双发射处理器设计报告

中山大学一队
杨旭明 杨升荣 余杨斌 邱仲禹

一、设计简介

本设计依托于“第二届系统能力培养大赛”提供的参考资料及测试资源，设计了一个基于 MIPS32 指令集的顺序双发射处理器。包括四个 FU 处理单元，其中两个是 ALU 单元，一个乘法单元以及一个除法单元。并且加入了 Cache 结构与 Store Buffer 以提高指令与数据的存取效率。

二、设计方案

（一）总体设计思路

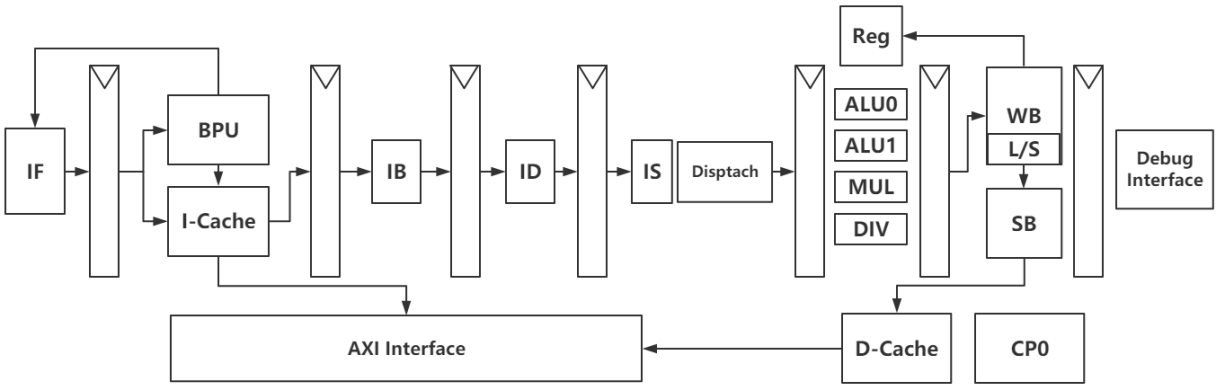

Fig.1 MyCPU 的整体框架

如图，MyCPU 设置了七个流水级，取指阶段 IF 经过握手同时向指令缓存 I-Cache 和分支预测单元 BPU 发送 PC，I-Cache 一次取出 4 条指令存入指令缓存 IB 中。指令缓存每周期向译码阶段 ID 发送两条指令，解码出指令类型、源寄存器编号、目的寄存器编号、立即数等信息后将解码信息随 PC 一起发往发射阶段 Issue queue。Dispatch 模块判断指令间的相关性后顺序分发至 EX 阶段的不同 FU 进行执行，同时指令队列会将已经解码但未来得及发射的指令保存下来，等待发射。EX 阶段设置了 4 个 FU：两个 ALU 处理大部分指令、一个 MUL 处理乘法指令、一个 DIV 处理除法指令。同时在 EX 阶段进行分支指令跳转结果的判断，若跳转错误触发错误恢复机制。EX 阶段经过握手同时向 WB 和 CP0 传递 PC 等信号，CP0 处理例外，WB 则负责处理需要写回的指令与访存指令。

