Unicore Simulator 结题报告

李春奇　彭焯
王kan　华连盛

PKU

December 30, 2010

Contents

1. 工程概述
   - 设计目标
2. 项目介绍
   - 项目视图
   - CPU 模块
   - 内存模块
   - 调试模块
3. 模拟器的特别设计
4. 模拟器验证
   - 函数级验证
   - 指令级验证

工程概述

设计目标

开题报告中所设定的设计目标:

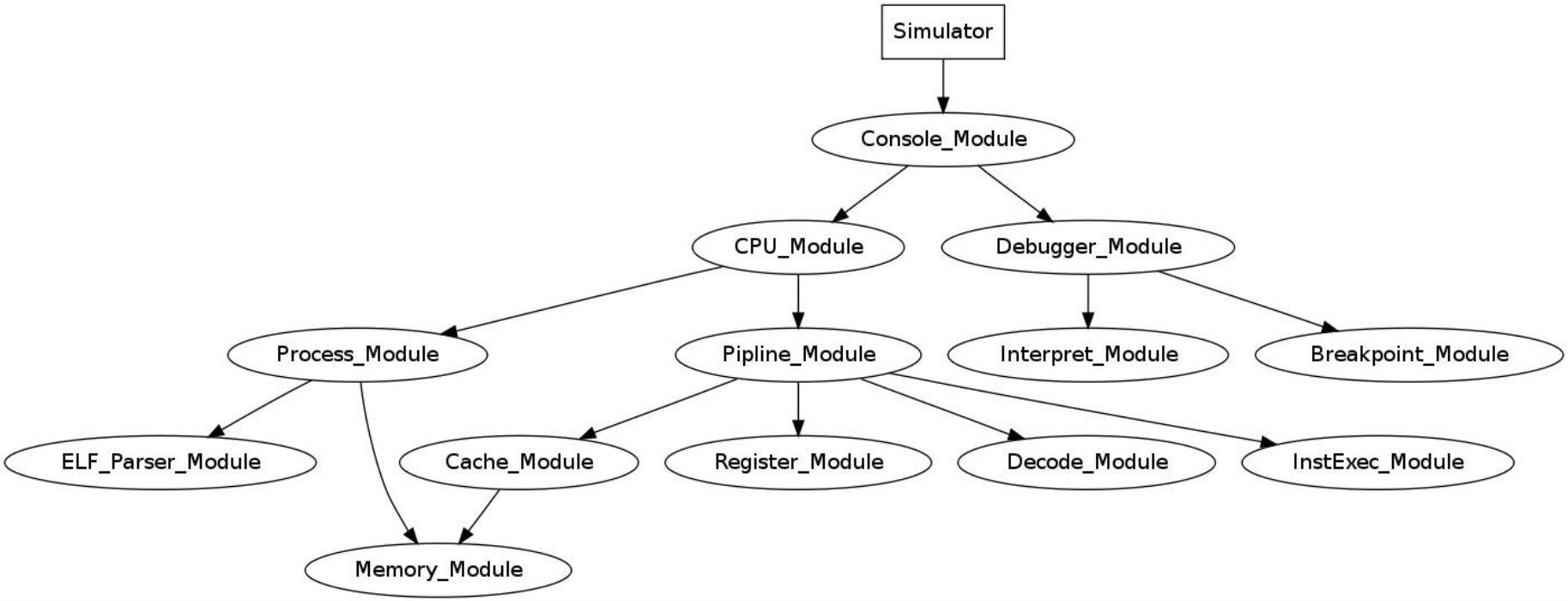
- 模拟五级流水的流水线CPU设计。
- 模拟至少一级cache，采用哈佛结构
- 模拟动态内存管理
- 一些库函数的模拟器级别的实现
- 调试工具的实现和CPU性能的统计

附加完成的功能:

- 反汇编模块的支持。
- 多种模式的支持（验证模式和自动运行模式）。

项目介绍

项目模块视图


Simulator
Console_Module
CPU_Module
Debugger_Module
Process_Module
Pipline_Module
Interpret_Module
Breakpoint_Module
ELF_Parser_Module
Cache_Module
Register_Module
Decode_Module
InstExec_Module
Memory_Module

CPU 模块

CPU模块是Simulator中最重要的模块，是整个Simulator的核心。CPU模块由两个子模块构成：进程模块和流水线模块。

- 进程模块负责ELF文件的解析、进程控制块（PCB）的初始化、进程内存空间的初始化。
- 流水线模块是CPU模块的核心，模拟了CPU的五级流水，实现了互锁和数据前递。

