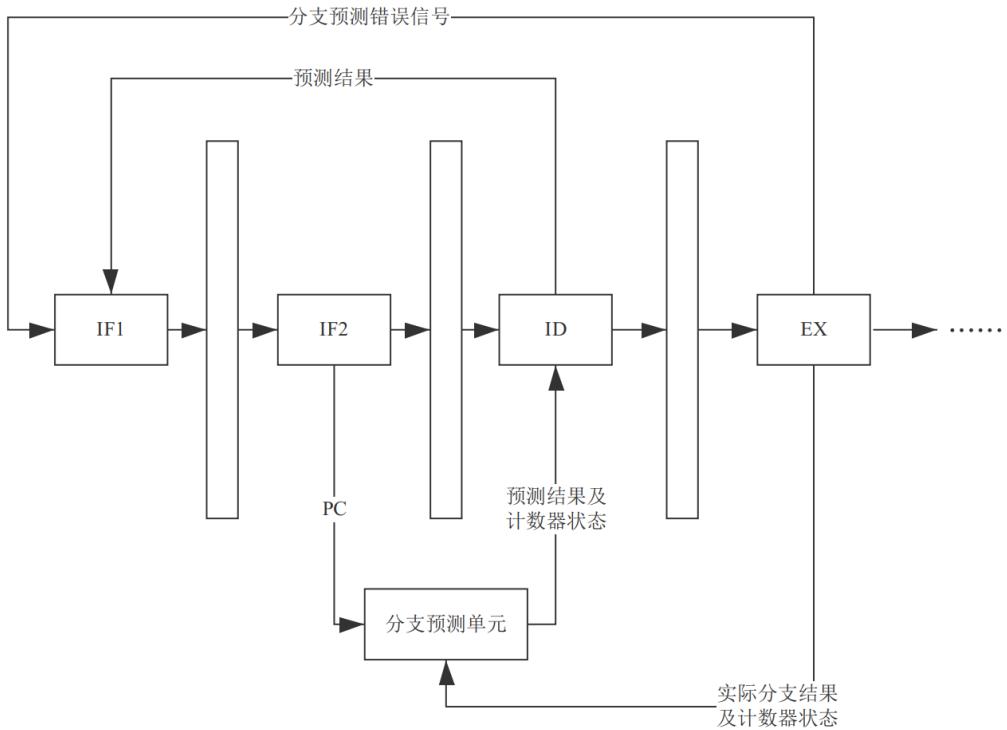


图 12：分支预测数据通路

IF2 单元将 PC 地址传送给分支预测单元，在下一个时钟上升沿，分支预测器给出 PC 地址相应分支历史表和饱和计数器的值。

指令、PC 地址和饱和计数器、分支历史记录的值同时进入 ID 阶段。需要注意的是，由于采用的是直接映射，不同的 PC 可能映射到同一存储单元，故传入的 PC 所指向的指令不

一定是跳转指令，需要结合 ID 阶段的译码结果，确认该指令为分支指令后，才可输出分支预测器的预测结果，否则忽略分支预测器的输入，图中无色部分即表示无意义的输出，输出结果被忽略。

