第 33 卷 第 11 期 计 算 机 工 程 2007 年 6 月
Vol.33 No.11 Computer Engineering June 2007
·软件技术与数据库· 文章编号：1000—3428(2007)11—0044—03 文献标识码：A 中图分类号：TP309

# 一种递归式汇编级代码模块分析算法设计

蒋烈辉，周 博，费勤福，何红旗，韩小琨，张有为
(解放军信息工程大学信息工程学院，郑州 450002)

**摘 要：**提出了一种对主流微处理器汇编级代码在汇编级进行模块分析的算法，给出了该算法的形式化描述和模块分析结果的记录方法，根据该算法在汇编级代码辅助分析系统中的应用情况取得了实验数据。实验表明，该算法在多款处理器目标代码的分析过程中都有较高的模块分析速度与准确度。

**关键词：**程序理解；控制流；基本块；反汇编；二进制分析

# Design of Recursive Module Analysis Algorithm for Code at Assembly Level

JIANG Liehui, ZHOU Bo, FEI Qinfu, HE Hongqi, HAN Xiaokun, ZHANG Youwei
(Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002)

**【Abstract】** This paper proposes a generic module analysis algorithm for code at assembly level in majority of processors with different architectures. A formal description for the algorithm and the recording method for module analysis results are given. Experimental data based on the implementation of the algorithm in binary code assistant analysis system (BCAAS) demonstrate that the algorithm can be used towards binary code in different kinds of processors and achieves relatively high speed and accuracy.

**【Key words】** Program comprehension; Control flow; Basic block; Disassembly; Binary analysis

二进制代码分析现已被广泛应用于恶意代码发现、软件维护、二进制翻译、反编译等诸多领域，其中初期处理过程一般为：二进制代码先被转换成汇编语言或中间语言的形式，而后解析器分析出代码的模块信息（子程序、控制流信息、数据流信息等）。因此获取汇编代码的模块信息成为了非常重要的代码分析基础。

目前很多系统具有汇编代码分析功能，例如EEL[1]、OM[2]、ATOM[3]等，作者参与设计的二进制代码辅助分析系统也属于其中的一种。该系统采用了文中提出的模块分析算法，并且取得了很好的效果。本文给出了该算法的形式化描述，使得该算法可以在多种汇编代码模块分析的场合使用。另外由于该系统本身具有较好的用户可操控性，允许用户进行手动的模块调整操作，因此本文的模块分析结果记录方法特别适用于这方面的需求。文献[1~3]中都使用了程序切片技术[4]解决间接控制转移问题，二进制代码辅助分析系统采用了静态仿真技术解决间接转移。

## 1 二进制代码辅助分析系统

二进制代码辅助分析系统针对嵌入式系统中主流处理器程序的可执行代码（或二进制码）进行反汇编，并对反汇编后的程序进行模块分析与模块调整，从而达到不参照程序源代码即可准确分析程序结构、功能和漏洞并进行模块调整的目的。

二进制代码在进行反汇编之前须经过一个指令集识别的过程。确定处理器的类型之后，通用反汇编模块[5]利用处理器体系模块信息库中的信息对二进制代码进行反汇编。反汇编的结果经汇编指令预处理后被提交到模块分析与模块调整部分，同时经过中间语音抽象后以中间语言的形式提交到静态仿真模块。后者负责模拟程序的动态执行。模块分析与模块调整部分的分析结果通过用户接口显示给用户，同时用户可以通过此接口动态地对结构进行调整。

系统的总体框图如图 1 所示。

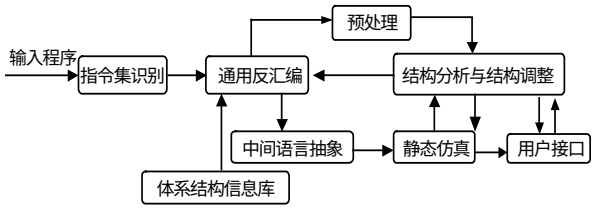


**图 1 系统体系结构**

指令集的不同使预处理部分将各款处理器的汇编代码用指令分类的形式进行统一。由于模块分析侧重于分析程序的控制流程，因此只须将改变控制流的指令进行分类。大致可以分为：调用类指令（如 call），跳转类指令（如 jmp），返回类指令（如 rtn）。

