基于 MIPS 指令集的单发射六级流水处理器设计

哈尔滨工业大学（深圳）3 队
摸鱼专业户队

一、设计简介

本次大赛我们所提交的设计是一个基于 MIPS 指令集的单发射六级流水处理器。在经典五级流水的基础上，为了减少流水线阻塞所带来的性能损失，我们将取指阶段拆分为取指请求与取指缓存两级，使取指与执行解耦合，充分预取指令以提升性能。

为提升处理器性能，我们进行了 Cache 的开发。ICache 采用二路组相联设计（每路大小为 8KB），使用伪 LRU 替换策略，Cache 行大小为 32 字节。DCache 采用二路组相联设计（每路大小为 8KB），使用伪 LRU 替换策略，Cache 行大小同样为 32 字节，并采用写分配法与写回法，当写未命中时，需要向 axi 发起访存请求，将数据块读入 DCache，再进行修改。若需要被替换的数据块标记为脏，则进行写回。

我们最终所提交的设计，功能测试分数为 100 分。也就是说，我们的作品通过了功能测试的全部 89 个测试点（封装为 AXI 接口）、记忆游戏、性能测试，同时也通过了系统测试，成功启动清华大学提供的监控程序并正确执行其内部集成的 8 个测试程序。与此同时，性能测试方面，我们最终提交版本的 CPU 频率为 60MHZ，IPC 比值为 3.368。

二、设计方案

（一）总体设计思路

1.CPU 总体结构

我们所设计出的 CPU 具有六级流水结构，具体包括取指请求、取指缓存、译码、执行、访存、提交等阶段，如图 1 所示。

处理器在六个阶段所需要完成的主要操作，可简述如下。

- **取指请求阶段** 计算当前的 PC 值，并判断是否需要发送读取指令请求。将得到的 PC 值与读取指令请求发送给 ICache，同时也送入 InstBuffer 中。
- **取指缓存阶段** 读入从 ICache 送来的指令、从取指请求级送来的 PC 值并存储到队列中，根据先入先出原则（First-In-First-Out）向译码级发送指令。
- **译码阶段** 接收从 InstBuffer 送来的指令与 PC 值，从寄存器堆取出操作数，进行指令

译码，并将得到的信息送入执行级。

- **执行阶段** 执行指令、计算结果，并判断是否要向取指级发送分支跳转信息、向 DCache 发送数据请求。
- **访存阶段** 从 DCache 中取回数据，并判断是否发生异常。
- **提交阶段** 将指令所作出的修改提交给通用寄存器以及其它寄存器。

大赛要求 myCPU 封装为 AXI 接口。为满足这一要求，我们对大赛所提供的类 SRAM-AXI 转接桥进行了一定程度的修改，使其适配 Cache 的读写功能。在处理 CPU 的数据交换时，我们仍然实现为类 SRAM 接口。