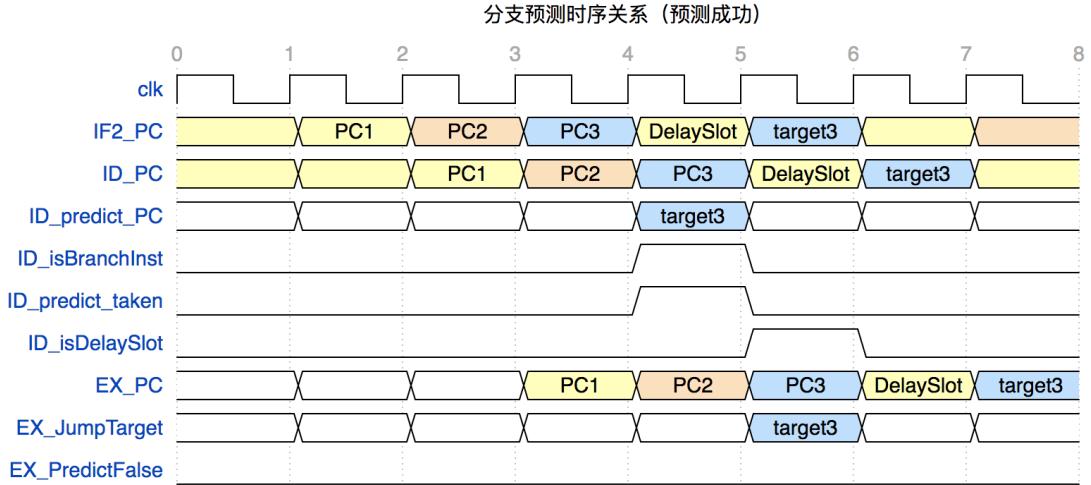


图 13：分支预测各个信号的时序关系（预测正确情况）

例如，在上面的时序图中，假设 Cache 均为命中情况。以 PC3 指向的指令为例，在上升沿 4 时，该指令进入 ID 级，同时分支预测器也读出了相应的预测结果送入 ID 级。结合其对应计数器状态和指令译码结果，确定其为分支指令(ID_isBranchInst 为 1)，指示其分支将发生(predict_taken 为 1)，并将读出的预测目标地址送入 IF1 级。与此同时，由于在上升沿 4 时，分支预测的结果并未到达 IF1，即有 1 个时钟周期的延迟，故仍然取出了(PC3)+4 处的指令。而由于 MIPS 的分支延迟槽设计，该指令为延迟槽指令，故无需清除该条指令。

当分支指令进入 EX 级时，由 EX 计算分支是否真正发生，如果发生，计算目标地址，然后结合从 ID 阶段传来的分支预测信息，判断分支预测是否正确。因此，需要将分支预测的信息（计数器、目标地址）通过流水线寄存器传递到 EX 级进行比较。如果预测正确，则无需清除流水线，如果错误，则需要清除错误的指令（下图的第 5 周期，发出了清除信号，清除了 IF2/ID 寄存器中取出的错误指令）。同时，根据分支的实际情况，更新分支历史信息表和饱和计数器。

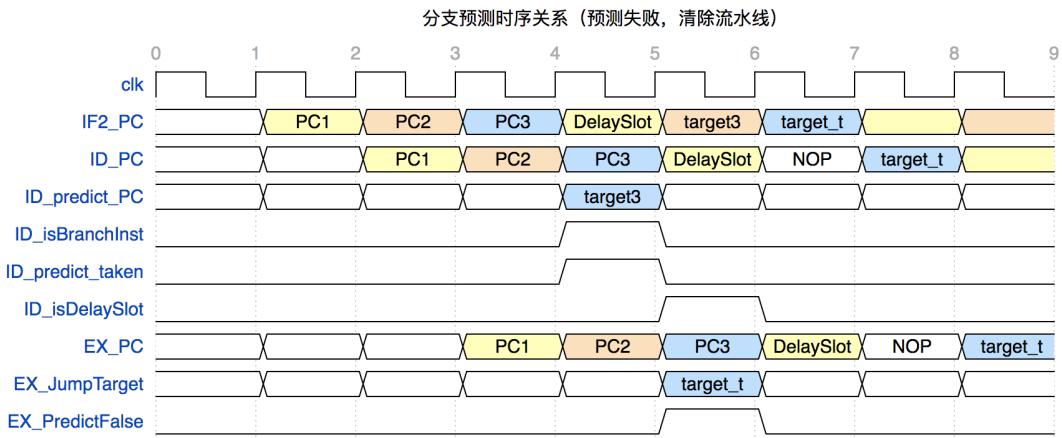


图 14: 分支预测各个信号时序关系 (预测错误情况)

