汇编嵌入式软件程序流程图自动生成的研究

汪文勇 王学东 向 渝 唐 科 刘利枚
(电子科技大学计算机学院 成都610054)

摘 要 程序流程图的自动生成是软件结构分析及软件测试的基础。为实现汇编嵌入式软件程序流程图的自动生成，本文首先分析汇编嵌入式软件的特征，将汇编指令划分为5种典型类型，分别定义其单链表存储结构，给出链表生成算法。在此基础上，给出汇编嵌入式软件程序流程图自动生成算法。最后，对算法的时间复杂度进行了分析。

关键词 汇编语言，嵌入式软件，程序流程图，流程图自动生成，软件测试

Research on Automatic Construction of Program Flow Graph for Assembly Embedded Software

WANG Wen-Yong WANG Xue-Dong XIANG Yu TANG Ke LIU Li-Mei
(School of Computer Science, UESTC, Chengdu 610054)

Abstract The purpose of this paper is to discuss the automatic generation of program flow chart of assembly embedded software. To do so, assembly instructions are classfied into 5 types according to their influence on the execution routes of the programs. An algorithm on storing and generating link tables of the 5 types of instructions and the automatic generation algorithm are presented, and it's time complexity is analysed.

Keywords Assembly language, Embedded software, Program flow graph, Automatic generation, Software testing

1 引言

目前嵌入式软件的开发语言以C/C++等高级语言为主。针对高级语言的测试，研究人员已经提出了一些典型的算法和思想[1,2]。而嵌入式系统的另一类型的开发语言是汇编语言，这是在一些特定的场合和应用背景下经常遇到的[3]。汇编语言是非结构化的程序设计语言，在具体编程时可能有很多跳转指令，不易阅读，更不易判断程序的执行路径。在很多实际应用中主程序是一个没有如END这样结束语句的死循环，并且源程序可以只有主程序或只有中断子程序，也可以包括主程序和若干个中断子程序。

上述特点决定了对汇编嵌入式软件结构的自动分析及其程序流程图的自动生成是一件困难的工作。本文提出了一种针对汇编语言程序结构进行分析的通用算法(本文以下部分的讨论均基于Intel 8051单片机指令系统)，成功地解决了汇编嵌入式软件程序流程图的自动生成问题，特别是子过程的多出口分析，为后面的程序结构评估、自动程序插桩、覆盖测试等环节奠定了坚实基础。

2 算法基本框架

本算法基于对汇编指令的分类来进行源程序的静态结构分析。根据每条汇编指令的执行对源程序执行流程所造成影响的不同，定义5种特定的基本块(顺序块、无条件跳转块、条件转移块、子调用块、RET块)数据结构。在对源程序进行初扫描时，将源程序分割成由这5种基本块所构成的结点的一个集合，每种类型的块用链表的形式组织成一个单链表，形成5个不同的单链表，然后按照一定的匹配原则来决定链表中每个结点的后继结点，最终令5个单链表中的结点相互之间发生一定的关联，进而形成一个对应于源程序语句级别的拓扑有序的程序流程图。

算法的基本流程框图如图1所示。

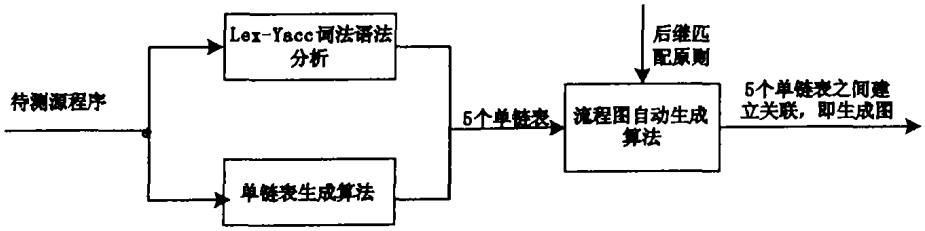


图1 算法框图

3 算法描述

3.1 汇编语句的分类

我们将汇编指令分为以下5种类型，分类的依据是看指令是否会引起程序流程的转移。

(1)普通指令类型：类型号为0。这种指令的执行不会令程序流程的线性顺序发生改变，也就是说，这种指令执行后，程序流程的走向为源程序文本上线性顺序的后续指令，如ADD,SUBB,INC等等。

