

文章编号:1001-9081(2005)02-0367-03

一种基于 ADL 的编译器自动生成方法研究

任小西,李仁发,张克环

(湖南大学 计算机与通信学院,湖南 长沙 410082)

(happyrrx@163.com)

摘 要:编译器是嵌入式系统软件中的重要组成部分,它对嵌入式系统的软件开发有重要影响。本文在将体系结构描述语言(ADL)与传统可移植编译器相结合,自动生成嵌入式系统编译器的思想基础上,对自动生成工具 genmd 的结构进行了分析。重点对其指令识别和机器描述生成部分进行了抽象和建模。同时,针对 genmd 不支持分支跳转类指令的问题提出了改进方案。

关键词:编译器;体系结构描述语言;自动生成;genmd;嵌入式系统

中图分类号: TP314 **文献标识码:** A

Study on ADL-Based automatic generation of compiler in embedded system

REN Xiao-xi, LI Ren-fa, ZHANG Ke-huan

(College of Computer and Communication, Hunan University, Changsha Hunan 410082, China)

Abstract: Compiler is an important part of embedded system software, it has great impact on software development of embedded system. This paper analyzed the structure of an compiler automatic generation tool genmd, based on the idea of combining ADL with retargetable compilers, and put emphasis on abstracting and modeling of instruction recognition and machine description generation. Meanwhile, a method was provided and implemented to solve the limitation of genmd that was unable to support instructions of branch and jump.

Key words: compiler; ADL(Architecture Description Language); automatic generation; genmd; embedded system

0 引言

近年来半导体和计算机技术飞速发展,极大地降低了各种计算系统的价格,同时功能也不断增强。在这样的背景下,嵌入式系统得到了日益广泛的应用。嵌入式系统的特点是:资源有限、功能灵活、生命周期短、更新速度快。这些特点决定了嵌入式系统的开发工具也要不断适应这种变化^[1]。编译器是嵌入式系统开发工具的重要组成部分,它允许开发人员使用高级语言替代汇编语言,从而加快系统开发进度,缩短研发周期,争取市场的主动权。然而,重新设计一个新的嵌入式系统的编译器需要很长的开发时间。因此,如何快速地为新系统提供编译器,这对编译器的设计提出了一个新的挑战^[2]。

目前,为嵌入式系统提供编译器主要有两种方法。一种是移植现有成熟的编译器,另外一种是从目标系统的体系结构描述脚本中抽取机器信息,然后自动生成一个编译器^[3]。这两种方法各有优缺点。在第一种方法中,由于被移植的编译器十分成熟,因此编译速度快,目标代码质量高,但是移植过程却需要相当大的工作量。与此相反,第二种方法只要求开发人员使用体系结构描述语言(ADL)对目标系统进行描述,并通过编译器自动生成器就可以得到一个专门针对新目标系统的编译器。因为 ADL 是一种高级语言,所以用它来描述目标系统的工作量相对较小。但是,这种方法的缺点是代码优化可能没有成熟的传统编译器做得好。由此可见,这两

种方法的优劣正好互补,如果能把它们结合起来,首先从目标系统的 ADL 描述文本中自动抽取移植一个传统编译器所必须的信息,然后将这些信息与传统编译器结合起来,就可以快速得到一个新的针对目标系统的高质量编译器。图 1 以 GCC 和 Sim-nML 为例,表示了这种方法的工作流程。

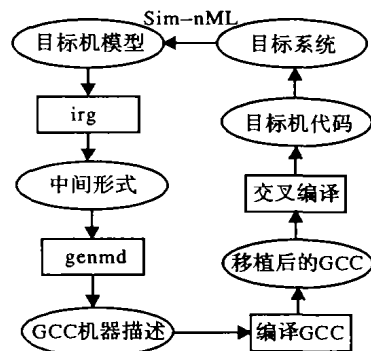


图 1 GCC 结合 ADL 方法的工作流程

Sim-nML 是印度理工学院 Kanpur 分校计算机科学与工程系开发的一种 ADL 语言^[4]。在此基础上相应地开发了工具集 genmd 和 irg。其中 irg 是中间信息转换工具^[5]。genmd 是 GCC 的机器描述生成工具,它使用 Sim-nML 作为 ADL 语言来描述目标系统的体系结构,然后从中抽取机器信息,最后生成移植 GCC 所需要的三个目标机器描述文件,从而实现 GCC 到目标系统的移植。尽管 genmd 可以从网上获得,并且可以