(4) 分支预测的具体实现方案

表 10: 分支预测器输入信号说明

信号名称	信号来源	信号含义
pc_i	IF2	目前分支指令所在 PC 的值
id_isBranchInst_i	ID	ID 级是否为分支指令 (决定是否输出预测信息)
ex_isBranchInst_i	EX	EX 级是否是跳转指令 (决定是否更新 BTB)
ex_branch_inst_pc_i	EX	EX 级的指令 PC (用于更新 BTB)
ex_real_branch_target_i	EX	EX 级实际的跳转地址
idx_predict_branch_target_i	ID/EX 寄存器	IF 级产生的跳转预测地址
idx_predict_counter_i	ID/EX 寄存器	IF 级读出的跳转状态计数器
ex_actual_taken_i	EX	此跳转指令是否真正跳转

表 11: 分支预测器输出信号说明

信号名称	信号去向	信号含义
pre_taken_o	IF	预测结果是否跳转
pre_target_o	IF	跳转指令后的地址
pre_counter_o	IF	对应跳转指令分支状态的计数器

使用位宽为 34, 深度为 1024 的 Block RAM 单元, 其中 33-34 位为记录分支历史跳转方

向的 2 位饱和计数器，余下 32 位为分支跳转的历史记录表。

由于 PC 的地址为 32 位，而 Block RAM 的大小有限，需要对 PC 进行映射，映射方式为直接映射。为减少地址冲突，由于 PC 低 2 位均为 0，采用 PC 的 12-2 位作为分支预测器的入口地址。

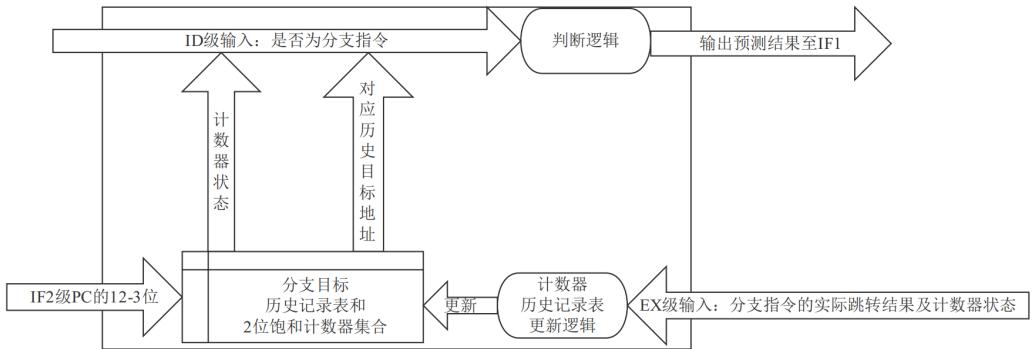


图 15：分支预测框图

7. AXI 总线接口设计

cpu 的接口设计采用 AXI 总线协议[3]。AXI 接口模块与 icache 和 dcache_confreg 模块连接。访问地址为 uncache 段时一次突发一个字，否则一次总是突发四个字，在各种随机延迟下均可正常工作。

(1) 仲裁

总体采用数据优先指令的仲裁策略。

对于数据，写数据优先于读数据，但就目前设计而言，读写数据不会同时发生。

在写数据时允许读指令发出请求，也就是写数据时，读地址通道和读通道可被读指令占用。

在一条写请求收到写响应之前不会撤下请求和发出下一个写请求；在读数据收到 last 信号之前不会撤下请求和发出下一个读数据请求；在读指令收到 last 之前不会发出下一个读指令请求和撤下请求，但允许发出下一个读数据请求或下一个写数据请求。