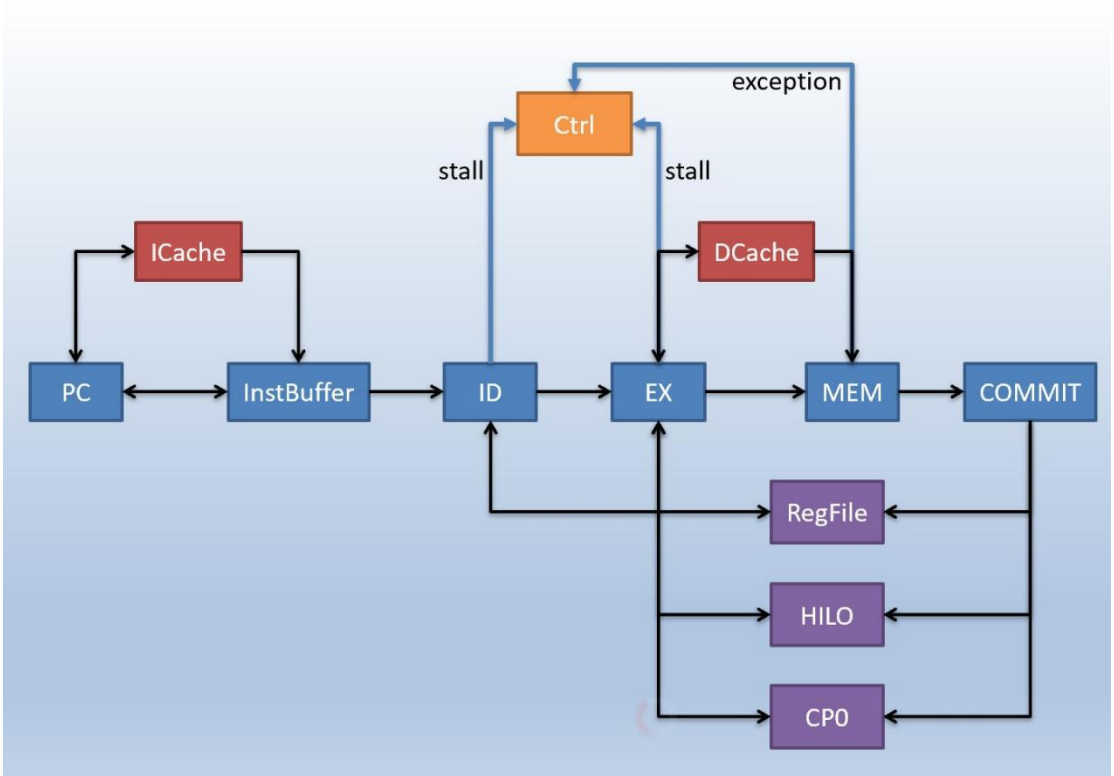

图 1 CPU 总体结构设计

2.指令集

我们实现了大赛所要求的全部指令。对它们按照功能分类，如下所示。

- **逻辑运算指令** AND, ANDI, LUI, NOR, OR, ORI, XOR, XORI
- **移位指令** SLLV, SLL, SRAV, SRA, SRLV, SRL
- **数据移动指令** MFHI, MFLO, MTHI, MTLO
- **算术运算指令** ADD, ADDI, ADDU, ADDIU, SUB, SUBU, SLT, SLTI, SLTU, SLTIU, DIV, DIVU, MULT, MULTU, MUL

- **分支跳转指令** BEQ, BNE, BGEZ, BGTZ, BLEZ, BLTZ, BGEZAL, BLTZAL, J, JAL, JR, JALR
- **访存指令** LB, LBU, LH, LHU, LW, SB, SH, SW
- **自陷指令** BREAK, SYSCALL
- **特权指令** ERET, MFC0, MTC0

3.CP0 寄存器与异常处理

CP0 寄存器定义了 MIPS 指令系统中与特权态相关的功能，用于处理硬件逻辑电路和例外处理软件之间的必要信息交互。我们所实现的 CP0 寄存器如表 1 所示。

编号	名称	功能简述
8	BadVAddr	记录最近一次异常的虚地址
9	Count	计数器
11	Compare	计数器中断控制器
12	Status	状态寄存器
13	Cause	记录最近一次异常的原因
14	EPC	记录最近一次异常的 PC
15	EBase	中断向量基地址寄存器

表 1 所实现的 CP0 寄存器

我们的处理器支持大赛所要求全部异常，具体如表 2 所示。

异常类型	助记符	ExcCode 编码	异常简述
中断	INT	0x00	检测到中断
读地址错	ADEL	0x04	指令地址或读数据地址不对齐
写地址错	ADES	0x05	写数据地址不对齐
系统调用	SYSCALL	0x08	执行系统调用指令
断点	BREAK	0x09	执行断点指令
保留指令	RI	0x0a	执行未实现的指令
整型溢出	OV	0x0c	算术运算溢出

