计算机系统综合实验报告

计21 韦福超 2012011392

计23 张振宁 2012011325

计23 甄显安 2012080060

目录：

1. [CPU](#CPU)
2. [CPU的概述](#CPU的概述)………………………………………………………………………………………………………2
   1. [指令集](#指令集)………………………………………………………………………………………………………..2
   2. [CPU的总体介绍](#CPU的总体介绍)…………………………………………………………………………………………4
3. [CPU的部件](#CPU部件)………………………………………………………………………………………………………5

2.1 [Controller](#Controller)…………………………………………………………………………………………………….5

2.2 [ALU](#ALU)………………………………………………………………………………………………………………6

2.3 [p\_MUL](#p_MUL)………………………………………………………………………………………………………….6

2.4 [RegistersFile](#RegistersFile)…………………………………………………………………………………………………6

2.5 [mm\_manager](#mm_manager)………………………………………………………………………………………………6

2.6 [顶层模块CPU](#顶层模块CPU)…………………………………………………………………………………………….8

3. [CPU的调试功能](#CPU的调试功能)………………………………………………………………………………………………9

二．[ucore操作系统](#Ucore操作系统)

1. [stdio.c](#stdio)………………………………………………………………………………………………………………….9

2. [intrinsic.c](#intrinsic)……………………………………………………………………………………………………………..9

2.1 [\_Alloc](#Alloc)………………………………………………………………………………………………………….9

2.2 [\_PrintInt](#PrintInt)……………………………………………………………………………………………………..10

2.3 [\_PrintChar](#PrintChar)…………………………………………………………………………………………………..10

2.4 [\_PrintString](#PrintString)………………………………………………………………………………………………..10

2.5 [\_PrintBool](#PrintBool)…………………………………………………………………………………………………..10

2.6 [\_ReadLine](#ReadLine)…………………………………………………………………………………………………..10

2.7 [\_StringEqual](#StringEqual)……………………………………………………………………………………………….10

2.8 [\_ReadInteger](#ReadInteger)……………………………………………………………………………………………..10

三．[编译器](#编译器)…………………………………………………………………………………………………………….10

1. [流出指令的修改](#流出指令的修改)……………………………………………………………………………………………….10

2. [传参的修改](#传参的修改)……………………………………………………………………………………………………….10

3. [函数返回的修改](#函数返回的修改)……………………………………………………………………………………………….11

4. [库函数的实现](#库函数的实现)……………………………………………………………………………………………………11

四．[实验结果](#实验结果)…………………………………………………………………………………………………………11

五．[实验总结与体会](#实验总结体会)……………………………………………………………………………………………..13

1. CPU
2. CPU的概述

1.1 指令集

在本次实验中，实现的基于简化的MIPS32指令集的多周期CPU。这里，将实现的指令罗列如下：

简化的MIPS32指令集

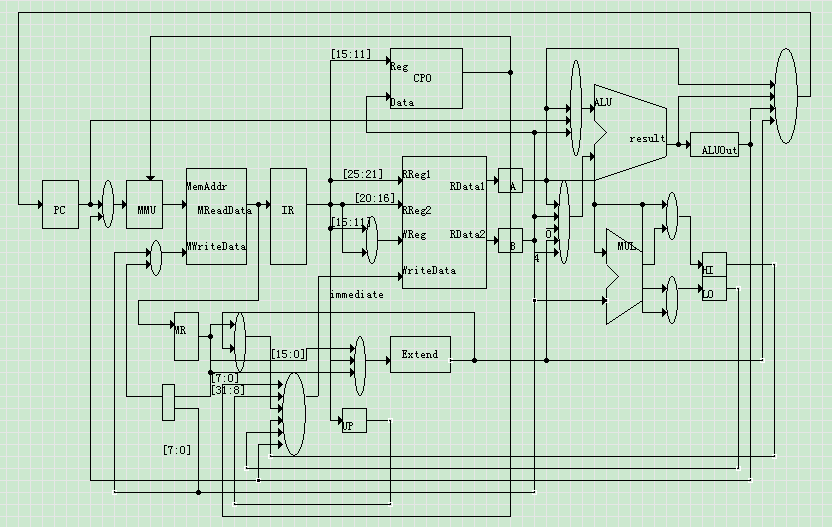
|  |  |  |
| --- | --- | --- |
| 名称 | 指令格式 | 功能 |
| addiu | addiu d,s,j | d=s+(signed)j |
| addu | addu d,s,t | d=s+t |
| slt | slt d,s,t | d=((signed)s<(signed)t) ? 1:0 |
| slti | slti d,s,j | d=((signed)s<(signed)j) ? 1:0 |