(2)无条件转移指令类型：类型号为1，如JMP,AJMP,LJMP,SJMP。

(3)子调用指令类型：类型号为2，如LCALL,ACALL。

(4)子程序返回和中断返回指令类型：类型号为3，包括RETI,RET。

(5)条件转移指令类型：类型号为4，如 JZ，JNZ，JC 等。

3.2 基本块的划分

对应于上述5种汇编指令类型，我们定义5个单链表，分别加以存储。这5个单链表的结点分别为：①线性块：由1～n 条类型为0的指令组成；②无条件跳转块：由类型为1的指令组成；③子调用块：由类型为2的指令组成；④RET 块：由类型为3的指令组成；⑤条件转移块：由类型为4的指令组成。

在每种块的数据结构中存放必要的信息，如块的起始和终止行号、引用标号、定义标号，以及指向后继结点的指针等。

3.3 五个单链表的建立

对程序进行扫描的时候，得到当前语句的一些参数：如语句类型、引用标号、定义标号、行号、前一条语句类型等参数，然后根据这些参数，进入不同的处理模块：无条件跳转链表处理模块、有条件跳转链表处理模块、线性块链表处理模块等。因此该部分只建立五个链表，为以后建立程序的流程图提供信息。即后面的程序的流程图的建立就不是扫描文件了，而是在5个链表中搜索，从而建立5个链表之间的相互关系(即程序流程图)。

3.4 相关术语及数据结构

为了建立流程图我们首先将一些用到的术语和数据结构约定如下：

算法中止：待测源程序本身是一个病态程序，出现逻辑错误或语法错误，致使本算法无法继续进行而非正常退出。

算法结束：在没有出现令算法中止的错误的前提下，本算法已无法再进行下一次的匹配，这说明流程图的建立已成功完成。

子调用块跟踪堆栈：在本算法的进行过程中，每当遇到一个“子调用结点”时，需要对该结点进行跟踪，我们用一个堆栈来实现这个数据结构，这个堆栈取名为 Track 堆栈。该堆栈元素称为“子调用块跟踪结点”，其中又有两个跟踪链表和一个计数器。

条件块跟踪链表：在本算法的进行过程中，每当遇到一个“条件转移块结点”，需要对该结点的位置进行跟踪。我们用一个链表来实现这个数据结构。

RET 跟踪链表：在本算法的进行过程中，每当在子过程的分析过程中遇到一个“RET”结点，需要对该结点的位置进行跟踪。我们用一个链表来实现这个数据结构。

条件块跟踪链中未顺序匹配的计数器：该变量主要是节省遍历条件跟踪链表，通过该计算数器的值，可以知道“条件块跟踪链表”中还有多少个顺序分支没有被分析过的条件转移块。

3.5 对条件转移块的处理

对于每个条件转移块结点，我们约定从该结点的引用标号分支开始分析，在适当的时候再返回到该结点对其另一分支(顺序分支)进行分析，因此对于每个子程序或者主程序，每当遇到一个“条件转移结点”，就将该结点链接在一个“条件块跟踪链表”中。当对一个局部范围内(例如一个子过程)的所有“条件转移块结点”的标号分支都分析完成后，就遍历这个“条件块跟踪链表”，以便对每个“条件转移块结点”的顺序分支进行处理。

在汇编程序中，循环的构成是由条件转移指令来实现的，因此，在遇到一个“条件转移结点”时，还要判断一次该“条件转移块结点”在本局部范围内的“条件块跟踪链表”中是否已经存在。如果已经存在，则说明这个“条件转移结点”构成了一个循环，就不应该对这个结点的当前分支(标号分支或顺序分支)再继续分析下去，而应该跳出这个循环，去对链表中的其他“条件转移块结点”进行顺序分支的分析。

