

基于 LISA 模型的汇编器研究与实现

魏 国¹, 吴 健¹, 阮 园²

(1 西北工业大学 计算机学院, 陕西 西安 710072; 2 中国电子科技集团公司 第五十八研究所, 江苏 无锡 214035)

摘 要: 介绍了一种基于 LISA 模型的汇编器设计方法, 着重分析了 LISA 模型的建模方法, 并针对一款 DSP 处理器的指令集进行建模, 在 LISA 模型的基础上借助 PD(Processor Designer)平台成功生成汇编器 lasm, 最后对所生成的汇编器进行了反汇编验证。

关键词: LISA; PD; 汇编器; DSP

中图分类号: TP314

文献标识码: A

文章编号: 1000-7180(2012)02-0114-05

Research and Implementation on LISA-based Assembler

WEI Guo¹, WU Jian¹, RUAN Yuan²

(1 College of Computer, Northwestern Polytechnical University, Xi'an 710072, China;

2 No. 58 Research Institute, China Electronics Technology Group Corporation, Wuxi 214035, China))

Abstract: To design and implement an assembler for target processor conveniently and efficiently, and verify processor design, this paper introduces a LISA-based assembler design method, and mainly analyses the modeling methods of LISA, models an instruction set of an DSP processor, and produces an assembler lasm successfully based on the LISA model. The produced assembler is verified according to disassemble results.

Key words: LISA; PD; assembler; DSP

1 引言

汇编器的作用是将用汇编语言编写的源程序转换成二进制形成的目标文件^[1], 并作为连接器的输入。设计新型处理器的同时必须为其开发配套的工具链, 而汇编器开发是其中的重要环节。然而编译工具链的复杂性决定了不可能在短时间内重新为一款芯片开发出高级语言编译器工具链^[2]。现有的汇编器开发环境有 GAS、NASM、TASM、a86 等等。本文中使用的 PD(Processor Designer)开发环境是由 CoWare 公司开发的处理器设计平台。PD 使用 LISA 语言对处理器进行建模, 实现了一种自动化的嵌入式处理器设计和优化环境。借助 PD 开发环境, 在 LISA 模型的基础上能够生成 C 编译器、宏汇编器、汇编器、链接器、反汇编器、调试器、模拟器等在内的一系列工具。

ZWXXCore 是一款高性能的 32 位微控制 DSP 处理器, 适用于通信领域、图像处理及视频处理的应用, 支持多种片上系统(SOC)。本文将根据 LISA 建模方法针对 ZWXXCore 的指令集建立模型。

2 LISA 建模基础

LISA(Language for Instruction-Set Architectures)语言是一种混合的功能/结构建模语言, 用来描述可编程处理器体系结构以及它们的外设和接口。该语言最新版本是 LISA2.0, LISA2.0 有效地提高了模拟处理器的灵活性, 同时显著减少了描述目标机所需的工作量^[3]。LISA 作为 PD(Processor Designer)开发环境的建模语言, 适于描述任何指令集驱动的目标机体系结构, 即那些功能由被称为指令集资源的特定资源决定的体系结构。在该范围内, LISA 提供了足够的灵活性, 它的语法元素能够覆

收稿日期: 2011-05-17; **修回日期:** 2011-07-29

基金项目: “核高基”重大专项基金资助项目(2009ZX01034-001-002-003)

盖包括 GP、RISC、DSPs、ASIPs、特殊功能协处理器等等在内的目标体系结构。

一个系统的行为是由它的内部状态以及内部状态和相邻内部状态之间的转换、内部状态和外部世界(输入/输出)之间的转换决定的。LISA 语言的基础结构主要基于一般处理器的两个主要属性,即存储元素和指令行为,也就是指处理器的内部状态和转换功能。基于这两个属性,LISA 处理器模型的基本对象主要有资源(resource)、操作(operation)。“资源”描述了处理器的存储元素;“操作”全局有效,是描述处理器完成转换功能的基本语言元素,它从设计者的角度表示目标机的行为、结构和指令集,包括与指令相关和指令无关的功能,例如取址操作^[4]。

