文章编号：1673－047X(2009)－02－028－04

一种 ARM指令集仿真器的实现与优化

徐怀亮, 刘晓升, 王宜怀, 朱巧明
(苏州大学计算机科学与技术学院, 江苏 苏州 215006)

摘 要：通过对比不同仿真策略的指令集仿真器的设计, 为便于调试, 实现了一种基于解释型策略的指令仿真系统, 并对传统的解释策略做了部分优化, 使得在功能正确的前提下, 速度比传统方式提高了近 10倍, 有效提高了嵌入式系统的软件仿真开发的速度。

关键词：指令集仿真；解释型仿真；嵌入式系统
中图分类号：TP311 文献标识码：A

0 引 言

随着嵌入式应用的普及和嵌入式产品更新换代速度的加快, 嵌入式系统开发的重要目标开始转向缩短开发周期和提高产品质量, 同时也就提高了对高效率开发工具的研究与应用的重视。因硬件设计完成的周期较长, 使得调试工作要等到硬件设计好之后才能进行, 延长了整个开发周期, 也就推动了软件仿真调试技术的发展。软件仿真的重点是微控制器的仿真, 而对微控制器的仿真重点又在指令仿真上, 因此本文给出一种基于解释型仿真策略的指令仿真器的实现, 并讨论了几种优化方法。

本文第一部介绍指令仿真, 并分析解释型指令仿真的过程; 第二部分仔细分析了指令仿真过程中的关键技术, 并结合实现过程给出了部分优化方法; 第三部分对本文工作进行了验证和总结。

1 指令仿真

指令集仿真器(Instruction Set Simulator, ISS)是用来在宿主机仿真另一种目标机上程序运行过程的软件工具。它通过仿真每条指令在目标处理器上的执行效果来仿真目标机程序, 是目标处理器的软件仿真器[1]。

在嵌入式软硬件的并行开发中, 指令仿真器是必不可少的工具之一, 在目标机可用之前, 通过它就可以完成软件的仿真调试, 真正做到了软硬件的并行开发。

当前, 指令仿真策略主要有两种[2]：一种是基于解释型的指令仿真策略, 它将应用程序存入仿真存储器中, 在仿真时完成取指、译码、运行等操作, 并将结果存入仿真寄存器或者存储器中; 另一种是基于编译型的指令仿真策略, 它先将应用程序反编译为高级语言文件(通常是 C语言), 然后再用 GCC优化为宿主机的汇编语言, 最后完成执行。

两类指令仿真器各有优缺点, 前者仿真速度比较慢, 但它可以很方便地实现对应用程序的控制。而编译型指令仿真器由于对源程序进行了反编译后又进行了优化, 已经丢失了原来的用户程序与高级语言的对应关系, 因而只能将程序一次执行完毕, 对它的执行过程不能提供如解释型指令仿真器那样详细的控制, 但它由于在运行时不需要逐条地解释指令, 因而具有很高的仿真速度。本设计是仿真调试平台的子模块, 所以采用解释型策略, 以方便完成调试工作。

收稿日期：2008－11－12
作者简介：徐怀亮(1984－), 男, 硕士研究生, 主要研究方向为嵌入式系统及应用。
基金项目：江苏省高校产业化基金项目(编号 JH07－082)；苏州市工业攻关项目。

传统的基于解释的指令集仿真策略是到目前为止应用最为广泛的一种指令集仿真器实现技术[3]。该仿真技术的主要优点是实现简单, 便于调试, 其缺点就是运行较慢。主要过程如下: 在内存中为其建立虚拟存储器, 存放将要执行的目标机汇编文件 (该文件为. ls类型, 下面将介绍为什么选择该类型的文件); CPU构件, 主要是寄存器部分; 然后将目标文件按照图 1所示的方式开始执行。

