Cortex-M3 反汇编程序的设计与实现

孙俊香[1] 于连庆[2]
[1](潍坊学院计算机工程学院 山东 潍坊 261061)
[2](北京信维科技股份有限公司 北京 100191)

摘 要 工程实践中,出于项目开发维护、程序调试,或者逆向工程的需要,经常需要对二进制代码进行反汇编。针对 Cortex-M3 处理器 bin 格式代码反汇编程序的总体设计思路以及一些技术难点,给出程序总体结构和技术难点解决思路,最后利用实现的反汇编程序给出 LPC1768 处理器的反汇编实例。无论是工程实践,还是学习 ARM 指令系统,或者是 C 语言编程实践,该项目都是极好的课题。

关键词 反汇编 ARM Cortex-M3 条件指令

中图分类号 TP313 **文献标识码** A DOI: 10.3969/j.issn.1000-386x.2014.05.066

DESIGN AND IMPLEMENTATION OF CORTEX-M3 DISASSEMBLER

Sun Junxiang[1] Yu Lianqing[2]
[1] (*School of Computer Engineering, Weifang University, Weifang 261061, Shandong, China*)
[2] (*Beijing Shineway Technologies Inc., Beijing 100191, China*)

Abstract In engineering practice, for the purpose of project development and maintenance, program debugging or reverse engineering, there often has the needs to disassemble the binary code. Aiming at the overall design idea and technological difficulties of disassembler of Cortex-M3 processor in binary format, we put forward a solution for overall structure and technical difficulties of the program. In end of the paper, we present the disassembling example of LPC1768 processor by using realised disassembler. This project is a very good subject for engineering practice, or learning ARM instruction set, or C language programming practice.

Keywords Disassembler ARM Cortex-M3 Conditional instruction

0 引 言

ARM 家族的 Cortex-M3 是一个通用的 32 位微处理器,它具有高性能和超低功耗的特性,近几年在各个领域得到了广泛的应用。出于项目开发维护、程序调试,或者逆向工程的需要,经常需要对二进制代码进行反汇编。目前广泛使用的 Keil uVision4 开发工具平台,可对 ELF 格式的目标代码进行反汇编,尚不支持二进制格式。通过文献检索,尚未发现公开的针对 Cortex-M3 处理器的二进制格式代码反汇编工具[1-3]。笔者根据项目需要,设计了可在 Linux/Windows 环境下运行的针对 Cortex-M3 处理器的二进制格式代码反汇编工具。

1 Thumb-2 指令

任何一款微处理器都有自己专用的指令集,ARM 处理器一直支持两种形式上相对独立的指令集,32 位的 ARM 指令集,对应处理器状态为 ARM 状态;16 位的 Thumb 指令集,对应处理器状态为 Thumb 状态。在程序的执行过程中,处理器可以动态地在两种执行状态之中切换,这种切换是由程序员完成的。

Thumb-2 是最近才加入 ARM 架构的一项新技术,是 16 位 Thumb 指令集的一个超集[6,7]。在 Thumb-2 中,16 位指令首次与 32 位指令并存。在 Thumb-2 技术下也不再像以往那样需要在 ARM/Thumb 两套指令之间切换。

Cortex-M3 系列处理器只支持 Thumb-2 指令集。

2 技术难点

二进制格式代码中既有指令代码,也有数据,反汇编程序需要正确识别哪些是指令,哪些是数据。另外,Thumb-2 指令集中引入了特有的 IF-THEN 指令,也是反汇编的难点之一。

2.1 数据的识别

如前所述,Thumb-2 指令是定长的,32 位或 16 位,对 32 位立即数的访问无法内嵌在指令码中实现,通常是通过文字池来实现,即在代码区的某个地址存放数据。汇编指令一般通过 PC 寄存器相对寻址来实现。反汇编程序需要解决的是如何根据指令码找出这些数据,并利用 DCD 伪指令定义这些常数。例如:

```
标号      指令码    汇编
L000120   4820      LDR R0, [PC, #128]; L0001A4
… …
L0001A4   20000020  DCD 0x20000020
```