## 2 模块分析算法

模块分析：指从汇编指令序列中抽取子程序和基本块，并分析各子程序之间以及各基本块之间关系的过程。通常，反汇编指令序列的模块化可采用自底向上和自顶向下 2 种方法。自顶向下的模块化方法首先将全部的指令序列作为一个虚拟的子程序，再通过分析将虚拟的子程序逐渐拆分成真正

**作者简介：**蒋烈辉(1967－)，男，博士生、教授，主研方向：嵌入式技术，嵌入式系统逆向分析；周 博，硕士生；费勤福，硕士、副教授；何红旗，硕士、讲师；韩小琨，硕士生；张有为，硕士

**收稿日期：**2006-07-24 **E-mail：**zhoubo2000@hotmail.com

意义上的子程序或基本块[6]。为了更准确地描述模块化算法，给出几个概念的形式化定义。

**定义 1** 指令序列 由某一指令集指令构成的指令序列，定义为五元组:

$$prog(I)=(A,e,map,ctg,jtg) \tag{1}$$

其中，$I$ 表示某一指令集；$A$ 连续，$|A|<\infty$，表示指令序列所覆盖的所有地址；$e\in A$，表示指令序列的入口地址；$map: A\to I$ 表示从 $A$ 中某地址到该地址处指令的映射；$ctg: A\to Set(A)$表示从 $A$ 中某地址到该地址处调用类指令的目的地址的映射，其中 $Set(A)$表示地址的集合；$jtg: A\to Set(A)$表示从 $A$ 中某地址到该地址处改变 PC 值类指令目的地址的映射。$[\min(A),\max(A)]$表示指令序列的地址范围，$\min(A)\geq 1$表示指令序列的起始地址，$\max(A)$表示指令序列的结束地址。这里的指令序列即未模块化的程序。

**定义 2** 程序块 设有某一指令集上的一段指令序列 $prog(I)=(A,e,map,ctg,jtg)$，则程序块定义为二元组：

$$block=(B,e) \tag{2}$$

其中，$B\subseteq A$且 $A$ 连续，表示程序块所覆盖的所有地址；$e\in B$，表示程序块的入口地址；$B^*$表示某一指令序列中所有程序块的集合；$B0\in B^*$表示一个特殊的不属于任何指令序列的程序块；$[\min(B),\max(B)]$表示程序块的地址范围，$\min(B)\geq 1$表示程序块的起始地址，$\max(B)$表示程序块的结束地址。

**定义 3** 子程序 设某一指令集上有一段指令序列 $prog(I)=(A,e,map,ctg,jtg)$，$B^*$是该段指令序列中所有程序块的集合，则子程序定义为二元组：

$$subroutine=(S,e) \tag{3}$$

$subroutine\subseteq Set(B^*)$且 $\cup_{block\in subroutine} block$ 所覆盖的所有地址连续，其中，$S$ 表示子程序所覆盖的所有地址；$e$ 表示子程序的入口地址；$[\min(S),\max(S)]$表示子程序的地址范围，$\min(S)\geq 1$表示子程序的起始地址，$\max(A)$表示子程序的结束地址。

### 2.1 拆分指令序列

模块分析算法总体上分为 2 个步骤：(1)将指令序列模块化为若干子程序的集合；(2)将指令序列模块化为若干基本块的集合。在整个模块化过程中有一些用于拆分指令序列的函数贯穿始终，本节将给出它们的形式化描述。

函数 $inblock: A\to B^*$表示从某地址到其所在的程序块的映像。如果输入的地址不属于任何程序块，则此函数值取 B0。

函数 $split: (B^*\times A)\to Set(B^*)$表示将某一个程序块在某一指定地址处拆分成 2 个程序块。给定的地址作为地址顺序靠前的程序块的结束地址。若给定的地址是程序块的结束地址，则函数不改变原程序块。