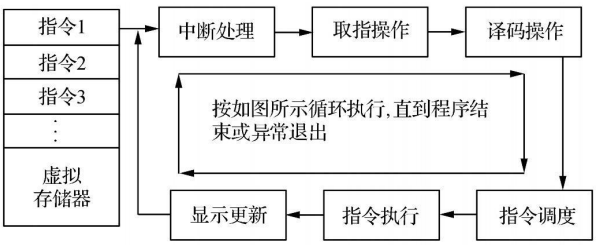

图 1　解释型指令集仿真流程图

系统在初始化时, 会屏蔽掉中断响应, 第一次执行将跳过中断处理, 在以后的每次执行过程中, 总要先检查是否有中断响应, 如果有, 进入相应的中断响应模块; 否则依次调用取指模块、译码模块、指令调度模块、指令执行模块, 完成指令的仿真运行。

2　指令仿真的关键技术

通过对. ls文件的分析, 能很好地完成源文件、反汇编文件和内存地址三者之间的对应, 以及变量地址的获取、函数入口的获取、pc值的计算等操作, 方便了指令仿真系统的实现与控制。

2 1　取指算法

目标机的源文件一般为 C语言或者汇编语言描述, 在 C语言文件中, 不包含仿真时所需的指令字或助记符, 而汇编源文件中涉及到变量的使用以及地址的跳转, 使得分析起来较复杂, pc值的更新也比较麻烦。因此二者都不适合作为仿真的分析对象。所以, 只能从源程序编译之后得到的文件中选取合适的文件进行分析。参考文献[4], 本文选择. ls文件作为仿真分析的对象。其结构如下:

```
00200390 <EndInitStack>:
EndInitStack():
  200390:   eafffffa    b    200380 <main>
00200394 <copydata>:
copydata():
  200394:   e2555004    subs   r5, r5, #4   ; 0x4
  200398:   e4936004    ldr    r6, [r3], #4
  20039c:   e4846004    str    r6, [r4], #4
  2003a0:   1afffffb    bne    200394 <copydata>
  2003a4:   e1a0f00e    mov    pc, lr
```

本文涉及到的. ls文件与文献[4]中的文件结构略有不同, 每一行一般为存储地址 +机器码 +指令。用户可以很清晰地从这个文件中看到程序代码的编译情况。采用. ls文件还有另一个好处就是寻址的问题, 在指令中, 尤其是跳转指令中, 使用偏移地址的情况居多, 这样就要时时计算 pc的值, 而. ls文件把相应的偏移地址都转换成了绝对地址, 方便了 pc值的计算。

ARM系列 CPU与其他系列有一些不同, 由于片上存储器比较大, 一般都移植了操作系统在 ARM目标板上, 所以仿真时只需找到“200390:　eafffffa　　b　　200380 〈main〉”命令行, 就可以获取仿真时指向 main入口的 pc值。

在设计虚拟存储器时, 采用一种类似. ls文件格式的存储格式, 按. ls文件的行来存放, 可以通过存储地址 (pc值)找到要执行的指令, 同时也可以方便地实现指令的跳转。从上述几方面我们可以得出 pc值的计算公式:

$$pc(next) = \begin{cases} pc(cur) + \Delta pc, & sequence\_do \\ addr\_return, & process\_return \\ addr\_jump, & jump\_instruction \end{cases}$$

其中：pc(next)为下一条指令的地址；pc(cur)为当前指令的地址；Δpc为 pc的偏移量，取决于工作状态，ARM状态一般为 4，Thumb状态一般为 2；sequence_do为顺序执行；addr_return为子过程返回的地址；addr_jump为跳转指令的跳转地址，因为采用.lst文件作为分析文件，所以该地址为绝对地址，可直接赋给 pc。