流水线

流水线的每一级说明如下：

1. IF-取指级：在I-cache取得指令并读入对应流水级的寄存器（具体实现为一个结构）。
2. ID-译码与访问寄存器级：对IF级取得的指令进行译码，获得指令类型、指令各域的值，存入对应流水级的寄存器中。
3. Ex-指令执行级：对ALU指令进行运算，跳转指令和访存指令进行计算跳转地址和存储地址。对ALU类指令做数据前递。
4. Mem-访存级：对访存类指令进行内存的存取。取内存指令在这一级实现前递，并标记需要互锁的寄存器。
5. WR-写回级：将目的寄存器写回。

流水线互锁

- 五级流水的互锁主要是Load指令和随后用到该指令目的寄存器的指令间会产生数据相关冒险。在模拟器的实现中，若遇到数据相关，则IF、ID级流水需要暂停一拍并在Ex级插入一个气泡。

下面是互锁在模拟器中的图示：

```
>s
Step 5:
Pipline #0 : IF  - addr:0x02000338  code:0x24eec004  sub r27, r29, #4 <> #0.
Pipline #1 : ID  - addr:0x02000334  code:0x70eec010  strw r27, [r29-, #16].
Pipline #2 : Ex  - addr:0x02000330  code:0x70ef400c  strw r29, [r29-, #12].
Pipline #3 : Mem - addr:0x0200032c  code:0x70ef8008  strw r30, [r29-, #8].
Pipline #4 : WR  - addr:0x02000328  code:0x70efc004  strw r31, [r29-, #4].

>s
Step 6:
Pipline #0 : IF  - addr:0x02000338  code:0x24eec004  sub r27, r29, #4 <> #0.
Pipline #1 : ID  - addr:0x02000334  code:0x70eec010  strw r27, [r29-, #16].
Pipline #2 : Ex  - Pipline Blocked Above.
Pipline #3 : Mem - addr:0x02000330  code:0x70ef400c  strw r29, [r29-, #12].
Pipline #4 : WR  - addr:0x0200032c  code:0x70ef8008  strw r30, [r29-, #8].
```

内存模块

Simulator实现了简单的内存管理，包括：

- 虚拟地址映射
- 栈空间的实现
- 内存访问越界检查
- 内存类型保护

内存模块-虚拟地址映射

- 进程的虚拟内存按照Segment管理，第0个Segment是栈空间，之后依次为ELF文件的可载入Segment。
- 对于任何访问内存的请求，都必须通过虚拟内存管理的中间层，经过验证地址是否合法、访问的数据是否可读（可写）才能够进行下一步的操作。
- 这样的设计能够保证任何ELF文件的错误都不会导致模拟器的崩溃。

断点设置模块

断点设置模块用来设置断点，断点地址为代码在内存中的虚拟地址。实现的功能有：

- 设置断点，并记录断点被命中的次数
- 删除断点
- 动态扩充断点集合，采用STL库中Vector的扩展方式。

截图如下：

```
love >>> ./simulator ../examples/fib -s                    10-12-30 0:09
MiniC Simulator begin.
File : ../examples/fib
CPU mode : normal
================================
>b 0x0200034c
Breakpoint added : 0x0200034c
>r
Breakpoint addr:0x0200034c  code:0x1a000004  mov r0, r4 << #0.
Hit 1 times.
>
```

反汇编模块

- 反汇编模块是在计划外完成的模块，主要目的是为了便于调试，避免每次调试前都需要进行objdump获得每个指令的汇编代码。
- 反汇编模块能够显示从指定行开始若干行的代码，也可以显示某函数中的代码。函数名到入口地址的查询在符号表中完成。

截图如下：

```
love >>> ./simulator ../examples/fib -s                    10-12-30 0:14
MiniC Simulator begin.
File : ../examples/fib
CPU mode : normal
================================
>l 0x02000344 5
0x02000344  :  0xbdffffd6       b.l 0x020002a0 <fib>
0x02000348  :  0x1a010000       mov r4, r0 << #0
0x0200034c  :  0x1a000004       mov r0, r4 << #0
0x02000350  :  0xbd000004       b.l 0x02000364 <print_int>
0x02000354  :  0x71df8004       ldrw r30, [r27-, #4]
>l f fib
<fib>
0x020002a0  :  0x70efc004       strw r31, [r29-, #4]
0x020002a4  :  0x70ef8008       strw r30, [r29-, #8]
0x020002a8  :  0x70ef400c       strw r29, [r29-, #12]
0x020002ac  :  0x70eec010       strw r27, [r29-, #16]
0x020002b0  :  0x24eec004       sub r27, r29, #4 <> #0
0x020002b4  :  0x24ef4014       sub r29, r29, #20 <> #0
0x020002b8  :  0x70d80010       strw r0, [r27-, #16]
0x020002bc  :  0x71d98010       ldrw r6, [r27-, #16]
0x020002c0  :  0x35300000       cmpsub.s r0, r6, #0 <> #0
0x020002c4  :  0xa0000002       beq 0x020002d0 <0x020002d0>
>
```