函数 $splitset: (Set(B^*)\times A)\to Set(B^*)$表示将一个程序块集合用某一指定的地址拆分成另外一个程序块集合。设有程序块集合 $B'$，指定地址为 $\alpha$，则函数的值为

$$splitset(B',\alpha)=\begin{cases} B' & if\ inblock(\alpha)=B0 \\ B'\backslash\{inblock(\alpha)\}\cup split(inblock(\alpha),\alpha) & otherwise \end{cases}$$

其中，$B\backslash\{inblock(\alpha)\}$表示从集合 $B'$ 中除去程序块 $inblock(\alpha)$；函数 $splitrecurs: (Set(B^*)\times Set(A))\to Set(B^*)$表示将一个程序块集合用一组指定地址的集合递归拆分成另外一组程序块集合。设有程序块集合 $B'$，指定地址集合 $A'$，则函数的值为

$$splitrecurs(B',A')=\begin{cases} B' & if\ A'=\varphi \\ splitrecurs(splitset(B',\alpha),A'\backslash\{\alpha\}) & \alpha\in A' \\ & otherwise \end{cases}$$

在每次递归中，函数从指定地址集合中随机选出某一指定地址对程序块集合进行拆分。在完成本次拆分之后，指定地址从集合中删除。函数在指定地址集合为空时结束。这里需要注意 2 点：(1)从地址集合中选取地址的顺序是任意的；(2)经过此函数处理后，如果 $\max(A)$不属于 $A'$，程序块的个数是 $|A|+1$。

### 2.2 子程序级模块分析

预处理后的指令序列首先被模块化到子程序级。将 2.1 节中函数 *splitrecurs*()的 2 个输入参数分别指定为整个指令序列（虚拟子程序）和子程序入口点地址减 1 的集合，便可将指令序列划分成若干个程序块，这些程序块的开始地址就是子程序的开始地址。

子程序的入口点地址分为 3 类：(1)指令序列中调用类指令的目标地址；(2)中断处理程序的入口地址；(3)程序的开始地址。对于第 1 类地址可以通过函数 *ctg*()得到，其输入参数为调用类指令的语句地址；第 2 类、第 3 类地址在预处理过程中分析得到。

对于间接寻址，子程序开始地址的确定可能存在不足，此类地址无法通过函数 *ctg*()得到。间接寻址的问题可以通过静态仿真技术解决。

确定各子程序的开始地址后，使用拆分后的程序块集合与各子程序的结束地址作为函数 *splitrecurs*()的输入参数，将指令序列拆分成为若干子程序和游离段的集合。子程序的结束地址即返回类指令的语句地址，可通过汇编指令序列得到。

指令集的多样性使返回类指令也存在多样性。但返回类指令一定是改变 PC 值类的指令。因此，需要确定是在子程序间跳转，还是在子程序内部跳转。设指令的语句地址为 $\alpha$，若 $inblock(jtg(\alpha))\neq inblock(\alpha)$，则作为返回类指令处理；否则不作为返回类指令。

### 2.3 基本块级模块分析

基本块的确定仍然用 *splitrecurs*()函数进行。根据基本块的定义，将以下 2 类地址构成的集合作为函数的第 2 个输入参数：

(1)跳转类指令的语句地址：

$C1=\{\alpha\mid map(\alpha)\in$ 跳转指令$,\alpha\in A\}$

(2)跳转类指令的目标地址减 1：

$C2=\{\beta\mid \beta=jtg(\alpha)-1,\alpha\in C1\}$

### 2.4 关系建立

指令序列被拆分成子程序集合或基本块集合之后，需要分析各子程序之间或各基本块之间的调用或跳转关系。为此定义 2 种图：(1)调用图：反映指令序列中各子程序之间的关系，图中的各结点代表各个不同的子程序，连接各结点之间的边代表子程序调用；(2)基本块图：反映子程序内部各基本块的关系，图中的各结点代表各个不同的基本块，连接各结点之间的边代表控制流的改变。