读数据 id=4'b0001, 读指令 id=4b0000, 只有这两种 id, 写操作忽略 id。

(2) 状态机

axi 接口模块中有两个状态机，一个用于读写数据，一个用于读指令。

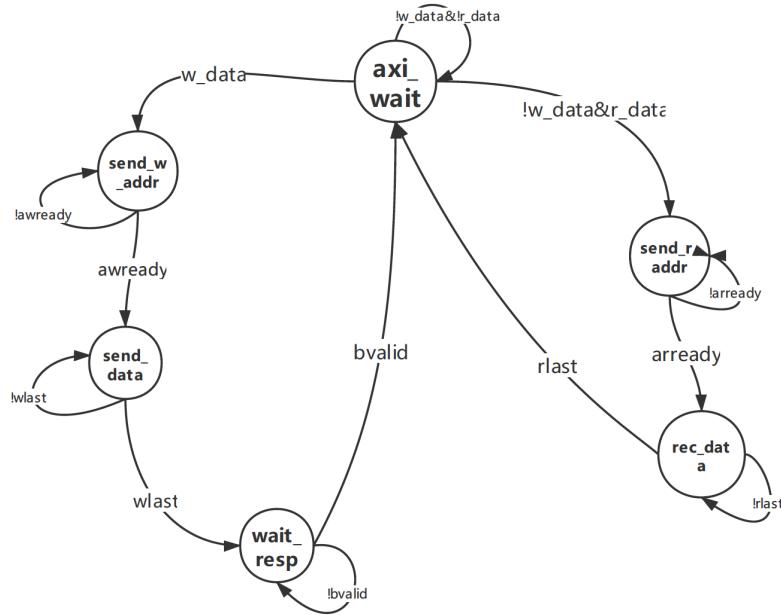


图 16: 读写数据状态机

表 12: AXI 接口读写数据状态机各个状态描述

状态	含义
axi_wait	无请求时的初始状态
send_r_addr	发送读数据地址
rec_data	接收读数据
wait_resp	等待写响应
send_data	写数据时发送数据
send_w_addr	发送写数据地址