改变 pc值的还有中断返回的情况，在 ARM系列中一般通过对 pc进行操作来设置返回地址，如未定义指令异常的返回：movs pc, R14_und等。

确定 pc以及其改变方法之后，仿真程序就可以被顺序地执行，pc值会在仿真过程中自动更改，下一条将要执行的指令就是地址为 pc值的指令行，可以在虚拟存储器中得到。虚存结构如.lst文件，其查找方法如下：比如 pc(cur)值为 200 000H，当前行号为 10，pc(next)为 200 008H，可以直接算出下一条指令所在的行号为 10+(200 008H-200 000H)/4=780，因此可以直接读取第 780行作为下一条指令。

2.2 译码和调度算法

指令译码就是根据 ARM指令集编码，将要执行的指令译成操作码、操作数、条件码等指令仿真需要的信息。译码的重点在于获取不同寻址方式下的操作数，通过分析指令字助记符和指令码，可以很方便地得到不同寻址方式的操作数。

指令调度就是把指令字与完成相应操作的功能函数对应起来，使得仿真时取到该指令后能知道该调用哪个执行过程。传统的解释型仿真策略的主体是一个大的 switch结构，并使用大量的 case语句，这将严重影响仿真运行的速度。本文给出一种改进的结构，采用 hash结构，如表 1所示。

表 1 指令存储调度表

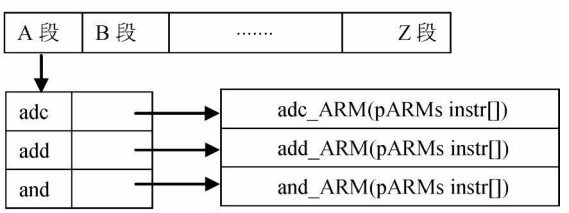

表 2 缓存表结构

指令助记符	对应执行指针
adc	adc_ARM(pARMs instr[])
add	add_ARM(pARMs instr[])
and	and_ARM(pARMs instr[])
……	……

根据表 1建立一个对应关系，指令调度时首先根据指令助记符的首字母查找在 hash表中的对应段，在段内可以采用折半查找，提高指令调度的速度。段内的数据分为两部分：指令助记符和对应的过程函数指针。指令调度时通过函数指针可以直接调用过程函数。

为了合理利用 20%和 80%的规律，即指令系统中有 20%的指令使用率高达 80%，而其他 80%的指令使用率为 20%，本文为常用的指令建立一个缓存表，存放常用的指令及对应执行函数的指针。解码时先从缓存表中查找，若找到对应的，根据表中执行函数指针直接运行仿真过程(此过程可以省去繁琐的译码过程)，否则，再从指令存储调度表中查找。其中缓存表的结构如表 2所示。

2.3 指令执行函数的设计

为了更好地提高扩展性和重用性，本设计把指令集分成三种类型：执行指令、跳转指令和寻址指令。执行函数对应着以指令值助记符命名的函数过程；跳转指令则是主要负责修改 pc值的函数过程；寻址指令就对应着不同寻址操作的函数过程。其中执行指令又是根据寻址方式再细分，同一类寻址方式的指令化为一类，当有指令增加时，只需要增加相应的执行过程就好了，其他的都无须改变。

在设计指令执行函数时，采用数据获取和数据操作相分离的方法，使得执行函数更具通用性，当新的指令系统添加时，只需要添加相应的寻址方式和原系统中没有的指令执行过程，就能完成添加指令集的操作，提高了系统设计的可扩展性。

3 指令仿真系统的实现及实验结果分析

该指令仿真系统采用面向对象技术，用.net为平台开发。主要包括 CPU构件的设计、寄存器构件的设计、存储器构件的设计等，其中存储器构件采用动态数组实现，可以根据用户需求分配相应的存储空间，同时提供了查看变量信息的图形接口，并实现了源文件与反汇编文件的对应，其仿真运行结果如图 2所示。