（二）分支预测模块设计

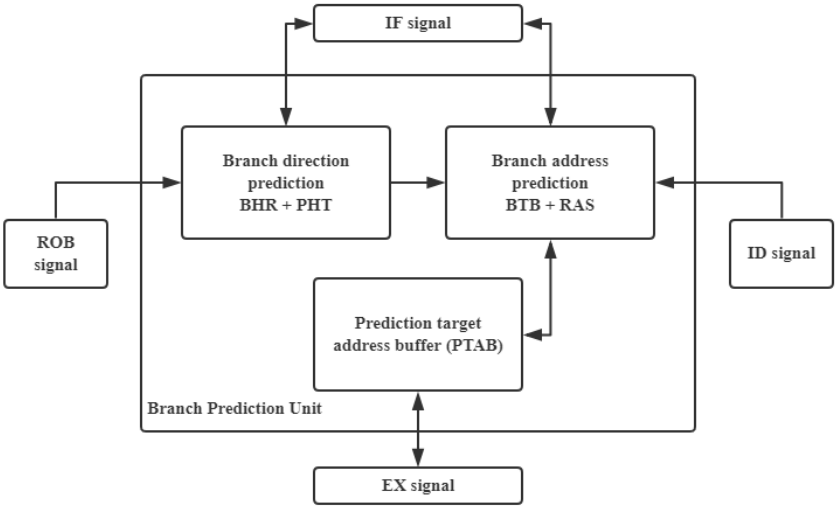

Fig.2 分支预测模块框图

分支预测模块主要分为三个部分：分支方向预测模块、分支地址预测模块、目标地址缓存（PTAB）[1]。分支方向预测模块采用“G-Share”策略，包含了一个全局历史寄存器（BHR）以及分支模式历史表（PHT），特殊的哈希映射函数降低了重名效应的影响。分支地址预测模块采用分支目标缓冲器（BTB）和返回地址堆栈（RAS）协作的策略，其中 BTB 部分包含了四个 BRAM 对目标地址进行存放。目标地址缓存（PTAB）的作用是存放预测结果，等到 EX 阶段计算出该指令的实际跳转情况后，将实际与预测进行比较从而决定是否需要触发错误恢复机制。

（三）Cache 模块设计

Cache 模块主要由 I-cache 和 D-cache 组成，还包括 cache_top 和 d_cache_arbitrator 部分。整个模块可以处理不同流水级同时访问 D_cache 或者同时访问 axi 总线的冲突问题。I-Cache 采用的是 4 路组相连结构，深度为 256，每个 cache line 容量为 16 words，I-Cache 总容量为 64KB。D-cache 采用直接映射，深度为 256，每个 cache line 容量为 16 words，D-Cache 总容量为 16KB。

（四）Issue queue 模块设计

Issue queue 模块的主要作用是将已经解码出来但是不能送到 FU 执行的指令保存下来，最多可以容纳 32 条指令。每条指令包括的信息用 pc, dst, src0, src1, imme, meaning, busy,

data_valid, ptab_addr, exe_code, delot_flag 表示。如果一条指令有目的寄存器，会将目的寄存的信息存入到 dst 当中，同样的，如果一条指令存在目的寄存器和立即数，会将相应的信息存入到 src0, src1, imme 当中。并且通过 data_valid 选择这些信息是否是有效的。Meaning 表示处理器内部对指令的划分，busy 表示指令队列中对应的项是否被占用。Issue queue 模块本质上是一个 FIFO，按照先入先出的原则，受 Dispatch 模块的控制将指令送出。

（五）Dispatch 模块设计

Dispatch 模块根据 EX 阶段的状态和 Issue queue 模块送来的指令，判断是否发射指令。Dsipatch 模块分两路，分别对两路指令进行判断。由于本处理器是双发射顺序执行的处理器，因此指令之间只存在写后读相关性，Dispatch 模块对写后读相关性进行检测，判断能否同时向 EX 阶段发射两条指令。如果两条指令存在写后读相关性，就只发射一条指令。同时，为了简化 cache 的设计，将处理器的可执行的访存指令限制在一条，这样可以简化 cache 的端口设计。因此，在 dispatch 模块还要对访存指令进行检查，避免同时向 EX 阶段同时发射两条访存指令。为了简化例外处理，也将分支跳转指令和延迟槽指令放在同一个周期发射。

（六）EX 模块设计

EX 模块由 ex_stage, FU_selecotr, ALU0, ALU1, MUL 和 DIV 组成，根据 Dispatch 模块传过来的 2 条指令的 op 利用 FU_selector 来选择该条指令由哪一个 FU 执行，默认情况下 ALU0 执行 way0 的指令，ALU1 执行 way1 的指令，当遇到乘除法时该路的指令自动跳转到 MUL/DIV 模块。并利用 bypass 决定源寄存器数据是通过利用上一拍的计算结果还是去读寄存器。当数据处理完后通过握手传给 WB 模块。