一个完整的指令集通常用一个有向无环图表示,如图1所示。

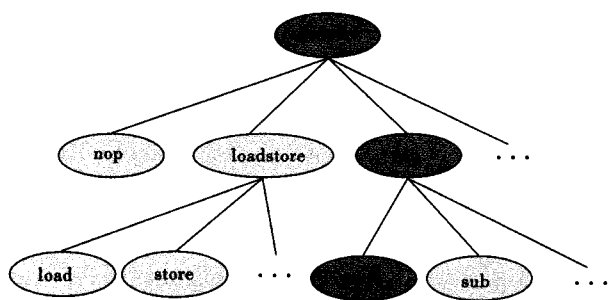


图1 LISA 指令集模型

LISA2.0 支持分层的操作描述,操作的层次是由对其他操作的引用建立的。所有的节点都有可能引用其他的对象。一个上层的非叶子节点最终由对下层的叶子节点的引用完成。一条指令不由一个操作定义的,而由一条类似图1中的操作序列决定。

通常,一个指令集的 LISA 模型树包含比实际指令集本身多得多的操作,只有在解码之后,指令的属性才是可见的。在寻址和取指阶段,对指令不做区分,也就是说,该过程中各个操作不能有 CODING 或者 SYNTAX 节(见下文)。第一个提供此类信息的操作被称为根操作,如图1中的 decode 节点就是根操作节点。如果我们要查找 add 指令,我们将从 decode 节点开始,依次向下找到 alu 节点和最终的 add 节点,其中 add 指令是属于 alu 类的指令。

一条指令的定义包括如下属性:

DECLARE:包含局部变量的声明,DECLARE 定义的内容将在后续的节(CODING、SYNTAX 等)中使用。

CODING:描述了作为指令集模型一部分的指令字的二进制映像,提供给汇编器、反汇编器、模拟

器使用。例如,汇编器使用编码信息将文本代码翻译为二进制目标代码,模拟器使用编码信息识别合法指令并选择对应的 LISA 操作。

SYNTAX:描述了作为指令集模型一部分的指令和它们的操作数的汇编语法。它提供的信息在处理器建模中主要用于两个方面:一是作为编译器翻译高级语言时的输出,二是作为汇编器、反汇编器处理文本代码与二进制代码之间转换的依据。

BEHAVIOR:描述操作模型的构成。BEHAVIOR 基于 C 语言描述操作的行为,最终 BEHAVIOR 中的 C 代码被转换为一个 C 函数体。这些函数体根据时序模型的合理调度,形成针对目标机的模拟器。在模拟的时候,BEHAVIOR 执行并改变资源的值,使系统进入一个新的状态。

EXPRESSION:用来直接访问资源或者访问某个编码元素表示的值,可以用来给 BEHAVIOR 节提供操作数。

ACTIVATION:描述与当前操作相关的其他操作的执行时机。它列出了后续将被执行的所有操作。激活的操作不一定能够马上执行,而是要根据它们的流水阶段属性,在相应的流水阶段之内才执行。

DOCUMENTATION:包含指令的相关文档。它允许你书写关于 OPERATION 模拟的部分指令集的文本描述。

由 LISA 模型产生的每一个工具都需要上述节集合的一个子集来完成特定的功能。例如,由于汇编器将指令语法转化为二进制代码,汇编器就要用到包含在 CODING 和 SYNTAX 节中的信息;由于反汇编器和汇编器的功能相反,故也需要用到 CODING 和 SYNTAX 节;而模拟器则严重依赖于 BEHAVIOR 的信息。