表 2 所支持的异常

（二）取指模块设计

取指模块包括取指请求模块与取指缓存模块。我们将经典五级流水的取指级拆分为两级，以降低取指操作与执行操作之间的耦合度。当流水线因为需要执行除法指令、访存指令而不得不暂停时，取指模块仍然可以继续工作、充分预取指令。这样，当流水线结束暂停状态时，可以从 InstBuffer 中连续取回多条指令进行执行，不必耗费时间等待 ICache 返回数据。

取指请求阶段所发送的地址信号 pc，对其更新逻辑按优先级排列，如下所示。

（1）复位信号 resetn 有效时，更新为 0xbfc00000；

（2）由于异常产生，导致 flush 信号有效时，更新为控制模块所送来的 epc；

（3）执行级送来分支跳转信号有效时，更新为分支地址 branch_addr；

（4）指令队列满，导致 ibuffer_full 信号有效时，保持原有 pc 值不变；

（5）上一次取指请求发送成功时，更新为 pc + 4；

（6）其它情况下，均保持原有 pc 值不变。

取值请求阶段所发送的指令请求信号 inst_req，对其更新逻辑按优先级排列，如下所示。

（1）复位信号 resetn 有效时，不发送请求；

（2）由于异常产生，导致 flush 信号有效时，不发送请求；

（3）指令队列满，导致 ibuffer_full 信号有效时，不发送请求；

（4）上一次取指请求发送成功时，不发送请求；

（5）ICache 可以接收请求，即 inst_addr_ok 信号有效时，发送请求；

（6）其它情况下，保持为上一次的发送值。

取指缓存阶段通过队列存储所取回的指令与 PC 值，对其进行的操作按优先级排列，如下所示。

（1）复位信号 resetn 有效时，清空队列；

（2）flush 信号或 flush_for_exception 信号有效时，清空队列；

（3）branch_flag 信号或 flush_for_branch 信号有效时，清空队列；

（4）当流水线没有发生暂停且队列非空时，从队列中取出一条指令以及相应的 PC 值，发送给译码级；当从 ICache 发送过来的 inst_data_ok 信号有效时，向队列存入一条指令以及相应的 PC 值。上述两个取出、添加数据的操作可以并发进行。

此外，若检测到上一时钟周期 InstBuffer 在向译码级发送数据的同时，流水线刚好由非暂停状态进入暂停状态，则到了下一个时钟周期，InstBuffer 仍然要向译码级发送同样的数据，直至暂停信号撤销。这是因为，该情况下译码级的指令由于暂停机制无法在下一时钟周期进入执行级，倘若立即撤销输送给译码级的数据，将会导致该指令停留在译码级而没有出

现在执行级。因此，必须保持原有数据直至暂停结束。

（三）译码模块设计

译码模块接收到来自 InstBuffer 的数据后，对指令进行译码，确定运算的操作数、操作类型、操作子类型等信息，并传递给执行级。

译码模块通过前递、暂停两种方式解决数据冒险。对于 read-after-write 型冒险，译码级会根据从执行级、访存级前推过来的修改信息生成运算的操作数，以确保使用的是寄存器堆的最新值；对于 load-use 型冒险，译码级会向控制模块发送暂停请求，停止从 InstBuffer 接收指令并停止将译码信息传递给执行级，直至从 DCache 中取回所需数据。

此外，译码阶段还会检查指令地址错、断点、保留指令、系统调用等异常，并将发现的异常信息传递到之后的流水级当中。

（四）执行模块设计

执行模块接收到来自译码模块的信号后，根据传递而来的操作数、运算类型、运算子类型计算出结果。对于一个周期内就能获得运算结果的指令，如逻辑运算指令、移位指令、数据移动指令等，无需进行停顿操作，直接将写回信息传递到下一级。

数据移动指令要求从 CP0 寄存器、HILO 寄存器取回数据写回到通用寄存器中，这一过程同样可能会发生 read-after-write 型数据冒险。因此，执行级需要接收从访存级、提交级前递回来的信息，以确保写回的是最新值。