收稿日期:2004-07-28;修订日期:2004-10-19 基金项目:湖南省科技厅制造业信息化示范工程项目(HNMIE-A-026)

作者简介:任小西(1978-),女,湖南长沙人,硕士研究生,主要研究方向:嵌入式计算、编译器;李仁发(1956-),男,湖南郴州人,教授,主要研究方向:分布式计算与网络、虚拟与仿真、嵌入式计算;张克环(1978-),男,湖南常宁人,硕士研究生,主要研究方向:SoC 及其体系结构。

在 Linux 平台下编译通过,但是这并不意味着 genmd 已经十分完善。实际上,genmd 的功能非常有限,有许多机器特征无法描述,比如流水线、分支跳转指令等。而且其作者测试采用的是 Intel 公司的 8 位微处理器 8085 这样简单的微处理器,足见其仍有许多值得改进的地方。尽管如此,genmd 对于编译器的自动生成仍具有较大的参考价值,可为今后的进一步研究奠定基础,从而更好地实现编译器自动生成。

1 genmd 的基本结构及工作原理^[7]

genmd 工作流程如图 2 所示。首先使用 Sim-nML 语言描述目标系统的体系结构,包括指令集、寻址方式、存储器以及其他硬件资源等,然后由中间代码生成器 irg 将 Sim-nML 描述转换为中间代码形式(IR)。IR 以二进制数据表的形式保存 Sim-nML 文本格式包含的全部信息,从而使得后面的分析程序能方便地获取这些信息^[5]。接着 genmd 进行模式分析。在 Sim-nML 中,寻址方式以及指令集都是通过 and 模式和 or 模式按照一定的层次关系组织成树型结构。例如对指令集的描述是从 instruction 开始,由若干类指令组成。这些指令之间是平行关系。某类指令的定义由具体的指令以 or 模式构成。然而某条指令的参数之间则是以 and 模式构成,因为它们有严格的先后关系。模式分析的结果是得到模式表(mode_table),该表是 genmd 中最重要的数据结构,它包含一些目标机器最本质的信息。得到 mode_table 之后的工作是进行指令收集和变换。前者从指令树的根节点开始遍历整个树,并用收集到的信息构造一个指令表(instr_table)。后者针对指令表示的多样性,通过一定的等价变化统一指令的表示形式,从而简化后面的指令识别过程。

最后 genmd 将进行指令识别和机器描述生成。指令识别过程将前面收集的指令与预先提供的指令模板进行匹配,如果匹配成功,则返回所属指令类的标识符,否则不做任何操作,仅输出错误信息。机器描述生成过程根据识别过程所返回的标识符调用对应的生成函数,产生该类指令的机器描述。同时,该过程还将产生一些全局信息,例如寄存器的使用、约束字定义以及各种预定义宏。

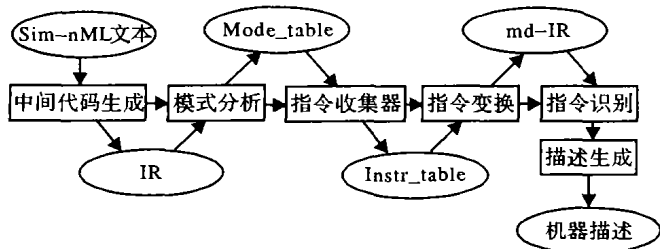


图2 genmd 工作流程

为了更系统和透彻地理解 genmd 的设计思想和方法,本文对它的源码进行了抽象,并利用自动机的理论对识别过程进行了建模。下面将详细介绍模型的建立和指令识别过程,其中以加法指令、减法指令、逻辑与指令和数据传送指令为例。首先,在 genmd 建立指令识别自动机之前,必须以波兰式的形式给出上述四类指令的模板,如下所示:

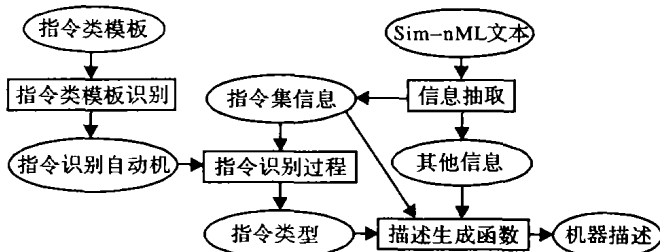


图3 genmd 指令识别结构

加法: = \$0 + \$1 \$2

减法: = \$0 - \$1 \$2

逻辑与: = \$0 & \$1 \$2

数据传送: = \$0 \$1

genmd 会逐个扫描指令模板,构造如图 4 所示的指令识别自动机:

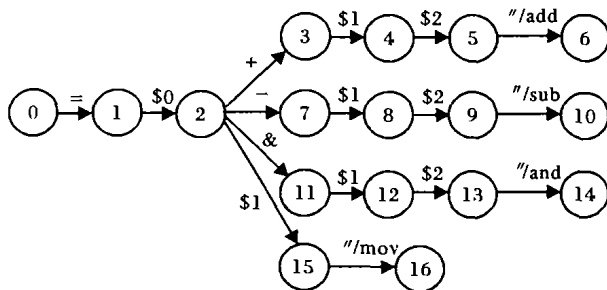


图4 指令识别自动机

其中状态 0 是初始状态。6、10、14 和 16 分别是自动机的终结状态,同时也是加法、减法、逻辑与和数据传送指令的识别状态,当到达这些识别状态时,自动机将分别输出指令对应的类型标识符。自动机的输入是从 Sim-nML 描述文本中抽取到的具体指令信息。例如,对于寄存器之间的加法运算指令,其 sim-nML 描述为:

```
op add( dest: REG, src: REG)
syntax = format( "add %s, $ s", dest. syntax, src. syntax)
action = {
    dest = dest + src;
}
```

其中,REG 表示 dest 和 src 的寻址方式为寄存器寻址,syntax 表示该指令的汇编码,action 表示了该指令的语义。genmd 从该描述中抽取了运算符以及操作数的有关信息,然后进入自动机的初始状态 0,开始识别过程。注意到此时 action 部分的语义信息已被转换为波兰式,因此自动机的输入将依次为: ' ', 'dest', '+', 'dest', 'src'。尽管这在形式上与自动机状态之间的转移条件不一致,但本质上 dest、src 所蕴涵的信息就是 \$0 和 \$1。这种蕴涵关系的转换通过一个信息查找函数实现。该函数能够在决定状态转移时,将 action 中的标识符转换为自动机所需要的形式,从而实现指令的正确识别。当 ' ', 'dest', '+', 'dest', 'src' 被依次输入到指令识别自动机以后,自动机会经过如下的状态转移: 0→1→2→3→

2 genmd 的指令识别和机器描述生成部分建模

指令的识别是在指令生成机器描述之前,识别出每条指令各属于哪一类指令,然后调用合适的生成函数,得到正确的机器描述。识别过程分为两个步骤:首先是根据预先设定的指令类模板生成指令识别自动机,然后把从 Sim-nML 描述中获取的目标机器指令集信息与指令识别自动机结合,判断每条指令的类型。最后,通过调用每种类型对应的描述生成函数,利用已经得到的指令集和其他机器信息,生成该类指令的机器描述文件,如图 3 所示。

4→5,并停留在状态5,等待下一个输入。虽然此时ADD指令语义要素已经全部满足,但是还不能输出ADD指令识别标识符。这是因为,如果此时就返回ADD指令识别标识符,则对于一个错误的输入序列:'=', 'dest', '+', 'dest', 'src', '-', 'dest',自动机也会把它识别为ADD指令。为了避免该情况的出现,自动机仅在状态6输出ADD指令识别标识符,并规定状态5到状态6的转移条件是外部输入一个空字符,则当前输入序列结束。

3 模型改进方法及其实现

3.1 模型的改进

建立和分析上述模型之后,不难发现genmd不支持分支跳转类指令的根本原因。由于根据指令模板所生成的指令识别自动机不能分辨数据传送与分支跳转之间的差别,从而导致了把分支跳转类指令当作普通数据传送指令来处理。指令模板反映的是数据处理的流动情况,尽管从这个角度看,分支跳转类指令与普通数据传送类指令是一致的,都是把数据从一个存储单元传送到另外一个存储单元,但是实际上两者有本质区别。此区别在于分支跳转类指令的目的存储单元是一个特殊的寄存器(程序计数器PC),而普通数据传送指令的目的存储单元是通用寄存器或者内存单元。这种区别映射到GCC的机器描述文件里面,就对应着两个完全不同的RTX表达式。为了解决这个问题,可以对上面得到的自动机模型进行修改。由于生成识别自动机的依据是指令模式,而对于分支跳转指令和数据传送指令来说,它们的指令模式相同,因此对分支跳转指令的支持并不会导致自动机形式上很大的改变。唯一需要修改的就是从状态15到状态16的转移条件。在自动机的原始模型中,如果在状态15时继续输入一个空字符,那么自动机会跳转到状态16,同时输出MOV类指令识别标识符。为了支持分支跳转指令,必须在这个状态转移过程中对目的操作数进行判断,即把自动机中状态15到状态16转移过程中的输出修改为一个条件输出。如果目的操作数是程序计数器PC,那么当前所识别的指令肯定是分支跳转指令,则输出BRANCH类指令识别标识符,否则仍然按照MOV类指令识别方法处理。

