第 33 卷 第 4 期 计 算 机 工 程 2007 年 2 月
Vol.33 No.4 Computer Engineering February 2007
·软件技术与数据库· 文章编号：1000—3428(2007)04—0055—03 文献标识码：A 中图分类号：TP314

基于 MIPSX 的模拟器研究与移植

高轶杰，郑扣根，冯骁斌
(浙江大学计算机科学与技术学院 CAD＆CG 国家重点实验室，杭州 310027)

摘 要：在开发基于 MIPSX 的嵌入式系统 IDE 的背景下，介绍了目标系统 MIPSX，分析了一个界面友好、功能强大的 MIPS 模拟器 SPIM 的代码和工作流程，在此基础上阐述了移植 SPIM 以支持 MIPSX 的流程。探讨了文法、指令模拟和可执行文件读取等几个重要环节。
关键词：SPIM；MIPSX；模拟器；移植

Research and Porting of MIPSX Based Simulator

GAO Yijie, ZHENG Kougen, FENG Xiaobin
(State Key Laboratory for CAD & CG, College of Computer Science, Zhejiang University, Hangzhou 310027)

【Abstract】 Under the background of developing a MIPSX based embedded system IDE, the paper introduces the target system MIPSX, and then analyzes the source codes and work flow of a user friendly, powerful MIPS simulator——SPIM. It illustrates the process of porting SPIM to support MIPSX, especially three key steps, namely syntax analyzing, instruction simulating and binary file reading.
【Key words】 SPIM; MIPSX; Simulator; Porting

当前，嵌入式系统应用是计算机领域的一个热点，越来越多的人正在从事与之有关的研究开发工作。在开发过程中，模拟器起着至关重要的作用，它可以极大改善开发环境，用户可以略过传统嵌入式系统应用调试过程中频繁的烧录工作，即可在 PC 机上完成大部分软件调试工作，因此节约了时间，提高了开发效率。

基于这样的需求，目前，大部分嵌入式软件集成开发环境（IDE）都提供了相应目标平台处理器的指令集模拟器。此外还有各种针对不同用途的模拟器，比如模拟PC的VMware、Virtual PC，各种游戏机模拟器，针对ARM的Sky EYE等。我们在IDE的开发过程中，也考虑提供基于MIPSX芯片的模拟器，虽然可以从头设计开发一个全新的模拟器，但是这往往需要很长的周期，因此移植一个能满足我们需求的优良模拟器不失为为一个快速有效的方法。本文论述了移植基于MIPS[1]芯片的模拟器SPIM[2]以支持MIPSX的过程。

1 目标系统

MIPS系列CPU是当今嵌入式产品应用广泛的一种RISC微处理器，它来源于斯坦福大学的 MIPS 计划，不采用伯克利分校 RISC 窗口技术而采用消除流水线各级互锁的微处理器 MIPS(Microprocessor Without Interlocking Pipeline Stage)技术。MIPS 系列当前有 R2000、R3000、R4000、R8000、R10000等多种产品。

MIPSX 是 MIPS 系列里的一种处理器，它是一个 32 位 RISC 处理器，有 5 级流水线，总共有从编号 0 到 31 的 32 个通用寄存器。所有寄存器为 32 位。此外，MIPSX 处理器还提供若干特殊寄存器： PC，PC1，PC2，PC3，PC4，PSW，MD。

MIPSX[3]处理器的位模式采用大端结尾，提供以字节为单位的统一寻址空间，存储空间从 0 到 2^{32}-1，但是读写以字为基本单位。MIPSX的地址空间可进一步划分成系统地址空间和用户地址空间，当一个内存访问的地址的高位（bit 0）被置 1 时，则是访问用户地址空间；否则就是访问系统地址空间。

相对于其他MIPS系列处理器，MIPSX与MIPS200 最为接近，但是还有许多特殊之处。除以上的体系结构外，还有：指令集[4]的不同，比如MIPSX没有浮点操作；MIPSX指令的基本操作码只占 5 位；MIPSX在跳转指令中的延时槽有两条等。

2 SPIM 分析

根据前面对于 MIPSX 的分析，移植针对 MIPS 系列芯片的模拟器是一个较为合适的选择。目前有许多的 MIPS 芯片的模拟器：比如 Vmips、Qemu、mips64emul、SPIM 等。相对于其他模拟器，SPIM 由于其开源、易用性成为了我们的选择。