| sltiu | Sltiu d,s,j | d=((unsigned)s<(unsigned)j) ? 1:0 |
| sltu | sltu d,s,t | d=((unsigned)s<(unsigned)t) ? 1:0 |
| subu | subu d,s,t | d=s-t |
| mult | mult s,t | Hi|Lo=(signed)s \* (signed)t |
| mflo | mflo d | d=Lo |
| mfhi | mfhi d | d=Hi |
| mtlo | mtlo s | Lo=s |
| mthi | mthi s | Hi=s |
| beq | beq s,t,label | If (s==t) goto label |
|  |  |  |
| bgez | bgez s,label | If (s≥0) goto label |
| bgtz | bgtz s,label | If (s>0) goto label |
| blez | blez s,label | If (s≤0) goto label |
| bltz | bltz s,label | If (s<0) goto label |
| bne | bne s,t,label | If (s!=t) goto label |
| j | j label | goto label |
| jal | jal label | goto label ; $ra = rpc |
| jalr | jalr d,s | $pc=s ; d=rpc |
| jr | jr r | $pc=r |
| lw | lw t,addr | t = \*((int\*)(addr)) |  |
| sw | sw t,addr | \*((int\*)addr)=t |  |
| lb | lb d,addr | d=\*((signed char\*)addr) |  |
| lbu | lbu d,addr | d=\*((unsigned char\*)addr) |  |
| sb | sb t,addr | \*((char\*)addr)=t |  |
| and | and d,s,t | d=s & t |  |
| andi | andi d,s,j | d=s & (unsigned)j |  |
| lui | Lui t,u | t=u<<16 |  |
| nor | nor d,s,t | d=~(s | t) |  |
| or | or d,s,t | d=s | t |  |
| ori | ori t,r,j | d=s | (unsigned)j |  |
| xor | xor d,s,t | d=s ^ t |  |
| xori | xori d,s,j | d=s ^ j |  |
| sll | sll d,s,shf | d=s << shf; /\* 0≤shf<32 \*/ |  |
| sllv | sllv d,t,s | d=t << (s%32) |  |
| sra | sra d,s,shf | d=(signed)s >> shf |  |
| srav | srav d,s,t | d=(signed)s >> (t%32) |  |
| srl | srl d,s,shf | d=(unsigned)s >> (t%32) |  |
| srlv | srlv d,s,t | d=(unsigned)s >> (t%32) |  |
| syscall | syscall B | exception(SYSCALL,B) |  |
| cache | cache k,addr | 不做cache，视作nop |  |
| eret | eret | $pc=EPC |  |
| mfc0 | mfc0 t,cs | t=cs |  |
| mtc0 | mtc0 t,cd | cd=t |  |
| tlbwi | tlbwi | 写索引TLB项 |  |
| lhu | lhu d,addr | t=\*((unsigned short\*)addr) |  |

其中rpc在流水线有延迟槽情况下，应为再下一条指令的地址。由于本实验实现的多周期版本，直接令其为下一条指令地址即可。Hi，Lo是乘法器的寄存器，分别为32位，记录结果的高位和低位。

在使用交叉编译器编译ucore时，可能会产生伪指令，常见的有：li，la，move等，在decaf和C0编译器中可能产生的伪指令有sge，sgt，b等，具体的宏定义可以参见*see MIPS run*的指令集章节。

* 1. CPU的总体介绍

CPU的数据通路如下：



是多周期CPU，可以简单地将指令的执行划分为：取指（instruction\_fetch），译码（decode），执行（execute），访存（mem\_access），写回（write\_back），另外异常处理有4个状态（interrupt，interruptx，interrupt2，interrupt3），在CPU启动前有一个初始状态（initialize）。

大多数指令，在4个周期就可以执行完，详见附件中“控制信号与指令.xlsx”，简单地说，除了lw，lb，lbu等访存指令以及异常处理外，其他指令基本都可以在3~4个周期完成，因此CPI平均来看接近4。

