

MIPS CPU 设计报告

北京理工大学 01 队
黄兴泽、罗益善、王振源

一、设计简介

本项目目的是实现一个 32 位 MIPS CPU。实验开发平台和环境为 vivado2019 以及龙芯赛务组提供的实验平台，(FPGA 为 Artix-7 XC7A200T)。实现的指令集为精简的 MIPS32 指令集，以及系统测试所需的例外与中断控制器。功能方面能够通过大赛方提供的功能测试、性能测试、系统测试；能够启动 PMON 引导程序，并可以通过 PMON 从网口烧录 ucore 操作系统并运行。性能方面频率达 97MHz，

二、设计方案

(一) 总体设计思路

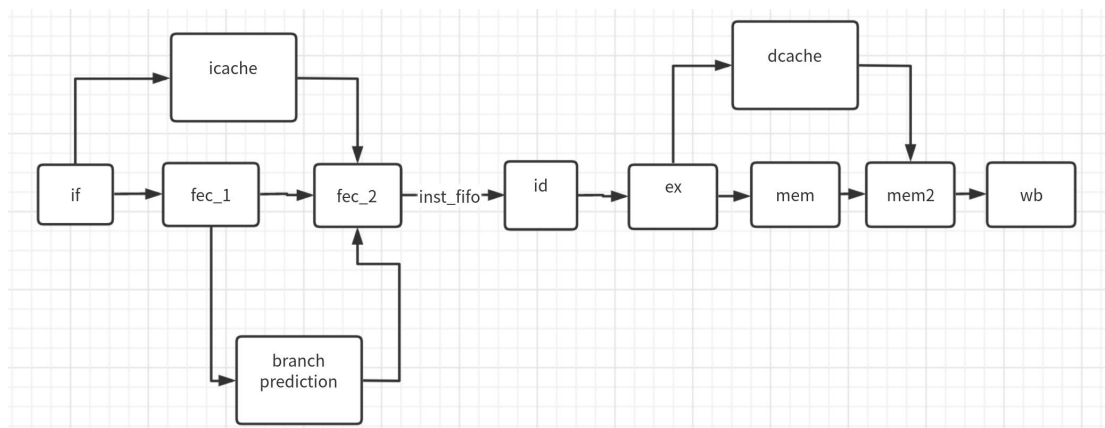
本 CPU 采用顺序单发射的八级流水线结构。实现了 MIPS32 Rev 1 指令集的全部指令以及例外中断部分，为了保证性能测试的得分，在性能测试的工程下载减了不必要的指令和例外机构。并且配置的一级 8KBCache，Cache 级内部分为 I-Cache 和 D-Cache，采用二路组相连，进行读写。同时配备分支预测器和指令缓冲队列以保证较长流水级的工作性能。

(二) Cache 模块设计

Cache 模块分为指令 cache 和数据 cache 两部分，均采用流水线设计。采用 lru 算法实现块置换，二路组相连设计。

通过转化的地址映射到 Tags 以确定是否命中。当 Cache 命中时，可以连续不断的进行数据数据交换而无需与内存进行数据交互，极大的提高运行效率。

(三) 流水和前递模块设计



为了能够进一步提升系统的并行性同时降低单个流水级的复杂度以提升频率。系统采用了七级流水线，同时考虑到寄存器间的冲突，流水级间可以通过前递避免退后处理和流水线断流，为了在避免流水线断流和频率之间达到较好的平衡，我们取消了有关于 cache 的所

有前递，尽量减少前递对于频率的影响

相比于传统的五级流水线，将访存和取指级进行细分流水。在取指部分和后续流水线中间增加指令缓冲区，尽量降低前后耦合。

(四) 分支预测模块设计

长流水线的设计会导致出现分支回退时的开销增大，为了降低流水断流和回退的开销，设计了分支预测器以提升处理的效率。

分支预测器采用了基于局部历史的分支预测技术，其中 pht 和 btb 采用 blockram 实现，最后效果良好，极大的降低了因分支指令回退的消耗

(四) 虚实地址内存管理

采用分段地址映射进行内存管理，对 kseg0 kseg1 进行直接映射，对 kseg kseg2 kseg3 进行 TLB 映射。对于未命中的地址，通过 TLB 例外交由操作系统进行处理。