3.6 对 RET 块的处理

在对子过程的分析过程中，由于子过程内部“条件转移块结点”的可能存在，从源程序文本的静态角度来看，这就造成一个子过程有可能存在着多个出口，即在该子过程中有多个 RET 块结点。在传统的解决方案中，对用户的待测源程序施加了一些限制，比如说限制一个子过程只能有一个出口[4]。本算法成功地解决了这个问题。在程序的分析过程中，每遇到一个“RET 块结点”，我们就将该结点链接在一个“RET 块跟踪链”中，然后在适当的时候再返回来队这个跟踪链表内的所有 RET 块结点进行处理，也就是对所有的 RET 块结点进行后继结点的匹配。

3.7 对子过程的处理

在对一个子过程内部的程序结构进行分析时，应该确保在该子过程分析完成之前不能跳出该子过程，而且前面提到的对条件转移块和 RET 块结点的处理都应该局限在一个子过程的内部。因此，在处理到一个“子调用块结点”时，就为该结点建立一个“子调用块跟踪结点”，以便在进入该特定的子过程中进行分析时，确保所以得分析行为都是针对该子过程的。也就是说，每一个子调用块都有其特定的“条件块跟踪链表”和“RET 块跟踪链”。对于源程序的主程序，将其视为子过程的一种特例，也为其建立一个“子调用块跟踪结点”。所有的“子调用块跟踪结点”用一个链栈(“子调用跟踪堆栈”)来实现，处于栈顶的“子调用块跟踪结点”就对应于当前正在分析的子过程。一个子过程分析完成以后，将栈顶元素出栈，表示我们的分析从一个子过程中退回到它的外层过程里继续进行分析。

针对子过程的多次调用问题，为了提高分析效率，应该避免对同一个子过程作重复的分析，因为一个子过程无论被调用多少次，其内部的静态程序结构是不变的。在算法的分析过程中，每当遇到一个“子调用块结点”，先判断该节点所引用的子过程是否已被分析过，判断方法：遍历“子调用跟踪堆栈”，是否有某个“子调用块跟踪结点”所对应的子过程名与当前遇到的“子调用块结点”所调用的子过程名相同。如果是，则说明这个子过程已经被分析过了，接下来所作的只是需要对这个子过程内部的所有“RET 块结点”的后继指针从新指向新的后继。从这点来看，每个“RET 块结点”的后继也需要用一个链表来实现，也就是说，针对同一个子过程的多次调用，其内部的“RET 块结点”在每次调用后都有不同的后继。这也是本算法的相对于已有算法的重要改进之一。

3.8 算法描述

1. 根据用户的选择，置全局变量 StartLayerType = 0 或 1；

2. 为最外层程序建立一个“子调用块跟踪结点”，压入“子调用块跟踪堆栈”栈顶，并置其成员 CurLayerType = StartLayerType；

3. 以“行号为1”为关键字在5个单链表中搜索一个匹配的后继结点，以此作为整个流程图的起始结点；

4. 判断上一步搜索到的后继结点类型：

若为“顺序块”，则：

(1)以该节点的“结束行号＋1”为关键字在5个单链表中搜索一个匹配的后继结点，若成功，则：在图上链接上该结点，

并转第4步；若不成功，则：继续；

(2)判栈顶元素的 CurLayerType 是否为0，若否，则：出错，算法中止；若是，则：继续；

(3)判栈顶元素的 UnSeqMatchCount 是否为0，若是，则：栈顶元素出栈，算法结束；若否，则：A. 在栈顶元素的"条件块跟踪链表"中搜索一个标记 SeqMatchFlag 为0的"条件块跟踪结点"；B. 将其标记 SeqMatchFlag 置1；C. 栈顶元素的 UnSeqMatchCount 减1；D. 以跟踪到的"条件块结点"的"行号＋1"为关键字在5个单链表中搜索一个匹配的后继结点，若成功，则：在图上链接上该结点，并转第4步；若不成功，则：转(3)；