对于经过多个周期才能获得运算结果的指令，如除法指令、访存指令等，执行模块需要暂停流水线的执行。对于除法指令，执行级会向控制模块发送暂停请求，停止 InstBuffer 传送数据给译码级、停止译码级传送数据给执行级、停止执行级传送数据给访存级，直至除法运算完成后撤销暂停请求。对于访存指令，执行级同样要执行类似的暂停操作。不同之处仅在于执行级发送暂停请求前需要先判断当前指令是否会产生异常（如地址错异常），如果会产生异常，则不发送暂停请求，并将异常信息传递到下一流水级。

执行模块还承担了发送分支信息的任务。当检测到送入执行级指令的分支结果为跳转时，需要向取指请求模块、取指缓存模块发送跳转信息，通知它们改变取指地址或清除掉已经预取的错误指令。值得一提的是，分支信息必须要确保在延迟槽指令已经进入译码级之后才能发送，否则，有可能导致延迟槽指令还未发送给译码级的时候就被清除掉，造成错误结果。

执行模块能检测出地址错异常、整型溢出异常，并将异常信息传送到之后的流水级当中。

（五）访存模块设计

访存模块负责接受从 DCache 传递过来的数据，进行处理后送入到提交阶段。观察图 1 不难发现，在处理数据读写时，PC、ICache、InstBuffer 三者与 EX、DCache、MEM 三者有着高度相似的关系。PC 向 ICache 发送数据请求与请求的地址，当数据成功返回时，由 InstBuffer 负责接收数据；同样，EX 向 DCache 发送数据请求与请求的地址，当数据成功返回时，由 MEM 负责接收数据。

访存模块的另一个主要职责是提交异常。当发现异常时，访存级会向控制模块和 CP0 寄存器发送异常信息，以便尽快清除流水线并进行异常处理。访存阶段能检测到的新异常只有中断，这是因为其它类型的异常早已在译码级、执行级就被发现，只不过所有的异常信息都要被传递到访存级才会被统一结算。

（六）提交模块设计

提交模块负责将指令执行后需要写回的数据提交给各寄存器，包括通用寄存器、CP0 寄存器、HILO 寄存器。

（七）ICache 设计

ICache 置于 PC 与 InstBuffer 之间，可以接受 PC 的请求，命中时将取指结果返回 InstBuffer，未命中时向 axi 发起访存请求。

ICache 使用 VIPT，以避免 PIPT 在访存指令之前每次访问 TLB 进行虚实地址转换造成的时间开销，亦或 VIVT 产生的重名问题。ICache 实现为二路组相联，每路大小为 8KB，Cache 行为 32 字节，使用地址中的[11:5]进行索引，[31:12]为 Tag 域。替换策略为伪 LRU 替换策略。

ICache 状态机设计为有 4 个状态，如下所示。

- IDLE：等待请求状态；当接收到来自 PC 的访存请求时，进入 TAG COMPARE 状态。
- TAG COMPARE：比较 tag 域与 valid 域，从而判断是否命中；命中则将对应数据发送给 InstBuffer，返回 IDLE 状态；若未命中，则进入 ALLOCATE 状态。
- ALLOCATE：向 axi 发送访存请求并等待数据返回；当数据返回时，则进入 WRITE BACK 状态。
- WRITE BACK：将读回的数据写入 cacheline 并存入对应 bank，同时将对应数据发送给 InstBuffer，完成后回到 IDLE。