构造调用图需要找到每个子程序结点的后继结点。设某子程序 $s$，令 $E(s)=\{\alpha\mid\alpha=ctg(\alpha),\alpha\in s\}$，结点 $s$ 的后继结点是含有 $E(s)$中任何一个元素的结点。构造基本块图的方法是，当 $jtg(\max(block))=\phi$时，后继地址应该取 $\max(block)+1$，令

$$E(s)=\{\alpha\mid\alpha=jtg(\max(block))\wedge jtg(\max(block))\neq\phi\}$$
$$E(s)'=\{\alpha\mid\alpha=\max(block)\wedge jtg(\max(block))=\phi\}$$

结点 $s$ 的后继结点时含有 $E(s)\cup E(s)'$中任何一个元素的结点。

## 3 模块分析结果的记录

经过上述模块分析步骤，汇编程序的模块信息已被抽取。

图 2 显示了某段代码模块分析后的结果。

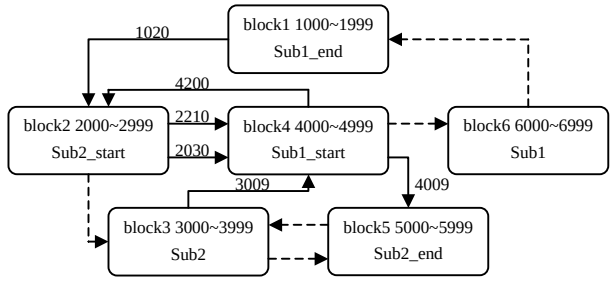


**图 2 结构分析结果**

图 2 中每个结点代表一个基本块，结点中标有基本块的标识号、地址范围和所属的子程序（start 子程序的开始，end 所属子程序的结束）;带箭头的虚线表示子程序内基本块级程序流程，带箭头的实线表示调用类指令引起的程序流程转移，而实线上的权值代表调用类指令的语句地址。

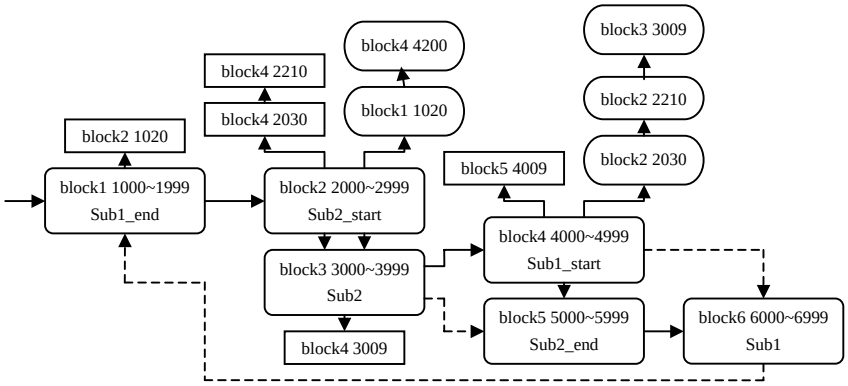


**图 3 交叉链表**

可以采用交叉链表的形式准确完备地记录模块分析结果信息。每个基本块都用结构 Block 记录，并用单链表将每个 Block 结构链接起来；调用类指令的语句地址和目标基本块号用结构 CallNode 记录，并用单链表将 CallNode 结构链接起来，链表的头结点链接到对应的 Block 结构上；调用某基本块的调用类指令的语句地址和其所属基本块号用结构 CalledNode 记录，并用单链表将 CalledNode 结构链接起来，链表的头结点链接到对应的 Block 结构上；对于从属于同一子程序的基本块，其对应的 Block 结构也通过链接的方式来表示基本块级的程序流程。图 2 中的模块分析结果用交叉链表记录后的结果如图 3 所示。