（七）WB 模块设计

WB 模块主要负责访存指令的执行和寄存器数据的写回。WB 模块将 EX 传来的地址和数据进行整合，向 cache 和 store buffer 发出 load 的指令和访存指令。其中，uncache 类型的指令直接访问内存。Cache 指令会经过 store buffer 和 cache。

（八）Store buffer 模块设计

Store buffer 主要是为了提高 cache 的利用效率。当 cache 因为 load 指令繁忙的时候，可以将 store 指令的数据和地址暂时存入到 store buffer，等到 cache 空闲的时候再将 store 指令的数据存入 cache。这样可以使得 cache 优先处理 load 指令，减少 cpu 的等待时间。Store

buffer 有 16 项，最多可以容纳 16 条 store 指令的地址和数据。值得注意的是，当 cacheable 类型的 load 指令需要数据的时候，会优先从 store buffer 中找数据，如果没有找到，再到 cache 中寻找，如果还是没有找到，才会到内存中寻找数据。这也一定程度上减轻了 cache 的负担。

三、设计结果

功能测试仿真上板通过完成，可通过记忆游戏测试。本设计使用 50MHz 的 CPU 频率进行性能测试，可仿真上板通过 bitcount、crc32、sha、stream_copy 等测试。

（一）设计交付物说明

Design：本设计的设计报告。

Soc_axi_perf：性能测试的测试环境。

Soc_axi_func：功能测试的测试环境。

Soft：功能测试与性能测试的软件程序目录。

（二）设计演示结果

1. 功能测试

```
——[7324515 ns] Number 8'd88 Functional Test Point PASS!!!
——[7367455 ns] Number 8'd89 Functional Test Point PASS!!!
================================================================
Test end!
——PASS!!!
$finish called at time : 7372531500 ps : File "C:/Users/50491/
```

Fig.3 功能测试仿真通过截图

Fig.4 功能测试板级验证通过照片

2. 性能测试

```
================================================================
Test begin!
bitcount test begin.

Bit counter algorithm benchmark



bitcount PASS!Bits: 811

bitcount: Total Count = 0x1594b

================================================================
Test end!
———PASS!!!
$finish called at time : 1033385500 ps : File "C:/Users/aigo/Desktop/
run: Time (s): cpu = 00:00:44 ; elapsed = 00:01:21 . Memory (MB): pea
```

Fig. 5 Bitcount 仿真通过截图

```
1601645211, 00000200

crc32 PASS!

crc32: Total Count = 0x153c3a

================================================================
Test end!
———PASS!!!
$finish called at time : 14030435500 ps : File "C:/Users/aigo/Desl
run: Time (s): cpu = 00:10:28 ; elapsed = 00:19:13 . Memory (MB):
```

Fig. 6 crc32 仿真通过截图

```
================================================================
Test begin!
stream copy test begin.

stream copy PASS!

stream copy: Count = 0xea04

================================================================
Test end!
———PASS!!!
$finish called at time : 722976500 ps : File "G:/ERIC/201file/nscscc
run: Time (s): cpu = 00:01:02 ; elapsed = 00:01:27 . Memory (MB): pe
```