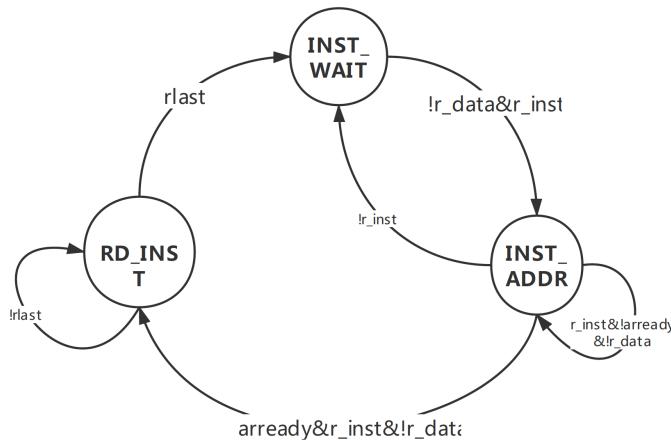


图 17: 读指令状态机

表 13: AXI 接口读写数据状态机各个状态描述

状态	含义
INST_WAIT	没有读指令请求或者有读数据请求
INST_ADDR	读指令地址
RD_INST	突发读取 4 个字

三、设计结果

(一) 设计交付物说明

1. 目录层次

-score.xls	功能、性能测试得分
-design.pdf	设计报告
-soc_axi_func/	目录, 自实现CPU的axi接口功能测试环境
--rtl/	目录, soc_lite源码
--soc_lite_top.v	soc_lite顶层
--myCPU/	目录, 自实现CPU源码
--blk_mem_gen_0/	目录, block ram ip核0, 用于cache
--blk_mem_gen_1/	目录, block ram ip核1, 用于cache
--blk_mem_gen_2/	目录, block ram ip核2, 用于cache
--bpu_mem/	目录, block ram ip核3, 用于分支预测
--div_gen_0/	目录, div ip核, 用于除法模块
--mul_gen_0/	目录, mul ip核, 用于乘法模块
--CONFREG/	目录, confreg模块
--axi_wrap/	目录, cpu axi接口包装一层
--ram_wrap/	目录, axi ram的封装层, 增加随机延迟设置
--xilinx_ip/	目录, Xilinx IP, 只有*.xci文件
--testbench/	目录, 仿真文件
--mycpu_tb.v	仿真顶层, 该模块会抓取debug信息与trace_ref.txt进行比对
--run_vivado/	目录, 运行Vivado工程
--soc_lite.xdc	Vivado工程设计的约束文件
--mycpu_prj1/	目录, Vivado2018.3创建的Vivado工程
--mycpu.xpr	Vivado工程, 可直接打开并进行仿真、综合实现
--func.bit	bit文件, 用于运行89个功能点测试
--memory.bit	bit文件, 用于运行记忆游戏

```

|-soc_axi_perf/          目录，自实现CPU的性能测试环
|  |--rtl/                目录，soc_lite源码
|  |  |--soc_lite_top.v   soc_lite顶层
|  |  |--myCPU/            目录，自实现CPU源码
|  |  |  |--blk_mem_gen_0/  目录，block ram ip核0，用于cache
|  |  |  |--blk_mem_gen_1/  目录，block ram ip核1，用于cache
|  |  |  |--blk_mem_gen_2/  目录，block ram ip核2，用于cache
|  |  |  |--bpu_mem/        目录，block ram ip核3，用于分支预测
|  |  |  |--div_gen_0/      目录，div ip核，用于除法模块
|  |  |  |--mul_gen_0/      目录，mul ip核，用于乘法模块
|  |  |--CONFREG/           目录，confreg模块
|  |  |--axi_wrap/           目录，cpu axi接口包装一层
|  |  |--ram_wrap/           目录，axi ram的封装层，增加随机延迟设置
|  |  |--xilinx_ip/         目录，Xilinx IP，只有*.xcii文件
|  |--testbench/            目录，仿真文件
|  |  |--mycpu_tb.v         仿真顶层，该模块会抓取debug信息与trace_ref.txt进行比对
|  |--run_vivado/           目录，运行Vivado工程
|  |  |--soc_lite.xdc       Vivado工程设计的约束文件
|  |  |--mycpu_prj1/         目录，Vivado2018.3创建的Vivado工程
|  |  |--run_allbench.tcl    仿真依次运行10个性能测试程序的脚本
|  |  |  |--mycpu.xpr        Vivado2018.3创建的Vivado工程
|  |  |  |--perf.bit          bit文件，用于运行性能测试

|-soft/                   目录，功能测试和性能测试软件程序目录
|  |--func/                目录，89个功能点测试程序
|  |--memory_game/          目录，记忆游戏测试程序
|  |  |--memory_game/        目录，记忆游戏测试程序

```

2. 仿真

功能仿真在仿真之前请确认 axi_ram 中加载的初始文件是否是功能测试程序或者是否在仿真文件中加入.mif 文件。运行记忆游戏时可能会报错不能综合，一般原因是功能测试的 inst-ram.coe 没有找到。

如果有关于 div ip 核的报错，请关闭“使用预编译的 IP 仿真库”以及关闭“增量编译”。

3. 上板演示

在性能测试的分数计算中，我们采用的是多次测试（同一个测试程序多次 reset），选择几次中耗时最短的一次进行计算的。

(二) 设计演示结果

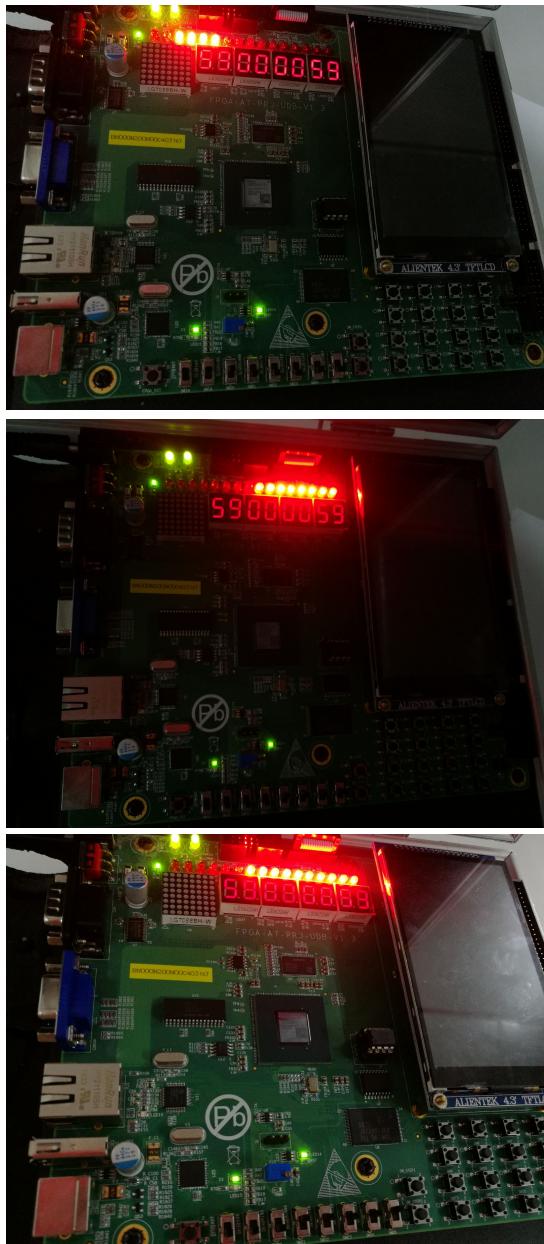
1. 功能测试综合时序报告 (109MHz)