## 4 测试与展望

本模块分析算法已经被成功地运用于上文中提到的二进制代码辅助分析系统。表 1 是针对 3 款处理器上运行的可执行代码进行模块分析的测试数据。

**表 1 测试数据**

| 程序编号 | 程序大小（B） | 处理器名 | 处理器类型 | 指令条数 | 子程序数 | 识别度(%) | 基本块数 | 识别度(%) | 模块化速度(s) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 903 | MCS51 | MCU | 1 216 | 18 | 100 | 147 | 100 | 3.2 |
| 2 | 5 350 | MCS51 | MCU | 3 273 | 44 | 100 | 376 | 100 | 5.3 |
| 3 | 12 103 | 8086 | MPU | 7 469 | 87 | 90.8 | 847 | 87.7 | 18.6 |
| 4 | 53 896 | 8086 | MPU | 31 153 | 376 | 90.4 | 3 572 | 88.2 | 76.3 |
| 5 | 3 772 | MC68000 | MCU | 2 008 | 30 | 94.4 | 232 | 93.1 | 5.1 |
| 6 | 113 249 | MC68000 | MPU | 69 203 | 846 | 93.6 | 8 017 | 90.5 | 188.7 |

本文提及的返回类指令多样性问题、子程序识别问题以及模块分析效率问题处于进一步的研究中。二进制代码辅助分析系统的下一目标是增加反编译模块。

### 参考文献


1 Larus J R, Schnarr E. EEL: Machine-independent Executable Editing[C]//Proceedings of the ACM SIGPLAN'95 Conference on Programming Language Design and Implementation. 1995-06: 291-300.

2 Larus J R, Ball T. Rewriting Executable Files to Measure Program Behavior[J]. Software Practice & Experience, 1994, 24(2): 197-218.

3 Srivastava A, Eustace A. ATOM: a System for Building Customized Program Analysis Tools[C]//Proc. of the SIGPLAN '94 Conference on Programming Language Design and Implementation. 1994-06: 196-205.

4 Weiser M. Program Slicing[C]//Proceedings of the 5th International Conference on Software Engineering. 1981-03: 439-449.

5 吴金波, 蒋烈辉, 赵　鹏. 基于控制流的反汇编算法研究[J]. 计算机工程与应用, 2005, 41(30): 89-90.

6 Aho A V, Sethi R, Jeffrey D. Compilers: Principles, Techniques, and Tools[M]. Addison-Wesley, 1985.


（上接第 30 页）

### 参考文献


1 Agrawal R, Imielinski T, Swami A. Mining Association Rules Between Sets of Items in Large Databases[C]//Proceedings of ACM SIGMOD Conference on Management of Data. 1993: 207-216.

2 Agrawal R, Srikant R. Fast Algorithms for Mining Association Rules [C]//Proceedings of the 20th Int'l Conference on very Large Databases. 1994.

3 Lin D I, Kedem Z M. Pincer-search: A New Algorithm for Discovering the Maximum Frequent Set[C]//Proc. of the 6th European Conf. on Extending Database Technology. 1998.

4 Zaki M J. Scalable Algorithms for Association Mining[J]. IEEE Transactions on Knowledge and Data Engineering, 2000, 12(3): 372-390.

5 Dunham M H. 数据挖掘教程[M]. 郭崇慧, 田凤占, 靳晓明, 译. 北京: 清华大学出版社, 2005-05.

6 王晓峰, 王天然. 相关测度与增量式信任度和支持度的计算[J]. 软件学报, 2002, 13(11): 2208-2214.

7 王晓峰, 王天然. 基于双空间搜索的频繁项挖掘方法[J]. 计算机科学, 2002, 29(4): 55-60.

8 王晓峰, 王天然, 赵　越. 一种自顶向下挖掘频繁项的有效方法[J]. 计算机研究与发展, 2004, 41(1): 148-155.

9 吕　静, 王晓峰, Adjei O, 等. 序列模式图及其构造算法[J]. 计算机学报, 2004, 27(6): 782-788.

10 王晓峰, 尹丹娜. Apriori 算法在红外光谱数据挖掘中的应用[J]. 计算机与应用化学, 2001, 18 (5/6): 477-483.