ICache 状态机的状态转换图如图 2 所示。

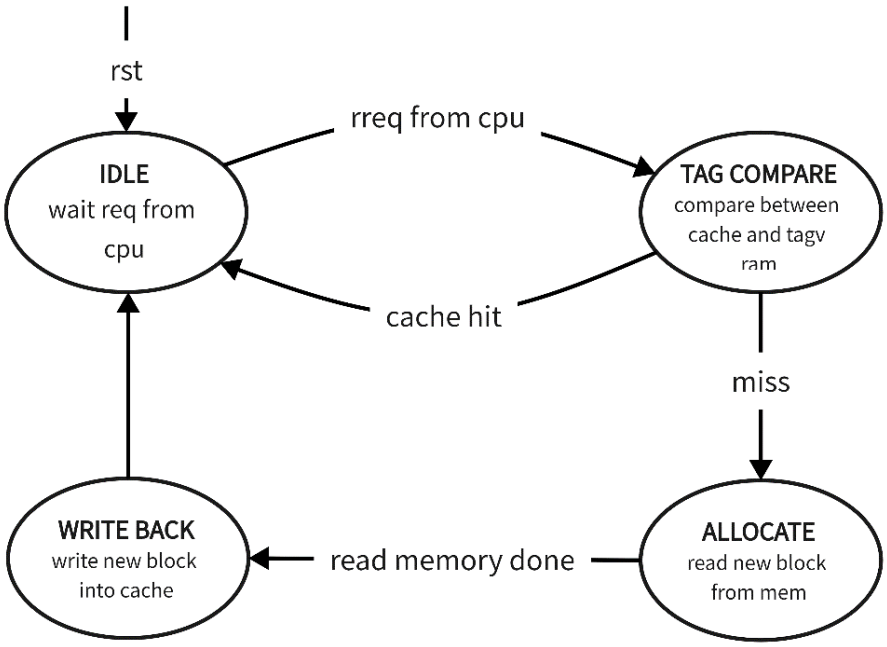

图 2　ICache 状态转换图

为了将 ICache 与 axi 等交互进行解耦合，我们将包括虚实地址转换、uncached 判断、发送接收来自 axi 的信号、处理以块为单位的读写等功能都封装至顶层模块 Cache_top。

（八）DCache 设计

DCache 在 CPU 访存阶段接受访存信号，处理完成后送回请求对应的数据。

DCache 实现为二路组相联，每路大小为 8KB，Cache 行为 32 字节，使用地址中的[11:5]进行索引，[31:12]为 Tag 域。采用伪 LRU 替换策略。采用写分配法与写回法，当写未命中时，需要向 axi 发起访存请求读入数据块进入 DCache，再进行修改。若需要被替换的数据块标记为脏，则需要写回。

基于上述逻辑，设计 DCache 状态机为有 5 个状态，如下所示。

- IDLE：等待来自 CPU 的读或写指令与数据。
- LOOKUP：比较 tag 域与 valid 域，判断是否读或写命中；若命中，则将对应数据放置于输出端口，并返回 IDLE；若未命中，则进入 MISS 状态。
- MISS：开始处理读或写缺失，等待 axi 的 wready 信号。
- REPLACE：若需要替换的数据块为脏，则读出该数据块，并等待 axi 的 rready 信号。
- REFILL：向 axi 发出访存请求，将缺失的数据块读回并写入 DCache，同时等待 rvalid 信号。