为了使CPU主频更高，同时使访存更加稳定，采取的方法是让控制器（Controller）和访存管理器（mm\_manager）采用不同的时钟，每次Controller通过发送MemRead或者MemWrite信号通知mm\_manager可以开始访存，mm\_manager于是根据mmu转换得到的物理地址向相应设备发送“读”或者“写”的信号，Controller、mm\_manager此时处于等待状态；当相应设备完成访存后，将访存结果返回给mm\_manager，同时拉高电平，告知已完成，mm\_manager于是拉高电平ready，并将数据返回给Controller，Controller得知ready后，继续进入下一个状态。因此，Controller时钟有望达到最高频50M，mm\_manager由于各个设备也是采用不同的时钟，也可以至少有11.0592M的性能。但是，最终的结果是，由于25M在教学机上就不稳定了，所以，只好统一使用11.0592M的时钟。

最后的演示结果，执行用户程序的速度，一方面和CPU有关，一方面和操作系统时间片划分有关，总体来看，交互速度不是很快。

1. CPU部件

下面介绍CPU的部件，其中使用的信号名称可以参见“控制信号与指令.xlsx”。

总共有Controller，ALU，p\_MUL，RegistersFile，mm\_manager五个重要部件，分别完成指令解析、控制信号的生成，计算，访存与存储的功能，mm\_manager下辖5个设备：片上Rom，Ram1，Ram2，Flash，串口，内置TLB，完成mmu的工作。

2.1 Controller

Controller是控制器，负责指令的解析以及控制信号的生成，其工作主要是完成一个大的状态机的转换。状态主要有：

Initialize：初始化各种信号，每当rst按下，就将进入此状态，随后，在时钟跳变沿来临后，转入instruction\_fetch

Instruction\_fetch：这里，访存读取指令，并且，也将中断的判断放在了这个阶段。其工作原理是：根据CP0协处理器的状态寄存器Status以及timer\_Int，com\_Int判断是否是中断开启状态，如果开启，中断的电平是否拉高，如果的确发生中断，就不再读取当前指令，而是跳入异常处理流程，开始中断的处理。否则，指令正常流入。没有访存错误后，就跳入decode状态。

Decode：decode阶段并没有做太多事情，而是应时序要求，将访存信号清零，同时PCWrite<=’1’，这样下个周期PC就更新为PC+4。同时计算PC+immediate，其中immediate的生成详见Extend部件，该部件根据控制信号与指令instruction的低26位，来生成立即数immediate，这里，是instruction第16位左移两位后符号扩展；因此，这里假定是分支指令，预先算好目标地址。

Execute：前两个状态是每条指令共用的，且未对指令的不同作区分。在这个状态，由于已经得到了指令instruction，从而可以进行译码。因此，译码其实主要在这个状态进行。

译码是按照前述MIPS指令集进行，对未知指令，采取跳入异常处理流程，由操作系统来解决的办法，在实验中，实际的未知指令可以被有效处理的，只有div除法指令，其他的非法指令都没有处理。

在这个状态，涉及每条指令的具体处理方式，其中算术逻辑指令，基本都是同样的方法，只不过ALUOp即ALU的操作因指令而异；而乘法指令，采用ISE内置的并行乘法器，包装在p\_MUL部件中，仍是采用类似访存的手法，通过信号mul\_ready来指示Controller继续执行。这里，对于几个特权指令，都在此阶段予以判断，eret，tlbwi，mfc0，mtc0都予以了特权级的检查，若是用户态执行，将进入异常处理流程，由操作系统来处理，杀死用户线程。这个阶段，主要根据各个不同指令，准备好ALU，MUL，mm\_manager的控制信号，以便下一步继续进行。

Mem\_access：这个阶段，应该只有几个访存指令会进入，lw，sw，lb，lbu，sb，lhu。这里，sw结束后，直接返回instruction\_fetch；而lw，lb，lbu，lhu实际都要进行MemRead，读取数据。其中，由于每次都是对一个地址上的4个字节即一个字进行操作，因此，sb必须先将地址里的整个字读出，然后在对应字节写入，在write\_back阶段再整个字写回原地址，注意，不同于lw，lb，lbu，lhu，sb这个阶段结束后，地址里的数据已得到。