3 汇编器工作流程及原理分析

汇编器的输入文件 app.s 可以为宏汇编器 lasp 的输出文件或手写行汇编文件。汇编器 lasm 的输出文件为二进制的 LISA 目标文件。如图2所示。

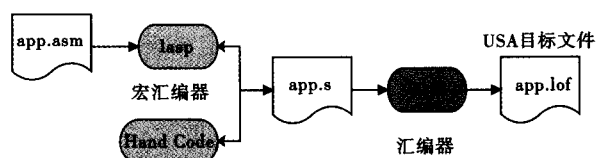


图2 汇编器的输入输出

汇编器使用 SYNTAX 节的信息分析汇编代码。假设汇编器读入一条汇编代码 add d2, d1, d2,

它将在 LISA 模型中自上而下地查找、匹配直到找到匹配的操作或者没有匹配对象退出,如图 3 所示^[5]。

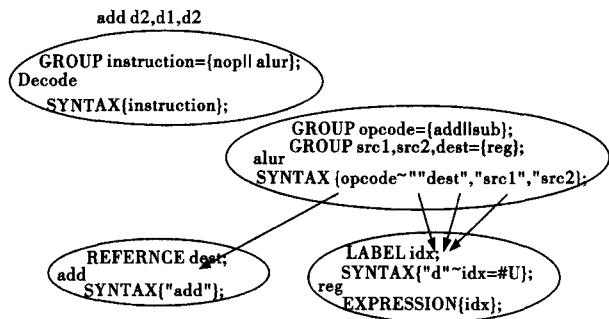


图 3 汇编器分析输入

汇编器使用 SYNTAX 节分析代码。在编码根节点,decode 操作的 SYNTAX 节确定可能的操作是 nop 还是 alur。nop 的 SYNTAX 不匹配,所以开始检查 alur 操作。alur 操作的 SYNTAX 节定义一条操作码,后面紧随一个空格。

add 的 SYNTAX 节符合汇编代码。目标寄存器和源寄存器匹配 reg 操作的 SYNTAX,reg 定义一个字母 d 后跟随一个无符号立即数。汇编器现在就知道了源寄存器和目的寄存器的 idx 值。

汇编器使用 CODING 节翻译指令到二进制的过程如图 4 所示^[5]。

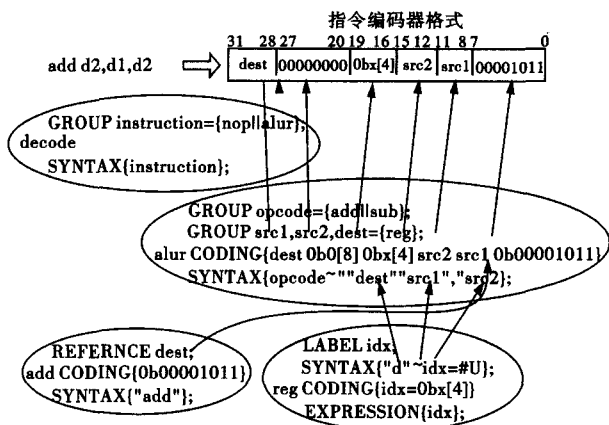


图 4 汇编器翻译助记符到二进制码

汇编器现在就可以利用 CODING 节生成二进制指令了。alur 的 CODING 节先定义表示目的寄存器的值,然后定义 8 位 0,接下来是 4 位保留位,再接下来是两个表示源寄存器的值。最终,指令由 8 位表示 add 操作的操作码结尾。汇编器获取以上信息后,就可以按照目标机指令格式将 add d2, d1, d2 翻译为目标二进制代码进行输出。

以上是汇编器识别汇编指令,并匹配产生二进

制代码的基本流程。值得注意的是,在建立模型过程中,16 位指令和 32 位指令要在根操作处进行分支判断,并将不同长度的指令写入不同的类别中。

4 目标 DSP 处理器的汇编器实现

使用 PD 平台生成工具链中任何工具都需要从 LISA 模型中提取信息。要实现针对目标 DSP 处理器的汇编器,首先需要建立 LISA 模型。建立的模型包括两方面的内容,一是对处理器资源建立模型,二是对指令集建立模型。

4.1 目标处理器及指令集简介

目标处理器 ZWXXCore 是一款高性能的 32 位微控制 DSP 处理器,支持 4 GBytes 地址寻址空间、小端模式存储、单指令多数据(SIMD)操作等,适用于通信领域、图像处理及视频处理的应用。

ZWXXCore 包含 32 个 32 位通用寄存器(GPRs)和三个系统寄存器,其中 16 个数据寄存器,16 个地址寄存器,一个程序计数器(PC)以及 32 位的程序状态字(PSW)和前文信息寄存器(PCXI)。