General Information	Name	Waveform	Period (ns)	Frequency (MHz)
Timer Settings	clk	{0.000 5.000}	10.000	100.000
Design Timing Summary	clkfbout_clk_pll	{0.000 5.000}	10.000	100.000
Clock Summary (4)	cpu_clk_clk_pll	{0.000 4.583}	9.167	109.091
> Check Timing (577)	sys_clk_clk_pll	{0.000 5.000}	10.000	100.000
Intra-Clock Paths				
clk				
clkfbout_clk_pll				
cpu_clk_clk_pll				
Setup 0.102 ns (10)				
Hold 0.073 ns (10)				
Pulse Width 3.453 ns (31)				
sys_clk_clk_pll				
Setup 0.100 ns (10)				
Hold 0.077 ns (10)				
Pulse Width 3.870 ns (31)				

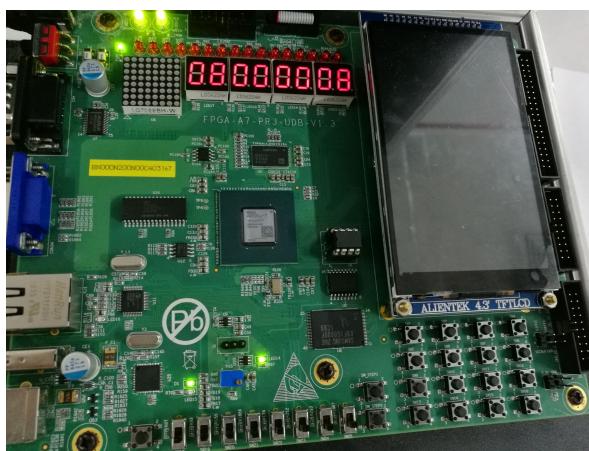
2. 性能测试综合时序报告 (109MHz)

General Information	Name	Waveform	Period (ns)	Frequency (MHz)
Timer Settings	clk	{0.000 5.000}	10.000	100.000
Design Timing Summary	clkfbout_clk_pll	{0.000 5.000}	10.000	100.000
Clock Summary (4)	cpu_clk_clk_pll	{0.000 4.583}	9.167	109.091
> Check Timing (577)	sys_clk_clk_pll	{0.000 5.000}	10.000	100.000
Intra-Clock Paths				
clk				
clkfbout_clk_pll				
cpu_clk_clk_pll				
Setup 0.152 ns (10)				
Hold 0.041 ns (10)				
Pulse Width 3.453 ns (31)				
sys_clk_clk_pll				
Setup 0.074 ns (10)				
Hold 0.077 ns (10)				
Pulse Width 3.870 ns (31)				