2.1 SPIM 简介

SPIM 是一种用来模拟执行运行在 MIPS 2000/3000 芯片上程序的模拟器，可以在多平台下运行。虽然目前最新的 7.1 版本可以支持 MIPS32 体系，但是对于我们来说完全支持 MIPS-I 命令集的 6.3 版本的 SPIM 已经足够，而且从功能来说差别不大。从字面可以看出，SPIM 就是 MIPS 的反写。SPIM 可以读入执行 MIPS 汇编代码（在有些平台上可以执行 MIPS 的二进制代码）。SPIM 是一个运行 MIPS 指令的独立的系统，提供了友好的操作界面，同时包含了一个调试器和一些常用的由操作系统提供的功能。

2.2 SPIM 代码分析

SPIM是MIPS的一个解释器[5]，它读入汇编文件后，在内存中建立一些代表MIPS处理器的状态的数据结构，包括寄存器、状态字、堆栈等内容，根据MIPS指令的定义以解释的方

作者简介：高轶杰(1980－)，男，硕士生，主研方向：嵌入式操作系统；郑扣根，教授；冯骁斌，硕士生
收稿日期：2006-03-24 **E-mail：**gaoyijie@gmail.com

式模拟执行指令，同时相应修改处理器的状态，从而完成处理器的模拟。

图 1 表示了 SPIM 源代码目录中几个核心的文件之间的关系，这些文件大部分都是和具体目标系统相关的，对于我们理解 SPIM 的工作流程和移植的开展有很重要的作用。其中 spim.c 和 spim-utils.c 包含 SPIM 最外层的函数，负责与用户进行交互和整个程序流程的控制。在 parse.y 和 scanner.l 中则定义了词法和语法的描述，负责对输入的汇编文件进行语法和词法分析，并将程序用内部形式存储组织起来。而 run.c 则负责对具体 MIPS 指令模拟执行。其余的几个文件定义了处理内部表示的 MIPS 程序时需要用到的工具程序。

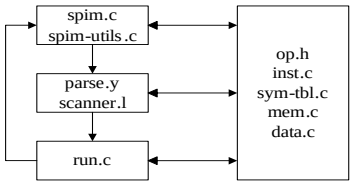

图 1 SPIM 的核心文件

根据图 2，简要总结一下 SPIM 执行流程。首先，SPIM 会调用 Initialize_world 函数来初始化整个模拟器，它分别初始化了内存结构、寄存器表和符号表。接着通过调用 read_assembly_file 函数读入汇编文件，通过词法和语法分析，SPIM 将合法的指令存储到模拟器的指令段中，同时解析符号、建立符号表等。之后就可以根据设定的起始地址、断点设置、运行步数，通过调用 run_spim 函数来模拟运行程序。在整个过程中，可以查看当前模拟器的寄存器、堆栈、符号表等信息。

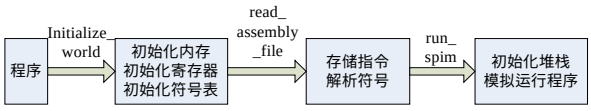

图 2 SPIM 流程图

3 SPIM 的移植

我们的目标是完成 SMIP 对 MIPSX 的支持，能够读入 MIPSX 的汇编代码，或者是 MIPSX 的可执行程序，并对指令的执行进行模拟。根据前面的分析，将工作分为两部分，即 MIPSX 指令集的移植和 MIPSX 可执行文件的读取。

3.1 MIPSX 指令集移植

这部分关键部分是指令的内部表示、读取汇编文件的文法分析以及指令的模拟。

3.1.1 指令结构

首要问题是 MIPSX 指令的数据结构的表示，由于 MIPSX 和 MIPS R2000 的差异，因此我们重新定义数据结构如下：

```
typedef struct inst_s{
        short opcode;          /*Op.h 中定义的操作数*/
        unsigned char rs1;/*源寄存器  Src1*/
        unsigned char rs2;/*源寄存器  Src2*/
        unsigned char rd; /*目标寄存器  Dest*/
        unsigned char sham;/*位移量*/
        int     imm;                /*指令中偏移等立即数  */
        imm_expr *expr;     /*立即数计算表达式*/
        uint32 encoding;   /*编码后的二进制指令*/
        char *source_line;/*源程序中的行号*/
} instruction;
```