我们希望把 0x20000020 存放在寄存器 R0 中,指令码中无法内嵌数据 0x20000020,先把数据存放在标号地址 L0001A4

收稿日期:2013-04-23。孙俊香,讲师,主研领域:嵌入式软件开发。于连庆,高工。

处,通过相对寻址方式实现存取。

2.2　条件指令(IT)

ARM 汇编指令可以有条件地执行,但在 Cortex-M3 中,指令中对条件后缀的使用有限制,只有转移指令(B 指令)才可随意使用。而对于其他指令,CM3 引入了 IF-THEN 指令块,在这个块中的指令才可以加后缀,且必须加以后缀。IF-THEN 块由 IT 指令定义。

IF-THEN(IT)指令围起一个块,里面最多有 4 条指令,它里面的指令可以条件执行。IT 已经带了一个"T",因此还可以最多再带 3 个"T"或者"E"。并且对 T 和 E 的顺序没有要求。其中"T"对应条件成立时执行的语句,"E"对应条件不成立时执行的语句。在 IF-THEN 块中的指令必须加上条件后缀,且"T"对应的指令必须使用和"IT"指令中相同的条件,"E"对应的指令必须使用和"IT"指令中相反的条件。IT 的使用形式如下:

```
IT <cond>;                          围起 1 条指令的 IF-THEN 块
IT<x> <cond>;                       围起 2 条指令的 IF-THEN 块
IT<x> <y> <cond>;                   围起 3 条指令的 IF-THEN 块
IT<x> <y> <z> <cond>;               围起 4 条指令的 IF-THEN 块
```

其中 <x>,<y>,<z> 的取值可以是"T"或者"E"。

而 <cond> 则是在参考文献[6]的 A6.3 中列出的条件(AL 除外)。例如:

标号	指令码	汇编
L001468	BF0C	ITE EQ
	2000	MOVEQ R0,#0
	2001	MOVNE R0,#1

反汇编程序需要解决的是如何根据指令码找出 IT 指令中的条件,根据这些条件正确反汇编 IF-THEN 块中的指令。

2.3　异常处理向量表

缺省情况下,CM3 异常处理向量表位于零地址处,各向量占用 4 字节。每个表项占用 4 字节,存放处理例程的入口地址。CM3 有 256 个预定义的异常类型,最多支持 240 个外部中断,但具体使用了多少个由芯片生产商决定。

反汇编程序需要确定向量表的具体大小,找到第一条指令代码的起始地址。

3　设计要点

3.1　总体结构

如前所述,Thumb-2 指令序列由 16bits 对齐的半字组成。在这个序列中,每条指令或者是 16bit,或者是由两个半字组成的 32bits,详情可参考文献[6]。基于此,选择了简单的表格驱动算法来实现这一程序,而不采用逻辑驱动算法,因为前者比后者更容易测试。

如参考文献[6]所描述的,如果一个半字的 bits[15:11]为 0b11101、0b111100、0b11111,那么这是一条 32bits 指令,否则就是 16bits 指令。如表 1 所示。

表 1　LDR 指令格式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
LDR <c> <Rt>, [<Rn{,#<imm5>}]															
0	1	1	0	1	imm5					Rn			Rt		

我们可以忽略指令中的 imm5、Rn 和 Rt 域,而只对其中明显标为 0 或 1 的二进制位感兴趣。在上例的情况下,我们只对其高 5 位感兴趣。如果去掉该指令中没有明显标为 0 或 1 的其他进制位,然后检查其高 5 位是否为 01101 或 16 进制数 0x68,就可以知道该指令是否是 LDR 指令:

```
if((inst & 0x6800) == 0x6800)
它是条 LDR 指令 ……
```

可以用这种先屏蔽后测试的办法来检查每条汇编指令,一旦确认为 LDR 指令就可调用译码函数恢复刚才忽略的 imm5、Rn 和 Rt 区域的内容。

这就是我们设计的反汇编程序的工作方式。

首先定义一个数据结构,描述指令识别特征码、掩码和译码函数:

```
typedef struct {
        unsigned short pat;                          //指令识别特征码
        unsigned short mask;                         //掩码
        unsigned int (* pcDecode)(unsigned short inst, unsigned int
offset);                                             //译码函数
} identity;
```