Write\_back：这里大部分指令操作都相同，而lw，lb，lbu，lhu实际在这个阶段才得到数据，并且准备写回寄存器。

Interrupt：这是中断和异常要进入的第一个状态，主要是拉高EPCWrite，set\_Cause，set\_EXL的电平，准备将正确的返回地址写到EPC，并且将异常的原因写到Cause，同时准备将Status(1)置1，即关闭中断，由内核态执行。并且，把PCWriteCond置为“111”，意即PC的下一个地址值将是EBase+0x180。

Interruptx：只是为了时序要求，添加了这个阶段，将之前的信号清零。

Interrupt2：拉高PCWrite，更新PC。

Interrupt3：拉低PCWrite，进入instruction\_fetch。

* 1. ALU

根据两个源操作数，由ALUOp指示进行何种算术逻辑运算，是组合逻辑电路，简化起见，只实现指令集中必需的几种ALU操作：ADD，SUB，AND，OR，NOR，XOR，SLL，SRA，SRL，compare(符号扩展后的比较)，compareu(零扩展后的比较)，其中，为了和Controller协调，实际若A<B，返回的是1，A=B，返回0，A>B，返回-1。

* 1. p\_MUL

使用ISE内置的并行乘法器，经16个时钟周期，得到结果，并置ready=1，将结果返回顶层模块。

* 1. RegistersFile

寄存器堆，32个寄存器，在RegWrite上升沿发生时，写寄存器。根据RegAddr1和RegAddr2读出寄存器值。

* 1. mm\_manager

这是访存部分的上层模块，下辖6个功能部件，其中TLB负责mmu，其外是5个设备。存储地址划分如下：

虚拟地址：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **虚拟地址** | **虚拟地址范围** | **权限** | **MMU** | **物理地址范围** |
| kuseg | 0x00000000-7FFFFFFF | 内核/用户 | 是 | ~ |
| kseg0 | 0x80000000-9FFFFFFF | 内核 | 否 | 0x00000000-0x1FFFFFFF |
| kseg1 | 0xA0000000- BFFFFFFF | 内核 | 否 | 0x00000000-0x1FFFFFFF |
| kseg2 | 0xC0000000- FFFFFFFF | 内核 | 是 | ~ |
|  |  |  |  |  |

这里对kseg0，kseg1而言，只要把地址高3位清零，就得到其对应的物理地址，无mmu转换。

物理地址：

|  |  |
| --- | --- |
| **设备** | **物理地址范围** |
| 片上ROM | 0x1FC00000-0x1FC00FFF |
| RAM1 | 0x00000000-0x003FFFFF |
| RAM2 | 0x00400000-0x007FFFFF |
| Flash | 0x1E000000-0x1EFFFFFF |
| 串口 | 0x1FD003F8-0x1FD003FC |

TLB：执行虚拟地址到物理地址的转换。对于虚拟地址“0x80000000”到“BFFFFFFF”的虚拟地址，将高3位清零，即为物理地址。

对于需要mmu转换的，这里，TLB内置16个63位的TLB表项，每项只记录一个虚页号，两个实页号，以及valid和dirty标志，可以将连续两个虚页映射到两个不同的物理页。不维护进程ASID、全局标记G，默认所有的G为1。

TLB表项

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | VPN2 | PFN1 | D1 | V1 | PFN0 | D0 | V0 |
| 范围 | 62-44 | 43-24 | 23 | 22 | 21-2 | 1 | 0 |

这里表项的组成和MIPS32的规定有所不同。每次由虚拟地址的高19位作为VPN2，第13位用于TLB中每一项两个PFN的选取，低12位作为页内偏置。

TLB表项的更新是以TLBWrite的上升沿为标志的，这时，Index的第4位作为索引，从16个tlb表项中找到对应表项，然后将EntryHi，EntryLo1，EntryLo0的信息填写到对应位置。

片上ROM：存放bootloader，负责从flash中加载ucore进内存，然后跳到kernel的入口，并不再返回。

Device\_Ram1：内存。

Device\_Ram2：内存。

Device\_Flash：本实验中，将Flash作为只读存储，在实验前，先将ucore，以及文件系统映像烧写进Flash中。之后，bootloader会将其ucore载入内存。