比如对于一条读取内存指令：ld 0[r3], r2，这个指令结构相应的域 opcode=YY_LD_OP, rs1=r3, rd=r2 以及 imm=0。

3.1.2 文法分析

通过修改 Op.h、inst.c、sym-tbl.c 等文件，完成了对 MIPSX 指令集的支持，同时在文件 parser.y 中，重新定义了完整的文法结构以及相应的处理过程，下面列出了部分内容：

```
LBL_CMD:    OPT_LBL CMD    |        CMD ;
OPT_LBL: ID ':' {
            record_label((char*)$1.p,text_dir?   current_text_pc()   :
current_data_pc(), 0);
 }
 | ID '=' Y_INT{
        record_label ((char*)$1.p, (mem_addr)$3.i, 1);
 };
CMD:    ASM_CODE TERM | ASM_DIRECTIVE TERM | TERM;
TERM: Y_NL | Y_EOF      ;
EXPRESSION:        EXPR
EXPR:  Y_INT | ID
EXPR_LST:   EXPR_LST  EXPRESSION      |
  EXPRESSION      | EXPRESSION    ':'        Y_INT
ID:    Y_ID
ASM_CODE:    Y_ADD_OP SRC1        SRC2   DEST{
            store_inst($1.i,$2.i,$3.i,$4.i,0,NULL);
          }
        | Y_ADDI_OP        SRC1    SRC2    DEST{
store_inst($1.i,$2.i,$3.i,$4.i,0,NULL);
          }
…;
```

其中的核心部分是对于 OPTL_LBL 和 ASM_CODE 的处理。当词法分析器匹配到 ASM_CODE 时，将会调用一个函数 store_inst 来将当前的汇编指令转变成 SPIM 中的指令结构，并存储到 SPIM 中模拟器代码段中的下一个位置。

如果碰到的是 OPT_LBL，则当前为标签，因此调用函数 record_label。如果这个标签是新定义的，则创建一个新的 label 结构，并且加入到符号表中，同时 SPIM 将会对所有引用该标签的指令进行解析。

3.1.3 指令模拟

完成词法分析将程序读入到模拟器内部后，就可以根据模拟器的断点、程序起始位置等设置，调用 run_program 函数模拟运行程序了。在函数 run_spim 中，使用 REDMEM_INST 宏从内存中读取下一条要模拟执行的指令。通过解析该条指令结构中的 opcode 域，我们分别执行相应的指令模拟。

由于指令结构的更改，因此我们重写了 run.c 以实现对 MIPSX 指令的模拟运行。其中大部分的指令是比较简单直观的，比如对于 op 域为 Y_ADD_OP 的一条指令 inst，我们只需要进行下列的操作：R[RD(inst)]=R[RS1(inst)]+R[RS2(inst)]，其中 R[]代表由数组形式表示的寄存器，RS1 是解析指令 inst 中的各个寄存器号的宏。然而对于某些指令，比如 op 域为 Y_JSPCI_OP 指令的话情况就会比较复杂。由于 jspci 指令有两个延时槽，我们应该先将延时槽中的指令执行完，然后再跳转到目标位置继续执行指令。

至此移植完成的 SPIM 已经实现了对 MIPSX 指令集的支持和模拟运行，通过早前移植的 GCC 中的编译器 cc1，我们将基于 MIPSX 工程的 c 文件编译成汇编文件进行了测试，实验证明 SPIM 的移植是比较成功的。

3.2 可执行文件读取

虽然移植完成的 SPIM 已经实现了对 MIPSX 指令集的支持，但是仅能读取汇编文件对于实用的模拟器来说，这是远远不够的。为此，我们需要继续改进模拟器，提供直接读取

可执行文件进行模拟运行的功能。

MIPSX 上可执行文件的类型是 a.out 文件格式的一种，但是与一般的 aout 的文件格式稍有差别，如图 3 所示，它包括文件头、代码段、数据段、代码段以及数据段的位置变换信息、符号表以及字符表。

Object file header	Text segment	Data segment	Relocation information	Symbol table	Debugging information

图 3 MIPSX a.out 文件格式

其中关键信息是它的文件头，这是一个 exec 结构，具体信息如下：

```
struct exec {
unsigned short   a_info; //魔力数
unsigned long    a_text; //代码段大小（以字节计算）
unsigned long    a_data; //数据段大小
unsigned long    a_bss; //未初始数据段大小
unsigned long    a_syms; //符号表大小
unsigned long    a_entry; //程序执行入口
unsigned long    a_trsize; //代码段重定位信息大小
unsigned long    a_drsize; //数据段重定位信息大小
};
```

根据对 exec 结构的分析，一旦读取了文件头，我们就可以根据 exec 结构的信息来分别加载 a.out 文件各部分的内容，并进行与词法分析类似的转换过程了。因此，我们添加了 read_x3_aout_file 函数来处理可执行文件，流程具体可以分为以下几个部分：

(1)读入可执行文件的文件头，即将 exec 结构的数据读入，由于 MIPSX 是大端结尾，而 PC 是小端结尾，因此需要对读入的数据进行字节转换，然后分别读取每个域的内容。