然后,定义一个数组,描述所有可能的指令:

```
static identity idInst[] =
{
{ 0xE800, 0xF800, pcDecode32bitInst },
{ 0x4400, 0xFC00, special_data_process_16bit },
… …
};
```

译码函数:

```
static unsigned int pcDisasm(unsigned short inst, unsigned int addr)
{
unsigned int rc;
identity * pid;

pid = &idInst[0];
while ((pid->mask != 0) &&((inst& pid->mask) != pid->pat)) pid ++;
rc = pid->pcDecode(inst, addr);
return rc;
}
```

我们看到,函数 pcDisasm 并不很大。它使用的算法非常简单:把当前的指令与掩码相与,在表中特征码比较,若匹配则调用相应的译码程序。译码程序返回一个修改后的程序计数器,因为指令可能是 16 位或者 32 位。

3.2　数据的识别

定义一个 256K 大小的 int 型数组,数组下标就是数据地址。如果某个地址对应的是数据,而不是指令代码,相应的数组位置置 1。

```
int is_data[1024 * 256];
```

识别到 PC 寄存器相对寻址指令,从指令码中计算出数据的存放地址,把相应的数组位置置 1,二次扫描时把相应的地址表示为数据,用 DCD 伪指令定义这些常数。

3.3　条件指令(IT)

定义两个数据结构,记录 IT 指令的相关信息。

```
typedef struct {
u_int count;                                    //IT 块中包含几条指令
u_int index;                  //扫描时记录处理到了 IT 块中第几条指令
char cond [4] [4];                              //每条指令对应的条件
} IT_BLOCK_t;
IT_BLOCK_t IT_block;
typedef struct {
char flag;                                      //该条指令在 IT 块中? 1 - 是
char cond [5];                //若该条指令在 IT 块中,相应的执行条件
} INSIDE_IT_t;
INSIDE_IT_t inside_IT [1024* 256];
                                        //256K,数组下标是指令代码地址
```

IT 指令的分析识别非常复杂,主要的实现思路如下。

IT 后面可能出现的组合:

```
static char * IT_3bit [8] = {"TTT", "TTE", "TET", "TEE", "ETT", "ETE", "EET", "EEE"};
static char * IT_2bit [4] = {"TT", "TE", "ET", "EE"};
static char * IT_1bit [2] = {"T", "E"};
```

IT 指令编码形式如表 2 所示。

表 2 IT 指令的编码形式

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
IT{x{y{z}}}				<firstcond>											
1	0	1	1	1	1	1	1	firstcond				mask			

其中,mask 占 4bit,表示 IT 后是 xyz、xy、x 或者没有,具体编码见参考文献 [6] 的 A6.7.37。firstcond 编码见参考文献 [6] 的 A6.3。

识别函数如下:

```
void decode_IT( char * buf, u_char cond, u_char mask)
{
unsigned char m,c,bit0,id;

bit0 = cond & 0x01;

// four instructions in the IT block.
if ( mask & 0x01) {
    m = ( mask & 0x0E) > > 1;
    c = ( bit0 < < 2) | ( bit0 < < 1) | bit0;
    id = m^c;
    strcpy( buf, IT_3bit [id]);
  }
  else {
    // three instructions in the IT block.
    if ( mask & 0x02) {
      m = ( mask & 0x0C) > > 2;
      c = ( bit0 < < 1) | bit0;
      id = m^c;
      strcpy( buf, IT_2bit [id]);
    }
    else {
      // two instructions in the IT block.
      if ( mask & 0x04) {
        m = ( mask & 0x08) > > 3;
        c = bit0;
        id = m^c;
        strcpy( buf, IT_1bit [id]);
      }
    }
  }
  return;
}
```

对应 T 的条件:

```
char* condition_flags [14] = {
  "EQ", "NE", "CS", "CC", "MI", "PL", "VS",
  "VC", "HI", "LS", "GE", "LT", "GT", "LE"
};
```