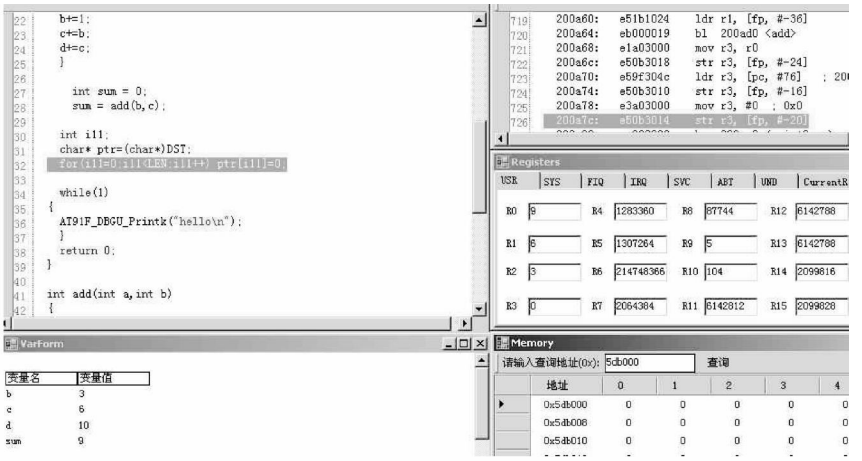

图 **2**　仿真运行图

笔者对采用传统方法实现的仿真系统和本文实现的仿真系统仿真效率进行了对比实验, 结果如表 3所示。

通过对比可以看出, 本文采用的技术在仿真速度方面比传统的设计方法有了 10倍左右的提高。不足之处在于需要一些额外的空间来完成运行, 是一种典型的以空间换时间的算法。随着 PC机内存价格不断下降, 与速度要求相比, 空间问题不再那么重要。所以本文实现的系统在一些对时间要求不是很苛刻的仿真中, 还是较好地完成了指令的仿真。

表 **3**　实验对比结果

测试例子	指令条数	传统方法	本文方法	提高倍数
冒泡排序	6 534	909 534 s	85 645 s	10. 53
小灯程序	856	121 274 s	12 289 s	9. 87
响铃程序	2 583	396 943 s	35. 17 s	11. 28

4 总　结

本文讨论的指令集仿真器是一种基于解释型的仿真器, 提供了友好的操作, 改进了传统的译码调度策略, 可以用在对时间要求较低的面向 ARM处理器的嵌入式软件集成开发环境中。本文分别对指令的获取、解析、调度、执行进行控制, 力求正确有效地实现指令系统的仿真。实验结果表明, 本文提出的方案能够较好地完成指令系统的执行, 仿真平台中用户程序的运行接近于真实平台, 为其他指令系统仿真的设计提供了参考。当然本文设计的仿真器只是考虑了一种指令集的情况, 虽给出了扩展性的说明, 但还是存在很多不足之处, 下一步工作是在本文的基础上设计一种高效的多指令集仿真器。

参 考 文 献

[1] 刑文峰. 一种高速灵活的指令仿真器[J]. 计算机工程, 2004, 30(22) : 74－75.
[2] 谢耀滨. 虚拟指令集的构建及翻译技术研究[J]. 计算机工程与设计, 2007, 28(14) : 3 489－3 491.
[3] 陶峰峰. DSP指令集仿真器的设计与实现[J]. 计算机仿真, 2005, 22(9) : 225－228.
[4] 廖桂华. 单片机指令仿真研究与实现[J]. 军民两用技术与产品, 2008, 45(4) : 45－48

A Design and Optimization of ARM Instruction Set Simulator

Xu Huailiang, Liu Xiaosheng, Wang Yihuai, Zhu Qiaoming
(School of Computer Science and Technology, Suzhou University, Suzhou 215006, China)

Abstract: Through the cooperation of instruction set simulator based on different method, in order to familiar with debugging, we design a instruction set simulator based on interpretation, and add some optimization to the traditional method. Under the condition that the function is correct, we improve the simulator speed by almost 10 times, and speed up the soft development of embedded system.

Key words: instruction set simulator; interpretative simulation; embedded system