(2)根据文件头提供的代码段大小信息，循环读入每一条指令，初始化模拟器的代码段。由于在 MIPSX 的体系结构中指令都是 4 个字节的，因此，每次读入 4 个字节的数据。在读入每一条指令后，首先会对读入的数据进行字节转换，然后通过 inst_decode_from_int 函数将读入的指令转变为模拟器中内部的 MIPSX 指令结构，然后将该指令结构储存到模拟器的代码段中的下一个位置。

(3)根据文件头提供的数据段大小信息，初始化模拟器的数据段。在这一过程中如同读入代码段的信息一样，每次读入数据段的 4 个字节，并进行相应的字节转码，并存储在模拟器数据段中的下一个位置。

(4)根据文件头提供的代码段以及数据段的重定位信息的长度，忽略这一段的信息，因为它对于可执行文件的模拟运行并没有什么具体的用处。

(5)读入可执行文件的符号表中的每个符号。根据文件头提供的符号表的大小信息，将整个符号表读入，进行相应的字节转码，并获得符号表的首个符号指针，供稍后建立模拟器自身的符号结构使用。

(6)读入字符表。由于字符表的开始 4 个字节用以标识字符表的大小，因此在获得字符表的大小信息之后，便开始将字符表的数据逐个读入，然后根据符号表每个符号的名字，建立模拟器内部的符号表结构。

当将可执行文件读入模拟器之后，就可以按照前面描述的过程对可执行文件进行模拟运行了，这与通过读取汇编文件进行模拟相比没有任何差别。

4 结束语

虽然 SPIM 作为一个模拟器界面友好，而且结构清晰容易移植，但是还是存在一些不尽如人意的地方，比如目前只是对指令集的支持比较完善，此外还有各方面需要进一步改进，如各种系统调用、I/O 等操作还需要加强，这些也是下一步的目标。

参考文献

1 夏仁霖. RISC 技术参考大全[M]. 北京: 电子工业出版社, 1992.

2 Larus J R. Assemblers, Linkers, and the SPIM Simulator[M]. Morgam Kaufmann, 1997.

3 Leadtek Research Inc.. MIPS-X3 Programmer’s Manual[Z]. 2001.

4 Chow P. MIPS-X Instruction Set and Programmer's Manual[R]. CA, Stanford: Stanford University, Technical Report: 86-289, 1986.

5 严迎建. ARM4 指令集模拟器设计及优化技术[J]. 小型微型计算机系统, 2005, 26(2): 315-317.

(上接第 54 页)

识符较粗时，这些网页所共有的分块可能未被识别；或当分块粒度较细时，这些分块又已经被分割开，不能作为聚类的特征。所以，算法的精度在分块粒度较粗和较细时会具有较高值。

(2) 由于 HTML 中的分块标识符是通用的，如 <table>, <hr>,
等，因此通过分块标识得到的分块是网页的一个普遍特征，它只与网页的排版格式有关，而与其所采用的语言和网页内容无关，也与站点的超链结构和网页在站点所处的位置无关，不会因为站点规模的变化而变化。

网页分块是网页中存在的普遍特征，用分块进行网页聚类能够保证算法具有较好的稳定性和适应性。

4 结束语

基于 Web 逻辑域模型的 Web 搜索模型和基于 DOM 文档的网页分块技术都是当前 Web 挖掘领域的热点，Web 逻辑域的构建需要以不同的粒度和视角对网页和 Web 超链接进行建模分析，而不同的网页分块技术则为这些模型的实现提供了一个很广阔的技术空间。本文通过对网页分块进行聚类来挖掘 Web 站点中的逻辑域，取得了较好的实验效果。所以，如何把二者有机地结合起来运用到 Web 搜索中，将是一个很有前景的研究方向。

参考文献

1 Li Wensyan, Kolak O, Wu Quoc, et al. Defining Logical Domains in a Web Site[C]//Proceedings of the 11th ACM Conference on Hypertext. San Antonio. 2000-05: 123-132.

2 Eiron N, McCurley K S. Untangling Compound Documents on the Web[C]//Proceedings of the the 14th ACM Conference on Hypertext and Hypermedia. 2003: 85-94.

3 Bar-Yossef Z, Rajagopalan S. Template Detection via Data Mining and Its Application[C]//Proceedings of the 11th International Conference on World Wide Web. ACM Press, 2002: 580-591.

4 Lin Shianhua, Ho Janming. Discovering Informative Content Blocks from Web Documents[C]//Proceedings of ACM SIGKDD'02. 2002.

5 Liu Z, Ng W K, Lim E P. An Automated Algorithm for Extracting Website Skeleton[C]//Proceedings of the 9th International Conference on Database Systems for Advanced Applications. 2004-03-17.

6 Deng Cai, Yu Shipeng. Extracting Content Structure for Web Pages

Based on Visual Representation[C]//Proc. of the 5th Asia Pacific Web Conference. 2003.