若为"无条件跳转块"，则：

(1)以该节点的"引用标号"为关键字在5个单链表中搜索一个匹配的后继结点，若成功，则：在图上链接上该结点，并转第4步；若不成功，则：出错，算法中止；

若为"子过程调用块"，则：

(1)判该结点在"子调用块跟踪堆栈"栈中是否存在过：若是，则：在该结点"调用行号链"链尾记录本次"调用行号"，然后转第5步；若否，则：继续；

(2)为该结点建立一个"子调用块跟踪结点"，压入"子调用块跟踪堆栈"栈顶，并置其成员 CurLayerType 为1；

(3)以该节点的"引用标号"为关键字在5个单链表中搜索一个匹配的后继结点，若成功，则：在图上链接上该结点，并转第4步；若不成功，则：出错，算法中止；

若为"条件转移块"，则：

(1)判该结点是否已存在于栈顶元素的"条件块跟踪链表"中，若是，则：转(4)；若否，则：继续；

(2) A. 为该结点建立一个"条件块跟踪结点"，置其标记 SeqMatchFlag 为0；B. 将此"条件块跟踪结点"加入栈顶元素的"条件块跟踪链表"；C. 栈顶元素的 UnSeqMatchCount 加1；

(3)以该节点的"引用标号"为关键字在5个单链表中搜索一个匹配的后继结点，若成功，则：在图上链接上该结点，并转第4步；若不成功，则：出错，算法中止；

(4)判栈顶元素的 UnSeqMatchCount 是否为0：若是，则：转第5步；若否，则：继续；

(5) A. 在栈顶元素的"条件块跟踪链表"中搜索一个标记 SeqMatchFlag 为0的"条件块跟踪结点"；B. 将其标记 SeqMatchFlag 置1；C. 栈顶元素的 UnSeqMatchCount 减1；D. 以跟踪到的"条件块结点"的"行号＋1"为关键字在5个单链表中搜索一个匹配的后继结点，若成功，则：在图上链接上该结点，并转第4步；若不成功，则：转(4)；

若为"RET 块"，则：

(1)将该结点加入栈顶元素的"RET 块跟踪链表"；

(2)判栈顶元素的 UnSeqMatchCount 是否为0：若是，则：转第5步；若否，则：继续；

(3)A. 在栈顶元素的"条件块跟踪链表"中搜索一个标记 SeqMatchFlag 为0的"条件块跟踪结点"；B. 将其标记 SeqMatchFlag 置1；C. 栈顶元素的 UnSeqMatchCount 减1；D. 以跟踪到的"条件块结点"的"行号＋1"为关键字在5个单链表中搜索一个匹配的后继结点，若成功，则：在图上链接上该结点，并转第4步；若不成功，则：转(2)；

5. (1)判栈顶元素的 CurLayerType 是否为0：若是，则：栈顶元素出栈，算法结束；若否，则：继续；

(2)判栈顶是否等于栈底：若是，则：A. 遍历"RET 块跟踪链表"，在跟踪到的每个 RET 块的"后继链"链尾加上一个后继结点，并将其后继指针置空；B. 栈顶元素出栈，算法结束。若否，则：A. 以栈顶元素的"调用行号链"链尾所记录的"调用行号＋1"为关键字在5个单链表中搜索一个匹配的后继结点，若成功，则：遍历"RET 块跟踪链表"，在跟踪到的每个 RET 块的"后继链"链尾加上一个后继结点，并将其后继指针链上搜索成功得到的后继结点，并栈顶出栈，转第4步；若不成功，则：出错，算法中止。