Device\_COM：

串口1：0xBFD003F8(虚地址)，这个地址和PC终端作为数据的交互。

串口2：0xBFD003FC(虚地址)，这个地址用来得到串口通信的状态，如果串口正在被占用，就不能再使用串口。

对mm\_manager而言，每次还要向上层模块CPU反馈访问设备的情况，具体放在mem\_error里，表明访存是否有错，如果出错，指示错误类型，并将虚拟地址赋给BadVAddr。

* 1. 顶层模块CPU

CPU是顶层模块，负责这几个重要部件信号与数据的传送，协调整个系统的运行。这里，重点介绍CP0协处理器，因为，为了方便起见，CP0的所有寄存器都放在这个模块下。

CP0协处理器，按照MIPS32体系，主要负责异常、中断机制，存储单元控制，以及时钟控制等等，是比较重要的组成部分，这里，为了简化实验，只实现了以下几个必要的寄存器：

Status：控制中断、异常的状态，实验中，其实只用到其中的5位。

Status(0)：使能中断，为1时开中断，但还需受到Status(1)的影响。

Status(1)：EXL，为1时屏蔽所有中断。

Status(4)：为1时是用户态。因为Status(4-3)是CPU的3种状态，00内核态，01监管态，10用户态。

Status(10)：使能串口中断。

Status(15)：使能时钟中断。

Index：TLB的索引项。由于只有16个TLB表项，因此实际只需要用到低4位。

EntryHi：前19位作为VPN2，来索引物理页表。

EntryLo0：其25到6位为偶数页表项，2到1位为dirty和valid。其中dirty标示该页是否是脏页，以决定该页是否可写；而valid位则标示该页是否有效。

EntryLo1：奇数号页表。

BadVAddr：与异常及中断相关，指明出现异常的地址。

Count：每次一个时钟周期就自加1。

Compare：与Count构成很好的时钟中断控制器，一般由操作系统来控制Compare，每次Count达到Compare时就产生一个时钟中断。

Cause：异常的种类，实验只涉及了其中的低16位。其中第6到2位，是异常类型的掩码，表示异常的原因，第10位是串口中断，15位是时钟中断。

EPC：异常返回的PC值。要根据异常或者中断发生的位置来区分。这里，由于执行译码前就先判中断，因此中断就有EPC=PC，而对于异常，由于已经执行完译码，PC已经加4，因此，EPC=PC-4，对于系统调用syscall，由于ucore中维护了EPC=EPC+4，因此，也是EPC=PC-4。

EBase：EBase一般为0x80000000，EBase+0x180作为异常处理流程的入口地址。

1. CPU的调试功能

为了方便CPU的调试，在片上也实现了单步调试的功能。每次运行前用拨码开关指定断点，然后rst执行，如果PC=breakpoint，时钟就会变为单步时钟，此时可以用拨码开关拨出所需查看的信号，这时LED灯上就能看到该信号的具体数值。这可以见附件“硬件调试信号拨码开关一览表.txt”。

二．Ucore操作系统

Ucore操作系统的移植，使用了往届武祥晋移植的MIPS版ucore，本实验对内核态的ucore几乎没有做任何修改，只是添加了SYS\_alloc系统调用，作为动态分配堆栈的方法（但好像并不能正常分配）。为了支持decaf编译器，主要修改的是其用户态的库函数user/libs/。现将增改的方法列举如下：

1. stdio.c

实现头文件stdio.h中的readline函数，用于从终端读取一行字符串。

因此在user/sh.c中，删除readline的实现。

1. intrinsic.c

这里增加的函数主要用于decaf编译器的使用。

* 1. \_Alloc

动态分配内存。由于ucore不支持内存的动态分配（也许是我仓促之下没有找到这样的方法来调用），这里，在该文件中开了一个4KB的静态数组，模拟堆的行为。由于decaf编译器只负责内存的分配，并不回收内存，因此，出于简化的考虑，具体实现中，也不过是使用一个指针来指向静态数组中待分配位置。

* 1. \_PrintInt

直接调用cprintf

* 1. \_PrintChar

直接调用cprintf

* 1. \_PrintString

直接调用cprintf

* 1. \_PrintBool

直接调用cprintf

* 1. \_ReadLine

首先调用readline获得该字符串首指针。这里，decaf给实现带来不方便之处在于其\_ReadLine是不带参数的，这样如果直接返回readline得来的首指针，就会导致指向readline里的那个静态内存区(可以参看readline的实现，里面只有一个静态的数组，作为接收的缓存)，这样，下一次用\_ReadLine得到字符串后，这个指针指向的内容就不是自己期望的。因此，这种不安全的读字符串的必将导致出错。因此，不得不自己再故技重施，开了另一个静态数组用来模拟堆的行为。这样的实现的确不是很好，如果有时间，自然是要调系统函数来支持分配，遗憾的是暂时没有找到。