DCache 状态机的状态转换图如图 3 所示。

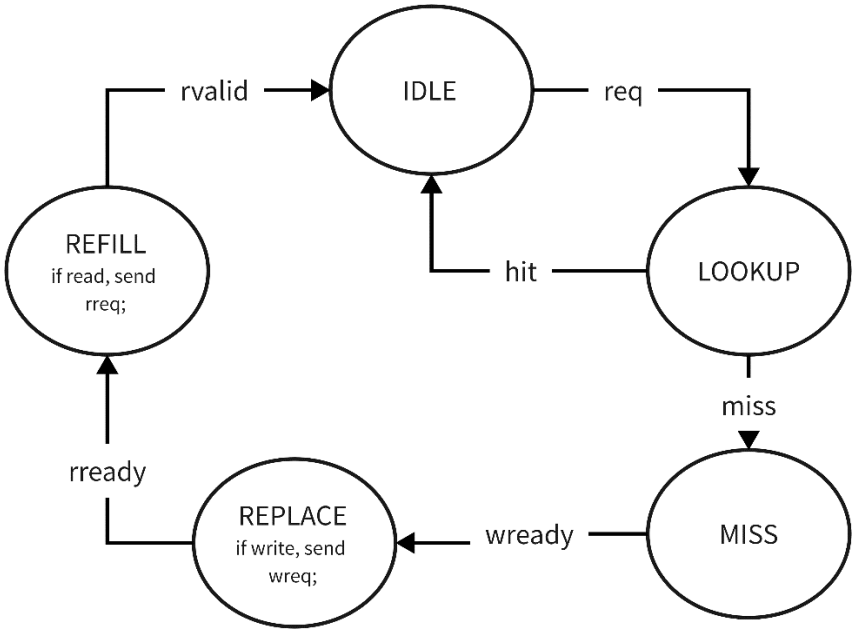

图 3　DCache 状态转换图

为了将 DCache 与 axi 等交互进行解耦合，我们将包括虚实地址转换、uncached 判断、发送接收来自 axi 的信号、处理以块为单位的读写等功能都封装至顶层模块 Cache_top。