ZWXXCore 指令集包含算术运算、存取、FPU(浮点运算单元)、MMU(内存管理单元)等类型的指令,部分指令包含 16 位和 32 位两种长度编码格式。

4.2 建立处理器资源模型

处理器资源包括内部存储元素、专用输入/输出引脚和全局变量。内部存储元素包括寄存器和内部存储器。当模型为时钟级精度模型时,还包括流水寄存器、交互信号等处理器资源^[6]。生成汇编器需要建立的模型为指令级精度的,故本文不涉及流水寄存器等资源的描述。

处理器资源定义在 RESOURCE 关键字声明的节中,如下:

```
RESOURCE {
.....//该节的主体,包含处理器资源的定义
}
```

(1) 定义目标机寄存器

寄存器使用关键字 REGISTER 声明在 RESOURCE 节内。定义如下:

```
//16 个 32 位数据寄存器、16 个 32 位地址寄存器
REGISTER uint32 D[0..15];
REGISTER uint32 A[0..15];
//32 位程序计数器
PROGRAM_COUNTER uint32 PC;
//32 位前文信息寄存器、程序状态字寄存器
REGISTER TClocked<uint32> PCXI;
REGISTER TClocked<uint32> PSW;
```

(2) 定义目标机内存

内存使用关键字 RAM 和 MEMORY_MAP 定义在 RESOURCE 节内,并且有一些属性关键字描述内存属性.内存不同构成定义如下:

```
MEMORY_MAP
{ //数据和程序内存地址映射
  RANGE(0x00000000, 0x000fffff) -> prog_mem
[(31..2)];
  RANGE(0xffff0000, 0xffffffff) -> data_mem[(31..
0)];
}
MEMORY uint32 prog_mem
{ //程序内存,数据内存 data_mem 定义与之类似
  SIZE(0x00100000);
  BLOCKSIZE(8);
  FLAGS(R|X);
};
```

以上描述了两块内存,一个代码段内存 prog_mem,一个数据段内存 data_mem,每块内存有 0x1000000 个 32 位的寻址单元.在该定义中,0x100000 个代码段单元映射到范围 0x00000000~0x000fffff 的虚拟地址空间中,0x100000 个数据段单元映射到范围 0xffff0000~0xffffffff 的地址空间中,并且代码区以 2^2 大小增长,数据区以 2^0 大小增长.由于程序的后续处理需要链接器对二进制代码进行连接,以上内存地址空间定义为虚拟内存地址空间,在链接器连接后再将独立的虚拟内存空间进行合并或转化为物理地址.

(3) 定义目标机全局变量

当定义数据类型变量的时候,需要定义一些在整个 LISA 模型内全局有效但是不会映射到寄存器上的变量,LISA 语言提供了定义这种没有特定语义的全局变量的方式,即不使用额外的关键字(如 REGISTER),如下:

```
uint16 forwarded_address;
```

以上定义一个全局变量 forwarded_address,它可以在整个 LISA 模型内被访问,但是不会被当作寄存器或者其它有特定语义的处理器资源.

4.3 建立指令集模型

LISA 模型中指令集的模拟如前所述,一条指令是用如图 1 中所示的操作序列表示的.指令助记符中,16 个数据寄存器用 d[0..15]表示,16 个地址寄存器用 a[0..15]表示.数据寄存器的定义如下:

```
REGISTER data_reg
{ //数据寄存器
```

```
DECLARE {LABEL idx;} //LABEL 型变量 idx
SYNTAX {"d" ~idx= #U} //“d”后跟无符号数
CODING {idx=0bx[4]} //idx 定义为 4 位大小
EXPRESSION {idx} //返回 idx 值
}
```

地址寄存器的定义与数据寄存器类似,只是 SYNTAX 中的助记符不是“d”而是“a”.

一条指令被表示为一个包含 DECLARE、CODING、SYNTAX 节的 OPERATION.以加法为例,假如要描述一条将两个寄存器相加,并将结果放入另一个寄存器的加法指令,则描述如下:

```
OPERATION add_rr{
  DECLARE{GROUP src1, src2, dest={data_reg};}
  SYNTAX{"add" ~"" dest"," src1"," src2}
  CODING{dest 0b0[8] 0bx[4] src2 src1 0b00001011}
  .....
}
```

DECLARE 中声明 OPERATION 内其它节可用的变量,其中定义了三个 data_reg 类型的寄存器.