对应 E 的条件:

```
char * reserve_condition_flags [14] = {
  "NE", "EQ", "CC", "CS", "PL", "MI", "VC",
  "VS", "LS", "HI", "LT", "GE", "LE", "GT"
};
```

根据识别出的 IT 指令,把相应的条件存在 IT_block 的 cond 域中。译码程序根据有关标志正确反汇编。

3.4 异常向量表

CM3 有 256 个预定义的异常类型,最多支持 240 个外部中断,但具体使用了多少个由芯片生产商决定,因此可以通过配置文件或者人机界面的形式指定异常向量表的大小。笔者实现的程序中通过配置文件指定。

4 运行实例

LPC1768 是恩智浦(NXP)公司基于 ARM 公司的 Cortex-M3 内核的微控制器[8],主要针对高集成度和低功耗的嵌入式应用。LPC1768 的异常向量表地址为 0x00-0xC8,第一条指令的起始地址为 0xCC。反汇编出的部分代码片段:

```
标号      指令      汇编代码
L000000  10002868  DCD 0x10002868
L000004  000000E1  DCD 0x000000E1
… …
L0000C4  000013C1  DCD 0x000013C1
L0000C8  000013C1  DCD 0x000013C1
L0000CC  F8DFD00C  LDR   R13, [PC,#12]
L0000D0  F000F878  BL    L0001C4
L0000D4  4800      LDR R0,     [PC,#0]; L0000D8
L0000D6  4700      BX    R0
L0000D8  00005949  DCD   0x00005949
… …
L000BCC 4608   MOV     R0, R1
L000BCE 2D08   CMP     R5, #8
L000BD0 D008   BEQ     L000BE4
L000BD2 2D09   CMP     R5, #9
L000BD4 D006   BEQ     L000BE4
… …
```

利用辅助工具去除指令编码域的内容,可以直接编译生成二进制代码。

```
AREA     RESET, DATA, READONLY
L000000   DCD 0x10002868
L000004   DCD 0x000000E1
… …
L0000C4   DCD 0x000013C1
```

```
L0000C8    DCD 0x000013C1
AREA    |.text|, CODE, READONLY
L0000CC    LDR    R13,    [PC,#12]
L0000D0    BL    L0001C4
L0000D4    LDR    R0,    [PC,#0]; L0000D8
L0000D6    BX    R0
L0000D8    DCD 0x00005949
L0000DC    CMP R0, #104
L0000DE    ASRS R0, R0, #0
L0000E0    LDR    R0,    [PC,#12]; L0000F0
L0000E2    BX    R0
L0000E4    B    L0000E4
… …
L000BCC    MOV    R0, R1
L000BCE    CMP    R5, #8
L000BD0    BEQ    L000BE4
L000BD2    CMP    R5, #9
L000BD4    BEQ    L000BE4
… …
    END
```

5 结　语

Cortex-M3 系列处理器采用的 Thumb-2 指令集是一套非常复杂的指令集,本文介绍了实现反汇编程序中的主要技术难点和实现思路。无论是工程实践,还是学习 ARM 指令系统,或者是 C 语言编程实践,本项目都是极好的课题。目前笔者实现了 Linux 环境下的运行版本,通过增加可视化人机界面,后续的工作中将实现 Windows 环境下的运行版本。

参　考　文　献

[1] 宋威,曾勇军,奚琪. 一种动静态结合的代码反汇编技术[J]. 计算机工程,2012,38(1):68-70,73.

[2] 马金鑫,忽朝俭,李舟军. 基于控制流精化的反汇编方法[J]. 清华大学学报,2011(10):1345-1350.

[3] 崔晨,李清宝,胡刚,等. 基于中断向量表重构的固件代码反汇编技术[J]. 计算机科学,2012(7):302-304.

[4] 彭波. 基于 ELF 文件的智能反汇编技术研究[J]. 微计算机信息,2011(6):28-29.

[5] 王振华,陈宝财,卢琦. 目标代码静态反汇编技术研究与实现[J]. 现代计算机,2010(3):163-167.