3.2 改进方案的实现及结果

参照改进后所得到的指令识别自动机模型,可对genmd的源代码进行修改。修改工作主要集中在两个部分:首先是修改状态16的输出函数,使之能够判别分支跳转指令,判断依据是指令的目的操作数(\$0)是否是特殊寄存器PC;其次是添加分支跳转指令的机器描述生成函数。当自动机识别出分支跳转指令后,调用该函数,并引用前面抽取的机器信息,向外部文件写入分支跳转指令的GCC RTL表达式^[8]。将修改后的源程序与genmd的其他部分一起编译,就可以得到一个可执行程序genmd。最后以Nios的Sim-nML描述文件为参数,执行该程序,就可以得到将GCC移植到Nios上所需的机器描述文件。下面是改进后的genmd所生成的关于跳转指令jmp的机器描述:

```
(define_insn "indirect_jump"
  [(set (pc) (match_operand 1 "register_operand" "a"))]
  ""
  "* {
```

```
switch( which_alternative) {
  case 0 :
    return \"jmp %1\";
    break;
  }
}
[ (set_attr "type" "branch")
 (set_attr "length" "4")])
```

从上述代码可以看出已经生成了具有分支跳转类指令的机器描述。在描述中,程序计数器PC作为第一个操作数,它体现了分支跳转指令与普通的数据传送指令之间的本质区别。如果该操作数不是PC,而是其他通用寄存器,那么该指令就是一条普通的数据传送指令。Jmp指令的汇编码输出由switch语句完成。这里,我们采用了统一的switch-case结构来生成汇编码,从而能够保证程序的简单和一致性。以上代码证明了本文对genmd的改进是成功的。

4 结语

genmd是一个基于ADL与传统编译器相结合来自动生成编译器思想的工具,它以Sim-nML为ADL语言,使用GCC作为传统编译器。genmd的目标是从Sim-nML描述自动生成移植GCC所需要的三个机器描述文件,从而实现GCC到新目标系统的移植。然而由于机器描述文件的自动生成过程本身很复杂,因此genmd目前的功能尚不完善,例如只支持很少的寻址方法和指令类型。本文对genmd的指令识别和机器描述生成部分进行了分析和建模,在此基础上,针对genmd不支持分支跳转类指令的缺点,提出了改进方案,并通过测试证明改进的正确性。今后的主要工作是以genmd为蓝本,设计出效率更高的机器描述自动生成模型和算法,使其达到实用化的程度。

参考文献:

- [1] LEUPERS R. Compiler Design Issues for Embedded Processors [J]. IEEE Design & Test of Computers, 2002, 19(4): 51-58.
- [2] LEUPERS R. Code Generation for Embedded Processors [A]. 13th Int. System Synthesis Symposium (ISSS) [C]. Washington DC: IEEE, 2000. 173-178.
- [3] GRUN P, HALAMBI A. Expression: An ADL for System Level Design Exploration [EB/OL]. http://www.cecs.uci.edu/~pgrun/publications/expression_tr.ps.gz, 1998.
- [4] RAJESH V, MOONA R. Processor Modeling for Hardware Software Codesign [A]. International Conference on VLSI Design [C]. Washington DC: IEEE, 1999. 132-137.
- [5] MOONA R. Sim-nML: Intermediate Representation Specification [EB/OL]. <http://www.cse.iitk.ac.in/sim-nml/moona/sim-nml/simnml-ir.ps.gz>, 2000-02.
- [6] Altera Co. Ltd. Nios Development Kit Getting Started [EB/OL]. http://www.altera.com/literature/ug/ug_nios_gsg_apex-20k200e.pdf, 2003-06.
- [7] POGDE P. Retargetable Code Generation Using Sim-nML description [EB/OL]. <http://www.cse.iitk.ac.in/sim-nml/moona/sim-nml/simnml-pogde-thesis.ps.gz>, 2000-04.
- [8] Free Software Foundation, Inc GCC Internals Manual [EB/OL]. <http://gcc.gnu.org/onlinedocs/gccint/>, 2003-11.