Fig. 7 stream_copy 仿真通过截图

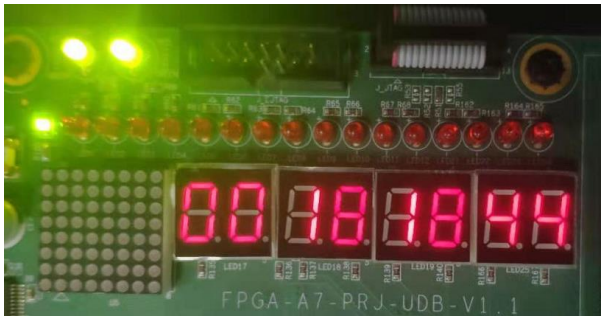

Fig.8　bitcount 板上通过截图

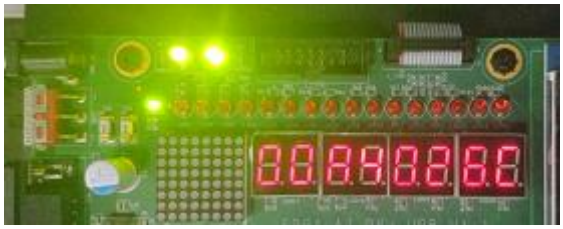

Fig.9　bubble_sort 板上通过截图

Fig.10　coremark 板上通过截图

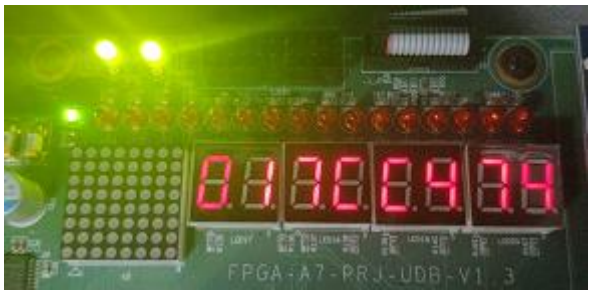

Fig.11　crc32 板上通过截图

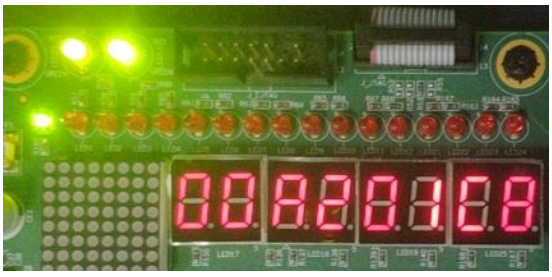

Fig.12　quick_sort 板上通过截图

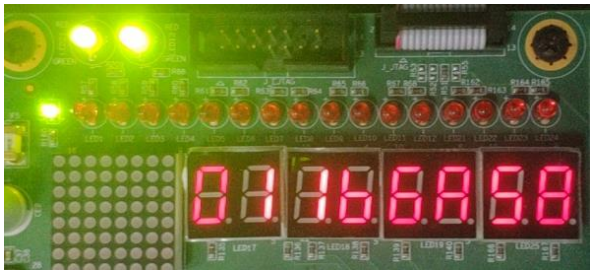

Fig.13　select_sort 板上通过截图

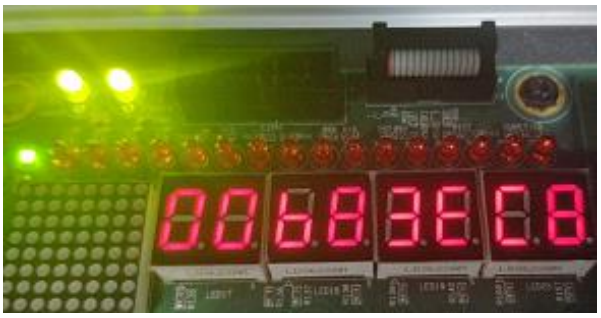

Fig.14　sha 板上通过截图

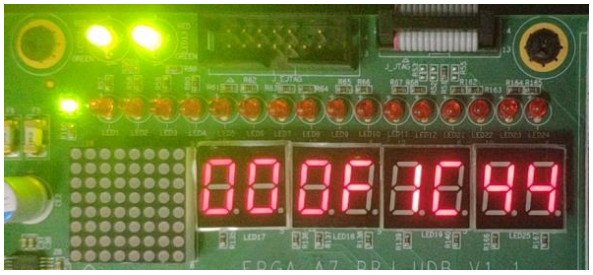

Fig.14　stream_copy 板上通过截图

Fig.15　stringsearch 板上通过截图

四、参考设计说明

本设计中宏定义、AXI_Interface、decoder、Icache 等模块借鉴了 2018 年龙芯杯全国大学生系统能力大赛中山大学一队的源码，进行端口设置和模块划分。

五、参考文献

[1]姚永斌. 超标量处理器设计[M]. 北京：清华大学出版社, 2014.