统计模块

统计模块主要分为两个部分：CPU信息的统计和指令数信息的统计。

- CPU信息的统计主要包括CPU总周期数、空闲周期数、流水线气泡数、CACHE命中情况和访存情况。
- 指令数信息统计主要包括总指令数和各个类型指令数及对应比例。

模拟器的特别设计

模拟器的设计过程中实现了一些计划外的独特的模块，这些模块都是用来使得模拟器的可用性更强。

- 反汇编模块的设计
- 断点模块的设计
- 内存模块的虚拟地址映射机制
- 对内存的修改和对寄存器的修改
- 对指令数及指令分类的统计
- 模拟器对三种模式的支持：普通模式、验证模式和自动模式

模拟器的三种模式

模拟器支持三种模式，分别用于不同的场合。

- 普通模式：用于普通的模拟，支持单步执行、显示流水线状态、CPU状态、寄存器堆状态等，在程序结束的时候显示统计信息。
- 验证模式：该模式用于指令级验证模拟器的正确性，该模式下不会显示上述状态，但是会显示每步的CMSR状态和程序结束的时候显示寄存器堆状态。
- 自动模式：该模式不会输出任何状态信息和统计信息，只会显示程序的输出，并且会在模拟器开始的时候自动运行。该模式经过重定位之后可以自动化得到程序执行的最终结果，用于验证编译器得到的结果是否正确。脚本batch_simulator和batch_simulator2就是用这个模式来实现的。

注：batch_simulator脚本用来验证编译器对于有正确答案的测试样例是否能够正确生成目标代码，batch_simulator2脚本用来验证编译器对于某文件夹下的所有.c文件生成的目标代码，全部编译执行并依次输出结果。

模拟器验证

模拟器验证

模拟器的验证主要分为三个方面：

1. 函数级验证：函数级验证主要负责验证Simulator的函数调用是否正确，着重验证栈和流水线的执行状态是否正确。重点验证递归函数的调用和主函数的返回、控制流是否正确（分支循环）。
2. 指令级验证：指令级验证主要负责验证Simulator实现的各条指令是否正确，指令验证分为三个部分：数据处理指令、分支跳转指令、访存指令，其中数据处理指令主要验证数据正确性和标志寄存器是否被正确写入，分支跳转指令主要验证跳转位置的正确性，访存指令主要验证存储器读写的正确性。

模拟器验证-函数级验证

函数级的验证主要验证了模拟器宏观的执行情况，主要的验证与验证代码如下：

1. 简单的main函数验证，验证代码：00simple.c
2. 函数调用验证，验证代码：01function.c
3. 递归函数调用验证，验证代码：02recursive_func.c
4. if分支语句验证，验证代码：03if.c
5. while循环语句验证，验证代码：04while.c
6. for循环语句验证，验证代码：05for.c

模拟器验证-指令级验证

指令级验证主要验证了模拟器在执行每条指令的时候是否有正确的行为，包括寄存器的写回、CMSR的设置、cache和内存的一致性等。主要验证代码如下：
数据处理指令的验证（D_Imm_shift、D_Reg_shift、D_Immediate）

1. opcodes域的验证，验证代码：opcode.s
2. shift域（前两类）和Rotate域（第三类）的验证，验证代码：shift.s
3. S域的验证（标志寄存器设置），验证代码：cmsr_logic.s, cmsr_arith.s
4. PC作为目的寄存器的验证，验证代码:pc_dst.s
5. 乘法和乘加指令的验证，验证代码：mul.s

模拟器验证-指令级验证（续）

分支跳转指令的验证（主要是条件转移指令的验证）

- 对cond域的验证和跳转地址正确性的验证，验证代码：30b_cond.c

访存指令的验证:
单数据传输指令的验证、半字和有符号字传输指令的验证

1. shift域的验证（L_S_R_Offset），验证代码：40mem_single_shift.s
2. 立即数访存的验证（L_S_I_Offset），验证代码：41mem_single_imm.s
3. U域、B域的验证，验证代码：42mem_single_U.s
4. S、H域的验证，验证代码：50mem_Hw_S_H.s

Simulator整体演示

Q&A