SYNTAX 节包含一个字符串或者一个字符串序列,表示汇编代码的语法.文本字符串用双引号引起.通常,SYNTAX 节也包含标签和特殊字符.

CODING 节模拟汇编助记符的二进制映像.CODING 包含一串位域,通常是由“0b”开头的任意数目的 0、1 或 x.重复的数字可以用 0b、数字、方括号内的重复次数表示,例如 0b1[5]等价于 0b11111. CODING 通常也包含变量.

4.4 汇编器功能验证

汇编器的输入及反汇编代码结果如图 5 所示,反汇编结果中的机器码为小端模式.结果证明,所生成的汇编器能够正确地完成汇编功能,并支持标号、常数以及一些预定义的符号,同时也支持节管理、数据定义、内存预约、内存对齐、符号处理、调试信息等相关类型的汇编伪指令操作.

```

1 .text
2
3 .global _start
4
5 _start:
6
7 add d2, d1, d2
8 add d3, d1, #126
9
10 sub d3, d3, d2
11 nop
12
13 .end
14
15 ***** THIS IS A GENERATED FILE *****
16 [00000000] <_start>: 210b 2000 add d2,d1,d2
17 [00000004]: e18b 3007 add d3,d1,#126
18 [00000008]: 230b 3080 sub d3,d3,d2
19 [0000000c]: 0000 nop
```

图5 汇编器输入及反汇编结果

5 结束语

目前所生成的汇编器已交付应用开发人员使

用,效果良好.使用 PD 基于 LISA 模型生成汇编器,开发人员只需专注于指令集建模,大大提高了汇编器开发人员的工作效率,同时支持符号处理和汇编伪指令,使针对特定目标机的汇编器开发更加方便、高效. PD 集成开发环境还针对汇编代码的优化问题提供了通用的窥孔优化器,如需针对目标机进行汇编代码优化,只需修改优化器配置文件即可实现.

参考文献:

- [1] 王晓栋. 基于嵌入式系统的交叉汇编器的研究与实现 [J]. 工业控制计算机, 2004, 17(3): 34-35.
- [2] 嵌入式系统下交叉汇编器的移植 [J]. 重庆邮电大学学报, 2009, 21(5): 650-652.
- [3] CoWare, Inc. LISA Language Reference Manual [EB/

DK]. 1996-2006.

- [4] CoWare, Inc. Assembler Reference Manual [EB/ DK]. 1996-2006.
- [5] CoWare, Inc. LISA Modeling Fundamentals [EB/ DK]. 1996-2006.
- [6] CoWare, Inc. Processor Designer Training Manual [EB/DK]. 1996-2006.

作者简介:

魏 国 男, (1986—), 硕士. 研究方向为计算机软件与理论.

吴 健 男, (1961—), 教授. 研究方向为 Web 计算、软件工程、服务计算.

阮 园 女, (1975—), 高级工程师. 研究方向为数字信号处理器.

(上接第 113 页)

参考文献:

- [1] 王海涌, 费峥红, 王新龙. 基于高斯分布的星像点精确模拟及质心计算 [J]. 光学精密工程, 2009, 17(7): 1672-1677.
- [2] Volder J E. The CORDIC trigonometric computing technique [J]. IRE Transactions on Computers, 1959 (C-8): 330-334.
- [3] 张建斌, 梁 芳, 刘乃安. 一种改进型 CORDIC 算法的 FPGA 实现 [J]. 微电子学与计算机, 2010, 27(11): 181-184.
- [4] Ahmed H M, Delosme J M, Morf M. Highly concurrent computing structure for matrix arithmetic and signal processing [J]. IEEE Comput, 1982, 15(9): 65-

82.

- [5] Hu Y H. The quantization effects of the CORDIC-algorithm [J]. IEEE Transactions on Signal Processing, 1992: 834-844.
- [6] 韩芳, 初建朋, 赖宗声. 一种 CORDIC 算法精度分析及其在 FFT 设计中的应用 [J]. 微电子学与计算机, 2004, 21(7): 14-16.

作者简介:

樊晓宇 男, (1985—), 硕士研究生. 研究方向为组合导航应用.

江 浩 男, (1971—), 副教授, 硕士生导师. 研究方向为导航、制导与控制.