(五) 中断与异常模块设计

CPU 实现了 MIPS 规范的一部分异常，支持六条外部中断和功能测试中要求的所有例外。

(六) 外设接口设计

CPU 实现了以下外设接口：

SPI 启动只读引导程序

通过 SPI 控制器进行 SPI 二次烧写

GPIO 控制的数码管 拨码开关 LED

MAC 网络控制器

DDR3 内存颗粒

NAND 存储器

RS232 串口控制器

(七) 协处理部分

为了启动操作系统，实现了 MIPS32R1 提出规范的协处理器，但是在实际的调试中，为了实现 ucore 从内存中启动，需要实现 CP0 \$15 号寄存器以重新指定中断入口地址。所有实现的 CP0 寄存器名称与编号如下：

Index Register \$0

Random Register \$1

Elo Register \$2

Context Register \$4

PageMask Register \$5

Wired Register \$6

BadVA Register \$8

Count Register \$9

Ehi Register \$10

Compare Register \$11

Status Register \$12

Cause Register \$13

Exception Program Register \$14

Precessor ID Register \$15

EBase Register \$15

Congration Register \$16

Congration Register1 \$16

三、设计结果

(一) 设计交付物说明

```

└── BIT_1_huangxinze
    ├── bit
    │   ├── func_test
    │   │   ├── axi_mem_game_test
    │   │   ├── soc_axi_func
    │   │   │   └── soc_axi_lite_top.bit
    │   │   └── soc_sram_func
    │   └── perf_test
    │       └── soc_axi_lite_top.bit
    └── system_test
        └── soc_up_top.bit
    ├── score.xls
    ├── sram_src
    │   └── mycpu
    └── src
        ├── mycpu
        │   └── axi_crossbar_0.xci
    
```

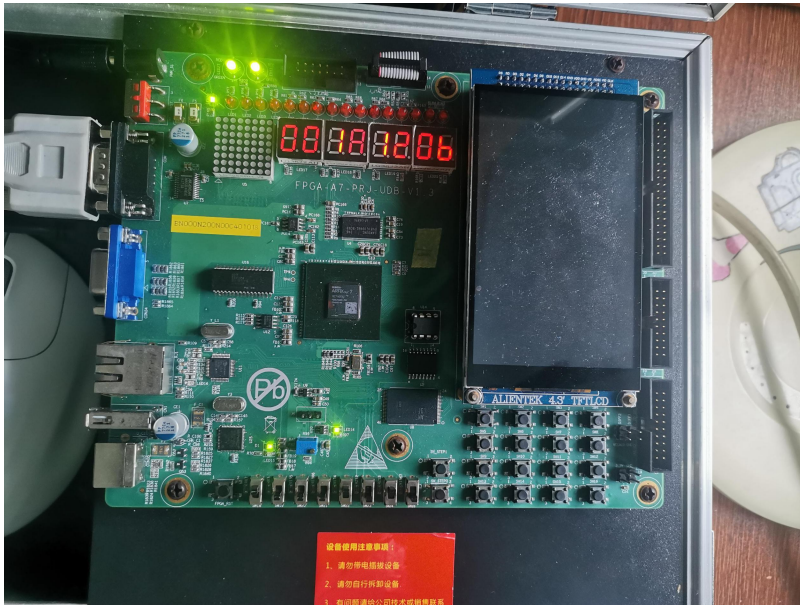
- | |—— bht_data_ram.xci
- | |—— btb_data_ram.xci
- | |—— btb_tag_ram.xci
- | |—— dcache_data_ram.xci
- | |—— icache_data_ram.xci
- | |—— mycpu_top.v
- | |—— pht_data_ram.xci
- | |—— signed_div.xci
- | |—— signed_mul.xci
- | |—— unsigned_div.xci
- | |—— unsigned_mul.xci
- | └—— perf_clk_pll.xci