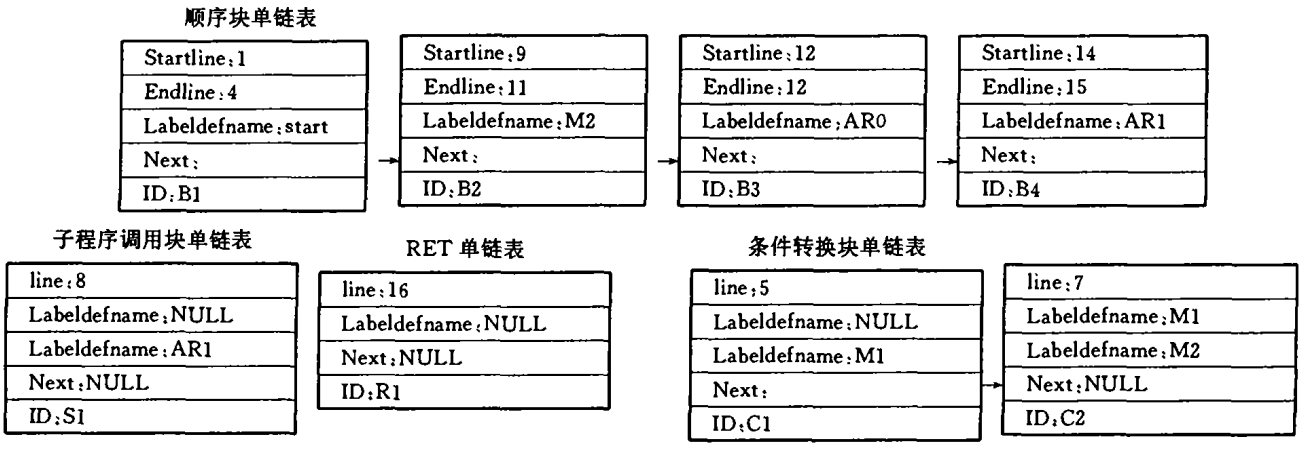
4 算法复杂度分析

从以上的分析过程可以看出，本算法的基本操作是在针对每个结点匹配其后继结点时，都要对5个单链表进行一次搜索。我们假设5个单链表的总结点数为 n，则在最坏情况下，时间复杂度为：$T(n) = O(n^2)$。

5 实验结果

下面我们以一段小程序为例，按照本算法生成其程序流程图。

```
1 START:  MOV    DPTR,#2000H
2         MOVX   @DPTR,A
3         ANL    P1,#0FH
4         MOV    P2,A
5         JNB    P1.0, M1
6         LJMP   AR0
7 M1:     JNB    P1.1, M2
8         LCALL  AR1
9 M2:     SETB   P1.2
10 MOV    SP,    #61H
11 END
12 AR0:   CLR    P1.0
13 LJMP   M2
14 AR1:   MOV    DPTR, #3000H
15 MOV    28H,   #00H
16 RET
```



(下转第201页)

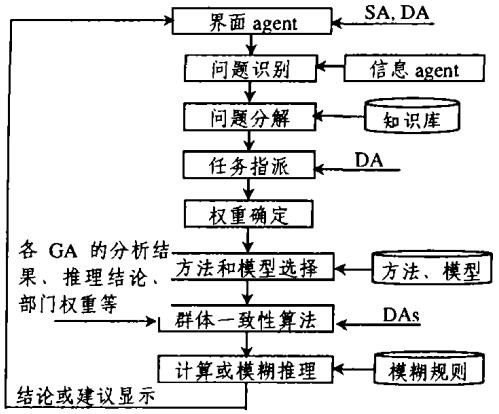


图5 GA 的决策实现

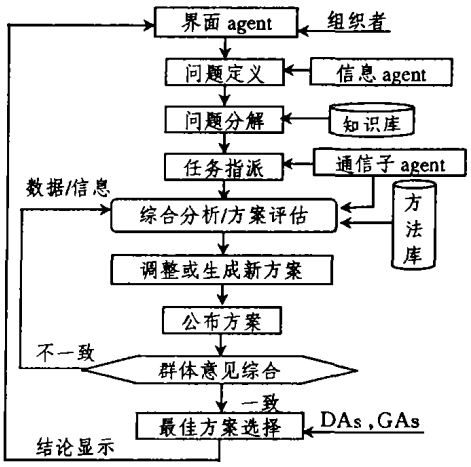


图6 SA 的决策实现

3 基于 MAS 的企业 GDSS 的实现过程

通过以上分析，基于 MAS 的企业 GDSS 的基本实现过程为：1)由组织者(协调员)提出决策问题；2)SA 的任务 agent 将决策问题进行描述后分解为子问题指派给不同的 agent 群，各 agent 群将这些子问题继续分解后求解或指派给不同的 DA；3)各部门的信息 agent 通过信息检索、信息过滤、信息分析后由任务 agent 利用 agent 的建模功能、推理技术和预测模型进行问题求解、推理或预测；4)由 GA 将各个 DA 的分析结果进行群体一致性问题求解后，向 SA 提供建议；5)SA 对各 agent 群提供的建议加以综合分析后，向群体决策成员提供第一轮的统计处理结果，形成若干决策方案，并解释说明这些决策方案的背景材料，然后群体成员通过界面 agent 进行交互，对所提供的所有方案进行打分和分级判断；6)SA 根据各方案的得分情况和群体成员的意见，选择群体意见一致的方案，再转向第2步；7)如此反复多次直至得出一个意见统一的决策结果。