* 1. \_StringEqual

调用strcmp

* 1. \_ReadInteger

调用readline后，使用strtol

三．编译器

这里对两款编译器中的decaf编译器做介绍，另一款C0类似。

采用的中间代码优化仍是原框架的活跃变量分析，求解数据流方程，下面主要介绍MIPS代码的生成，主要的改动在：decaf.backend内。

1. 流出指令的修改

首先，要解决指令的问题，不能产生一些不能处理的指令或者伪指令。

在genAsmForBB中：

ADD的add不能识别，改成addu；

SUB的sub不能识别，改成subu；

MUL的mul改成mult和mflo两条指令；

DIV的div改为div和mflo；

MOD中的rem改为div和mfhi；

NEG中的neg改为subu，用$zero减去操作数。

1. 传参的修改

由于目标平台约定传参时先将头四个参数赋给a0~a3,剩下的参数再压栈，而decaf框架中，使用的方式是全部参数依次压栈。因此，有必要对decaf的传参方式进行修改。但是，在genAsmForBB的PARM中改动传参方式是很容易的，而在emitProlog中又如何得知参数具体的情况呢？要知道，原框架在函数体里时只管从栈里取参数。如果完全按照前述要求来改，肯定要有大的代码调整，此时更改三地址码的生成，或许可以简化修改，但仍显复杂。因此，采用的办法是，一边将前4个参数送给a0~a3，一边仍是将参数统统压入栈中。

1. 函数返回的修改

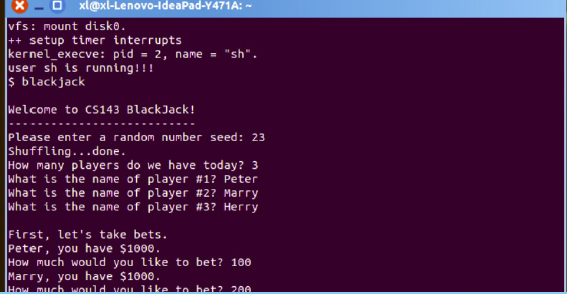
在emitTrace中BY\_RETURN，由于操作系统要求用户程序返回需有返回值，因此，即便返回类型为void的函数，仍是返回0。

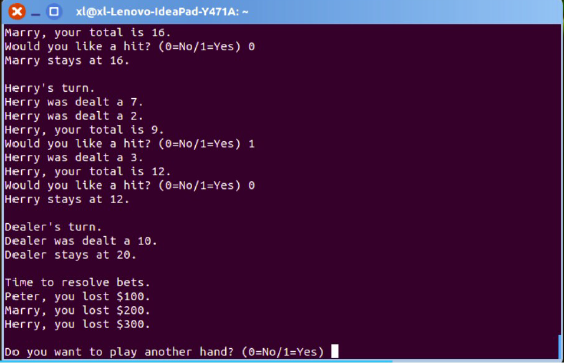
1. 库函数的实现

见操作系统部分的实现。

四．实验结果

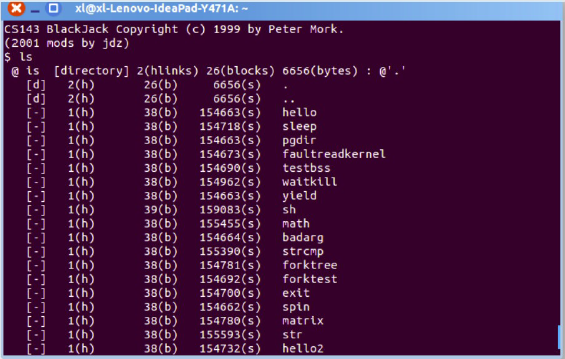
通过本实验，实现了一个简化的MIPS32体系的CPU，并在其上移植了ucore操作系统，负责CPU的计算资源以及存储资源的调度，并且完成了编译器，使得用decaf或者C0高级语言编写的程序能最终在CPU上运行，并得到正确结果。最终的演示是lab8操作系统shell下的交互，这里给出运行decaf程序blackjack.decaf的截图：