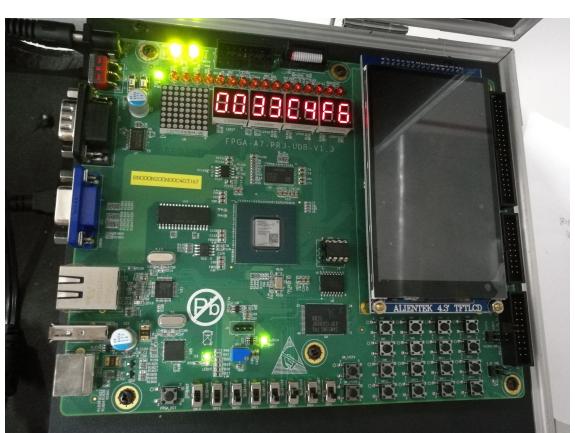
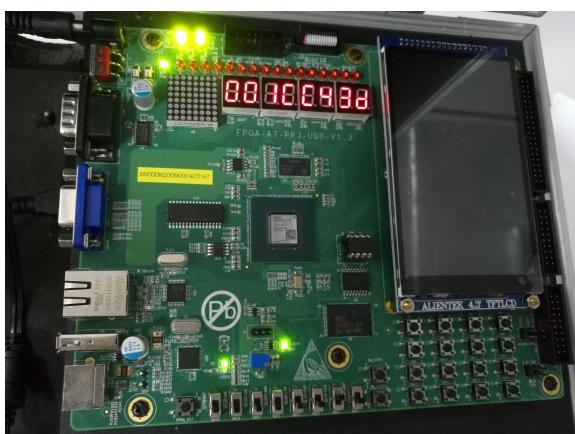
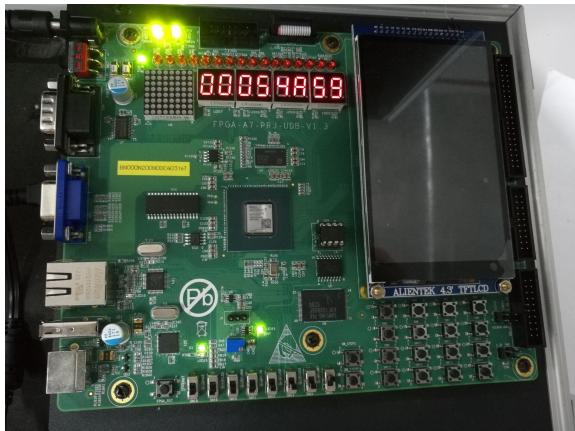
3. 功能测试 (3 种随机种子)

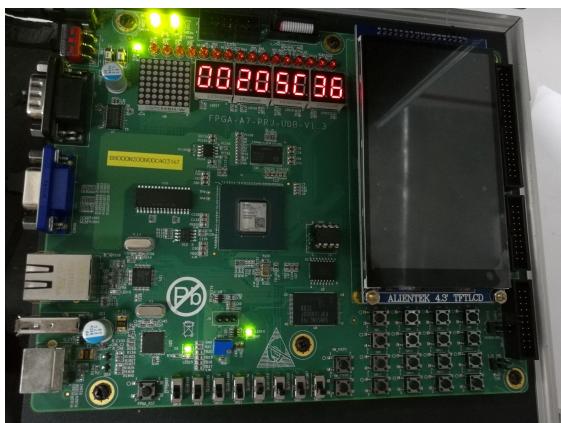
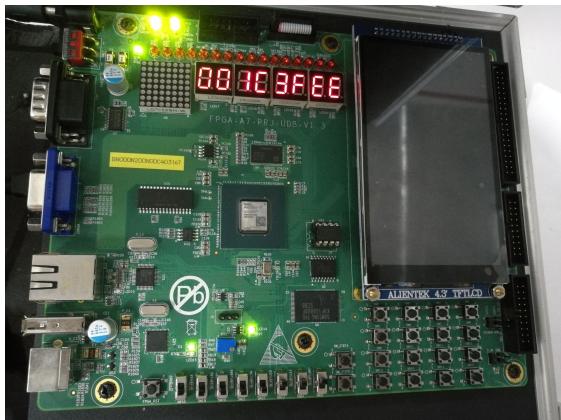
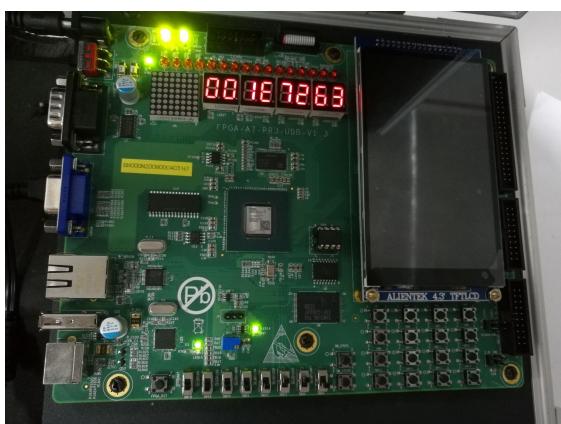
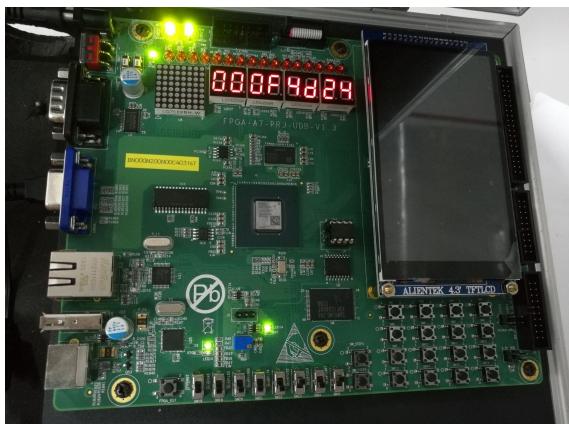