(二) 设计演示结果

功能测试演示：



性能测试演示：



系统测试演示:

```

管理员: C:\WINDOWS\system32\cmd.exe - python ./term.py -s com4 -b 57600

F:\nscsc2022_group_v0.01\nscsc2022_group\system_test_v0.01\windows_env\supervisor-mips32-master\supervisor-mips32-master\term.py
python ./term.py -s com4 -b 57600
MONITOR for MIPS32 - initialized.
>> d
>> addr: 0x81000000
>> rmm: 16
0x81000000: 0x20000480
0x81000004: 0x00000080
0x81000008: 0x01800400
0x8100000c: 0x00000000
>> a
>> addr: 0x81000000
one instruction per line, empty line to end.
[0x81000000] nop
[0x81000004] nop
[0x81000008]
>> d
>> addr: 0x81000000
>> rmm: 16
0x81000000: 0x00000000
0x81000004: 0x00000000
0x81000008: 0x01800400
0x8100000c: 0x00000000
>> r
R1 (AT) = 0x00000000
R2 (v0) = 0x00000000
R3 (v1) = 0x00000000
R4 (a0) = 0x00000000
R5 (a1) = 0x00000000
  
```

(三) 操作系统移植

目前本 CPU 仅成功移植了 PMON 和 UCore 操作系统, 对于 Linux, 由于时间限制, 没有足够的指令支持以及调试时间, 未能成功运行。

1. PMON

PMON 的运行过程在调试中分为三步: 1. ROM 拷贝 2. 内核自解压 3. 实际代码执行。在调试中, 出现意外的 SPIFlash 破坏和 NAND 读取失败, 经过检查, 定位了错误出现在分支预测和访存部分, 访存部分使用了错误的信号量进行写入完成检查, 分支预测部分则是由于指令队列满导致的执行异常, 在解决这些问题后, 成功运行了 PMON。



2. UCore

Ucore 的运行需要依赖于 PMON 的成功实现，由于 SPIFlash 的大小限制，需要 PMON 将 Ucore 拷贝到内存的指定位置。类似于 BIOS 引导操作系统，不同之处在于此时的 PMON 需要将 Ucore 内核从局域网中下载。

Ucore 在调试的过程中，出现了以下问题：

1. 中断向量入口不正确，Ucore 在启动时，会更改中断入口至内存区，但是由于没有实现 CP0 \$15 号寄存器，中断会被定向到 0xBFC00000 处。此处的汇编对应部分为 PMON 的中断入口，因此会引发错误，导致程序退出。

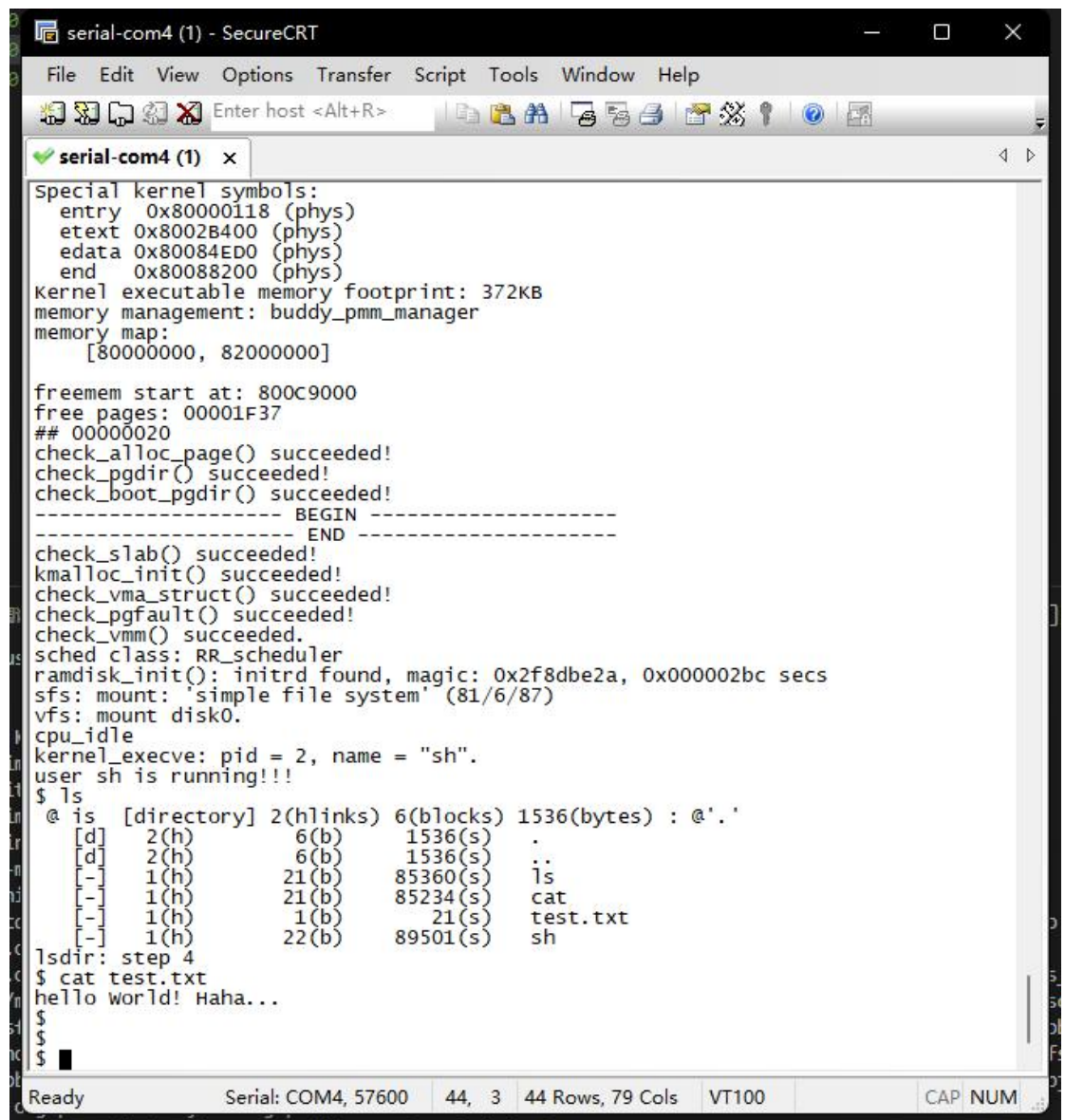
2. TLB 置换错误，由于指令限制，我们将 TLBWR 指令重新实现，通过软件的指定 Index 写入 TLB，但在一开始由于置换 TLB 缺页恒置同一页，导致一条指令在运行时，指令本身和数据会分别引发两次缺页中断，进而导致指令和数据所对应页在执行时不能同时在 TLB 当中。最终导致死锁，用户进程卡死在第一条访存指令。