给出ls命令查看文件：



五．实验总结与体会

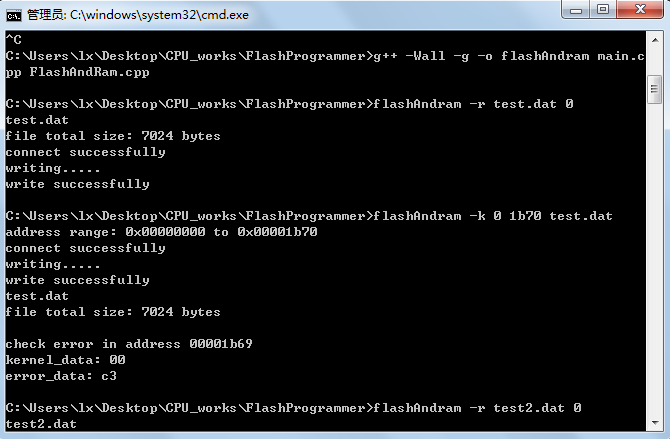
通过一个月的学习与实践，终于实现了基于简化MIPS32的多周期CPU，并且成功移植ucore操作系统，实现编译器后端，最后呈现了一个简单的shell交互界面，完成了一个整机的系统。收获、体会都是很大的。

首先，实现CPU需要扎实的硬件知识，必须对整个CPU执行指令、变换状态的逻辑时序有一个清晰的认识，对每条指令，都要区分其各自的执行方式。相较于《计算机组成原理》课，这次的实验，指令集改为MIPS32，指令条数为常用的47条，并且加入了中断、异常处理机制，增加了mmu地址转换，增加了CP0协处理器，而且对指令的特权级、未知指令等进行了严格的检查，可以说比起之前的计算机，这时的计算机更接近于实际，支持的功能更为丰富。

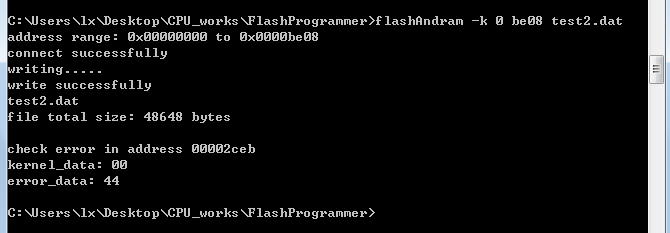
要调试这么一个复杂的CPU，不得不在开发的时候就做好调试的准备，这才有了单步调试的功能实现，用于监控CPU执行过程中的各个信号。除了这些，还是不够的，在移植ucore过程中，遇到很大困难，这么大一个操作系统，要找到故障发生地，定位在硬件里的错误，只靠单步调试，这是一个大海捞针的活。为此，使用mips-linux-gnu-objdump对ucore反汇编，在执行过程中记录一些标志性的寄存器值，估计出错的位置。同时将汇编代码对应到源文件中进行比对，加深对程序执行过程的理解。另外，也使用qemu和gdb对ucore执行进行模拟，用info registers得到模拟中的寄存器状态和教学机上的状态比对。在ucore中函数的出入口添加cprintf，在终端检验结果。应该说，偌大的工程，就是这样一点一点地debug出来的。

正是有了这样艰辛的调试过程，使我们对硬件的调试，操作系统的执行，程序的运行有了更加深刻的理解，不然也不会有之后编译器的迅速完成。

其次，在实验中遇到的问题也很多，比如串口的问题，困扰我们很久，甚至于最后展示我们也十分提心吊胆，于是带了两块板子来演示。以下是flashAndram烧写ucore时出现的错误：







可见，如果一些基本的程序没有保障好，要完成下一步工作，该是多困难的事情。

这次实验，有挑战，更多的也是学习，虽然ucore操作系统基本采用往届的程序，但是为了做好移植，我们也费了大量时间熟悉交叉编译器，gnu工具链，qemu虚拟机，gdb调试，make的使用，学习而得的知识、技巧为之后移植中的调试打下很好的基础。

最后，最为巨大的收获是对CPU、操作系统、编译器、汇编语言程序设计几门重要专业课有了整体而全面的理解，加深了系统的认识，从而为计算机系统的进一步学习打下很好的基础。