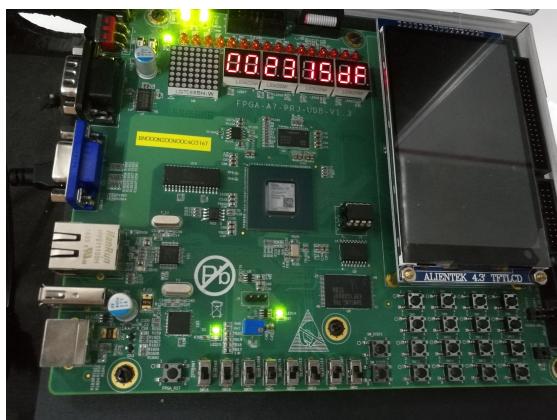
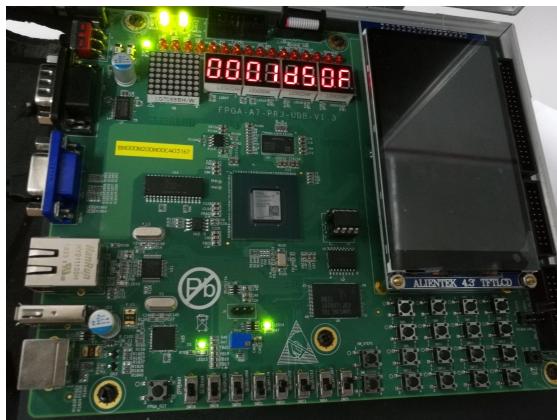
4. 记忆游戏



5. 性能测试







6. 采用的优化策略和带来的性能提升

表 14: 采用的优化策略以及带来的性能提升

优化策略	优化后性能得分
增加 ICACHE	7
增加 DCACHE 及改进 AXI 接口	23
优化关键路径, 提升频率	29
优化流水线结构, 将 IF 拆分为 IF1 和 IF2, 提升单位时间内连续取指数	39
简化 ID 级结构, 将分支指令判断移至 EX 级, 提高整体频率	43
改变 Dcache 访问时机, EX 段提前访问 DCache	48
增加动态分支预测	50
再次优化关键路径, 提升频率	54

四、参考设计说明

id 模块,ex 模块和基础数据通路参考《自己动手写 CPU》。

动态分支预测的实现思想参考《计算机组成与设计：软件/硬件接口》

使用乘法 IP 核，除法 IP 核，blockram IP 核。

五、参考文献

- [1] David A. Patterson, John L. Hennessy. 计算机组成与设计：硬件/软件接口[M]. 北京:机械工业出版社,2015.
- [2] 雷思磊.自己动手写 CPU[M]. 北京:电子工业出版社,2014.
- [3] ARM. AMBA AXI Protocolv1.0[P].