三、设计结果

我们所提交的设计，成功通过了系统测试，可以启动清华监控程序并正确执行其内部集成的 8 个测试程序，运行结果如图 4 与图 5 所示。

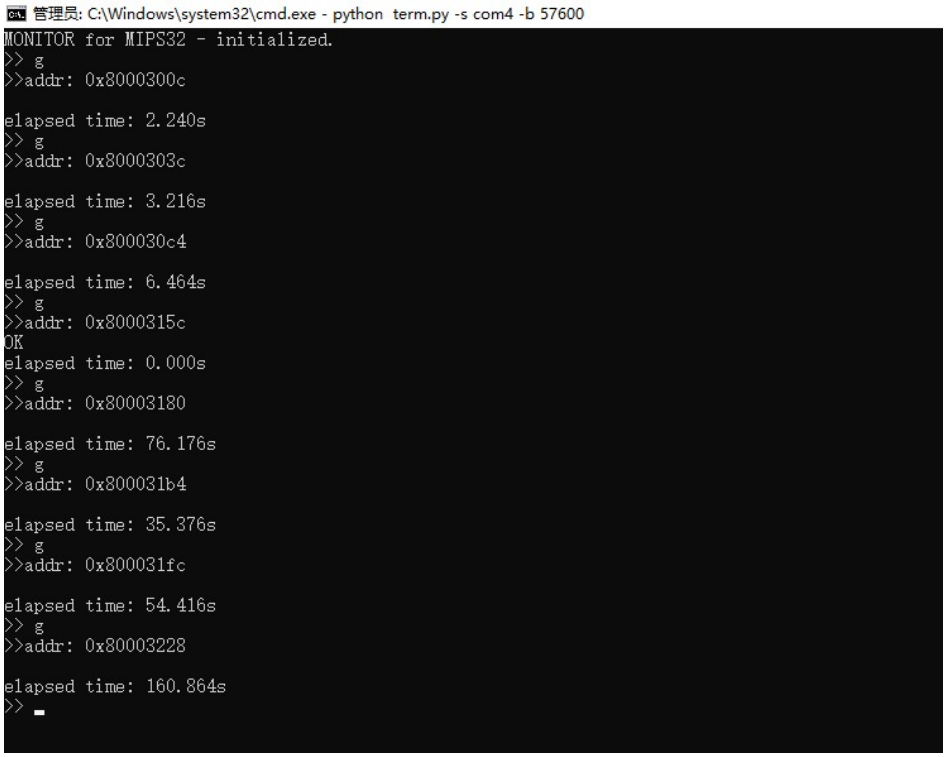

图 4　系统测试运行成功截图（一）

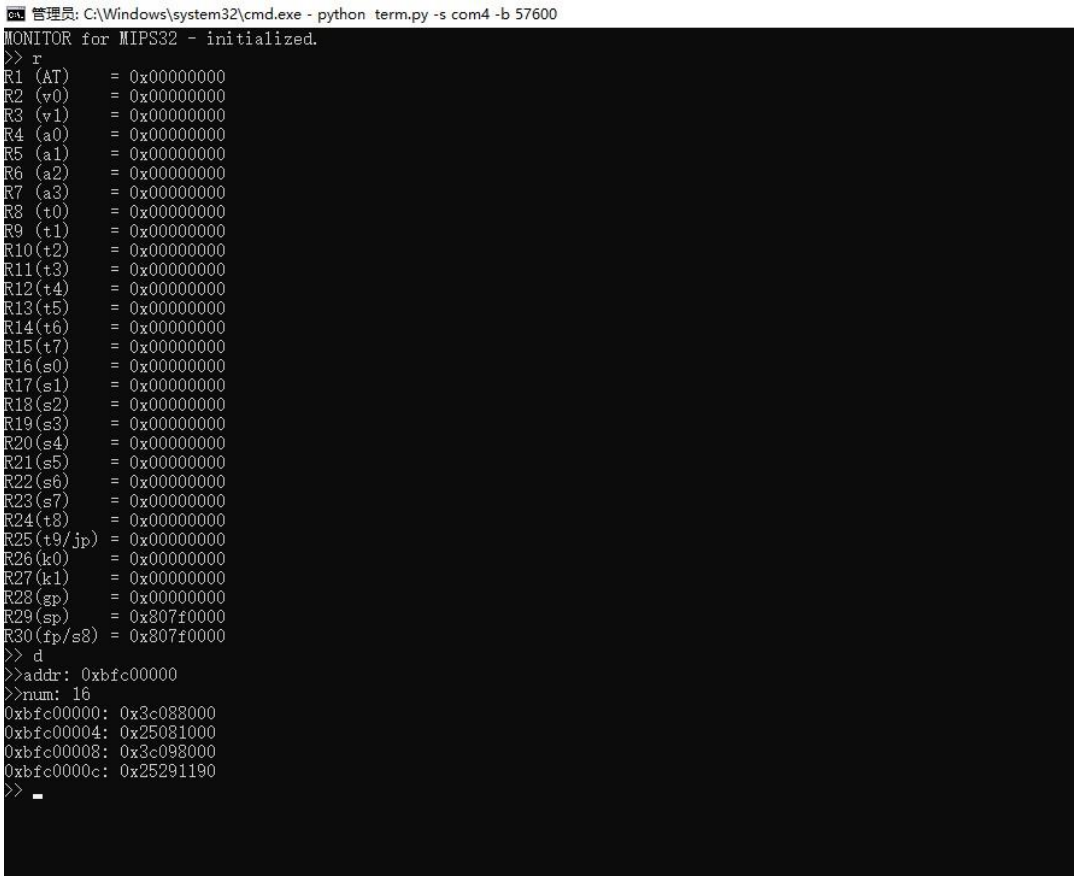

图 5　系统测试运行成功截图（二）

四、参考设计说明

我们在进行设计的时候，参考了雷思磊的五级流水线 OpenMIPS 以及 2020 年“龙芯杯”团队赛一等奖作品 UltraMIPS、2021 年“龙芯杯”团队赛二等奖作品 ZirconMIPS，沿用它们的部分命名与数据通路，并进行了一定程度的修改。

五、参考文献

- 雷思磊. 自己动手写 CPU. 电子工业出版社
- 汪文祥, 邢金璋. CPU 设计实战. 电子工业出版社
- 姚永斌. 超标量处理器设计. 清华大学出版社
- 李程浩, 刘定邦, 宫浩辰, 任翔宇. UltraMIPS 设计文档
- 朱旗, 尚奕扬, 邓起源, 丁浩卓. ZirconMIPS 设计文档