结束语 通过建立基于多层次 MAS 的企业 GDSS，避免了诸如社会化、求同压力、少数人支配等人为的缺点，系统具有很好的模块性、动态性、层次性、开放性和可扩展性，有效地降低了系统的复杂性和构造成本，提高了决策效率，缩短了决策周期。

参 考 文 献

1 Postmes T, Lea M. Social processes and group decision making: anonymity in group decision support systems. Ergonomics, 2000, 43(8): 1252～1274
2 Abhijit G, Pushkala P. Understanding GDSS in symbolic context: shifting the focus from technology to interaction. MIS Quarterly, 2000, 24(3): 509～546
3 Jackie R, Gary J K. An Evolutionary Approach to Group Decision Making. INFORMS Journal on Computing, 2002, 14(3): 278～292
4 Neal G S, Ahmad M, Surya B Y. A comprehensive agent-based architecture for intelligent information retrieval in a distributed heterogeneous environment. Decision Support Systems, 2002, 32: 401～415
5 Kwon O B, Lee K C. MACE: multi-agent coordination engine to resolve conflicts among functional units in an enterprise. Expert Systems with Applications, 2002, 23: 376～389
6 Ronald R Y. Defending against strategic manipulation in uninorm-based multi-agent decision making. European Journal of Operational Research, 2002, 141: 217～232

(上接第175页)

无条件跳转块单链表

line:6
Labelredname:NULL
Labelrefname:AR0
Next:
ID:U1

line:13
Labeldefname:MULL
Labelrefname:M2
Next:NULL
ID:U2

结束语 本文将汇编语言的指令分为5种类型，对应于5种基本块。通过扫描被测源程序生成5种基本块的单链表。进一步通过扫描这5个单链表建立程序流程图。由于汇编嵌入式程序结构的复杂性，建立相应程序流程图的算法也非常复杂。试验证明，本文提出的算法是正确有效的。

参 考 文 献

1 郑人杰，殷人昆，陶永雷. 实用软件工程(第二版). 清华大学出版社，1997
2 孙林，岳丽华，贾文举. 一种从源程序代码到其流程图的自动转换算法. 微计算机应用，2001，22(3)：181～186
3 吴金戍，沈庆阳，郭庭吉. 8051单片机实践与应用. 清华大学出版社
4 http://www.autosoft-jitong.com.cn/products/crests.asp
5 严蔚敏，吴伟民. 数据结构. 北京：清华大学出版社，1997. 156～192
6 郑人杰. 计算机软件测试技术[M]. 北京：清华大学出版社，1992
7 Huang J C. Program Instrumentation and Software Testing [J]. Computer, 1978, 11(4): 3
8 Hawkins J, Howard R B, Nguyen H V. Automated real-time testing for embedded control system [DB/OL]. IEEE, 2002
9 刘超. 程序交互执行流程图及其测试覆盖准则. 软件学报，1998(6)
10 邓支益，何亦征. 嵌入式软件测试研究. 田冀航空电子技术，2003(3)

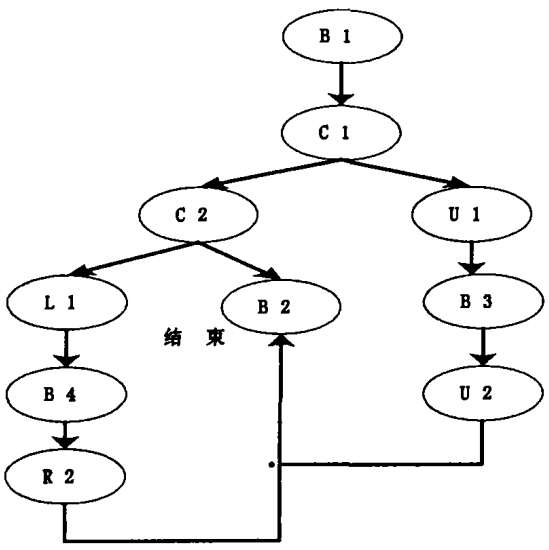


图2 算法生成的链表及实验结果