[6] ARM Limited. ARMv7-M Architecture Reference Manual[CP/DK]. ARM Limited,2010. http://infocenter.arm.com/help/index.jsp.

[7] Joseph Yiu. ARM Cortex-M3 权威指南[M]. 宋岩,译. 北京: 北京航空航天大学出版社,2009.

[8] NXP Semiconductors. LPC17xx User manual[CP/DK]. NXP Semiconductors,2010. http://www.nxp.com/products/microcontrollers/cortex_m3/series/LPC1700.html.

(上接第 246 页)

和卡图现象,将每次重定位时需要重新检测整个图像全部关键点,改进为结合跟踪失败位置的二维坐标,搜索邻域 320 × 240 像素范围内图像。这样只要对该邻域图像进行 1/4 的局部搜索就可得到特征集中区域,大大缩小特征点查找范围。改进前后跟踪匹配效率对比效果如表 1 所示。

表 1　改进前后跟踪匹配效率对照表

匹配和跟踪效率参数	改进前	改进后
特征检测数量(个)	368	252
特征匹配数量(个)	75	43
特征匹配时间(ms)	168.47	122.36
跟踪失败重新检测特征(个)	75	26
重新跟踪匹配时间(ms)	168.47	56.26

由表 1 可知,经过预处理的彩色图像转化为经过灰度化、二值化、双边滤波处理的高对比度图像,检测到的特征点数量明显减少。经过双向匹配改进匹配点数量精简近一半; 相应地正反匹配和特征检测的总时间,对比于改进前只有正向匹配的时间也得到优化。最后由于跟踪失败后缩减搜索区域后重定位的特征检测数量和匹配时间大大减少,跟踪匹配整体耗时减少近 50%,系统整体性能得到一定提升。

5 结　语

增强现实因其自然的人机交互方式和强烈的现实沉浸感而被美国时代周刊、高德纳(Gartner)咨询公司等权威机构列为未来十大最有前景的技术之一。现广泛应用于数字营销、设计仿真、科教以及移动导览等领域。

本文是基于局部自然特征 SIFT 研究而研发的增强现实系统,系统在进行算法的部分改进后,具有更好的实时性和鲁棒性,适用于户外复杂变化的环境下的数字平台展示。由于局部特征的描述特征能力的有限性,结合深度学习的中层 patch 将会是未来的一个研究热点。另一方面结合体感摄像头获得的 RGB-D 数据在人机交互方面也将有更广阔的发展空间。

参　考　文　献

[1] 陈亮. 桌面增强现实环境下的虚实融合和三维交互技术研究[D]. 北京: 北京邮电大学,2011.

[2] 李江. 基于 ARToolKit 的增强现实技术在恐龙博物馆中的应用研究[D]. 成都: 西南交通大学,2010.

[3] 李向阳. 增强现实环境下自然特征点的注册方法研究[D]. 广州: 广东工业大学,2011.

[4] 李玉,王涌天,刘越,等. 基于彩色标识点的增强现实注册算法研究[J]. 系统仿真学报,2008,20(3):654-661.

[5] 刘嘉敏,安乐祥,常燕,等. 增强现实中基于三角标识的三维注册方法[J]. 沈阳工业大学学报,2013,35(1):79-84.

[6] Daniel Leiis Baggio,Shervin Emami,David Millan,et al. OpenCV with Practical Computer Vision Projects[M]. UK(Birmingham): Packt Publishing Ltd,2012.

[7] 陈明,陈一民,黄诗华,等. 增强现实中的视频对象跟踪算法[J]. 计算机工程,2010,36(12):229-231.

[8] 陈家琪,郝妍. 一种改进的 SIFT 特征反向匹配算法[J]. 软件时空,2011,27(3):208-211.

[9] 张立鹏. 基于自然特征的增强现实实时跟踪研究[D]. 上海: 华东理工大学,2012.

[10] 马颂德,张正友. 计算机视觉(计算机理论与算法基础)[M]. 北京: 科学出版社,1998:52-92.

[11] Luo Juan,Oubong Gwun. A Comparison of SIFT,PCA-SIFT and SURF[J]. International Journal of Image Processing,2009,3(4):143-152.