3. 未实现内核定时器错误 由于 PMON 的运行并不依赖于定时器的实现，导致这个显然的错误并没有暴露出来，但是在 Ucore 在运行用户程序时，在串口输入第一个字

符后，引发的中断只会将需要字符输入而挂起的进程重新加入可运行的队列，但是不实际运行进程。真正的进程切换需要通过内核时钟引发。由于内核时钟没有正确配置，导致系统卡死在 `cpu_idle` 进程而无法顺利切换。

4. 其他错误 得益于 Ucore 在启动前完备的检查机制，内核执行前的报错可以显式的输出。进而快速的解决。

在解决了以上问题后，成功的运行了 Ucore 操作系统，并能够通过 `sh` 运行用户程序。



```
serial-com4 (1) - SecureCRT
File Edit View Options Transfer Script Tools Window Help
Enter host <Alt+R>
serial-com4 (1) x
Special kernel symbols:
  entry 0x80000118 (phys)
  etext 0x8002B400 (phys)
  edata 0x80084ED0 (phys)
  end   0x80088200 (phys)
Kernel executable memory footprint: 372KB
memory management: buddy_pmm_manager
memory map:
  [80000000, 82000000]

freemem start at: 800C9000
free pages: 00001F37
## 00000020
check_alloc_page() succeeded!
check_pgdir() succeeded!
check_boot_pgdir() succeeded!
----- BEGIN -----
----- END -----
check_slab() succeeded!
kmalloc_init() succeeded!
check_vma_struct() succeeded!
check_pgfault() succeeded!
check_vmm() succeeded.
sched class: RR_scheduler
ramdisk_init(): initrd found, magic: 0x2f8dbe2a, 0x000002bc secs
sfs: mount: 'simple file system' (81/6/87)
vfs: mount disk0.
cpu_idle
kernel_execve: pid = 2, name = "sh".
user sh is running!!!
$ ls
@ is [directory] 2(hlinks) 6(blocks) 1536(bytes) : @'.'
[d] 2(h) 6(b) 1536(s) .
[d] 2(h) 6(b) 1536(s) ..
[-] 1(h) 21(b) 85360(s) ls
[-] 1(h) 21(b) 85234(s) cat
[-] 1(h) 1(b) 21(s) test.txt
[-] 1(h) 22(b) 89501(s) sh
lsdir: step 4
$ cat test.txt
hello world! Haha...
$
$
$
```

四、参考设计说明

在总体架构部分，参考了往届北京理工大学参赛队 CPU 的部分架构；在流水线模块中，参考了琉璃晨露的部分设计结构。以上模块均通过 Chisel 重写后整合进本项目当中。在分支预测和流水线 cache 部分，则参考了《超标量处理器设计》中的实现逻辑

使用了 xilinx 的 AXI Crossbar (2.1) , Block Memory Generator (8.4) , Divider Generator 和 Multipiler (12.0) IP 核。

五、项目回顾和未来展望

项目心得和反思:

在系统移植部分, 尽管在上层可以将操作系统简单理解为运行应用程序的程序, 但是如果以这种心态进行底层调试, 会不可避免的陷入错误。尤其是在底层的硬件系统并不稳定的情况下, 这种心态将是致命的---很多时候, 并不是操作系统的代码的问题, 而是底层的偶发性异常产生内核执行错误。此时企图通过打印串口、查看反汇编 (类似设置断点) 等手段, 往往并不能对调试产生帮助。正确的做法是屏蔽内核的 CPU 可能出错的模块, 检查错误是否出现, 逐步定位错误, 或者剥离出错的内核代码, 进行仿真。而不要带入程序开发思维, 企图检查已经发行的内核程序的逻辑错误。

项目回顾:

由于暑假前课程时间紧张和暑期疫情的影响, 又因为在项目初期对 CPU 总体架构和调试缺乏基本的认识。本项目的开发谈不上顺利, 最终的提交的项目与当初提出的架构相比做了非常多的妥协。因此未来的优化方向也主要集中在逐步实现原有的 CPU 架构设计。对于已经实际运行的模块, 例如 Cache 和分支预测等, 也有进一步优化的空间。以分支预测模块为例, 《超标量处理器设计》中介绍的分支预测模块, 其实际的性能和规模受限于 FPGA 的资源 and 查表性能限制, 实际的预测效果并不尽如人意。另外, 尽管开发用的语言支持参数化 CPU 设计, 但由于时间限制和 CPU 各个模块间相互耦合的影响, 我们没有办法通过多次上板来保证每一个模块处于最优设计下, 在未来有进一步的提升空间。

未来展望:

实际的调试过程中, 预赛项目由于有充分的调试技术支持 (diffest), 这部分的调试较为顺利。但是在移植系统的过程中, 系统内核与外设紧密耦合, 又缺乏仿真调试手段, 无法快速的定位错误, 导致效率低下。未来可以考虑通过仿真器, 如 Qemu 等软件, 进行指令级别的仿真, 以生成 Trace 进行比较, 可以大大加快调试进程。在新的调试的手段的支持下, 才能进一步进行更大规模 CPU 的开发和实现。未来, 可能考虑更新分支预测模块, 同时引入多发射处理器结构, 以进一步提升处理器性能。在 CU 方面, 则需要实现更多的指令, 以支持 Linux 操作系统的启动, 以弥补今年的遗憾。

六、参考文献

- MIPS: Architecture For Programmers Volume I—A: Introduction to the MIPS32 Architecture.rev3.02
- MIPS: Architecture For Programmers Volume II—A: The MIPS32[®] Instruction Set
- MIPS: Architecture For Programmers Volume III: The MIPS32[®] and microMIPS32 Privileged Resource Architecture
- 超标量处理器设计
- Xilinx IP: : AXI Crossbar (2.1) LogiCORE IP Product Guide
- Xilinx IP: : Block Memory Generator (8.4) IP Product Guide

Xilinx IP:: Divider Generator (5.1) IP Product Guide

Xilinx IP:: Multipiler (12.0) IP Product Guide

七、致谢

在本项目的开发过程中, 我们获得了来自王娟和李元章老师的方向指导以及富余的开发板资源。同时, 感谢陈康冰学长在项目初期对比赛的经验分享。对我们能够顺利完成系统测试提供了巨大帮助。最后还要感谢北京理工大学机器人队和灵动工厂提供的服务器资源。极大地方便了本项目的仿真测试以及疫情期间的数据资源共享。