TMS320C54X DSP 汇编程序的几种优化方法

李章林[1]，吴岳[2]，卢桂章[1]

（1 南开大学信息学院机器人所　天津 300071，2 南开大学信息学院通信工程系　天津 300071）

摘要：本文主要通过研究 TMS320C54X DSP 汇编指令的特点及其流水线特性提出了四种优化其汇编程序的方法，它们是“部分循环展开法”、“并行指令使用技术”、“合理利用指针增减的思想”、“ARO 作为循环次数法”。其中“部分循环展开法”是消除循环内部多余 NOP 语句的通用方法；“并行指令使用技术”提出了一种增大并行指令使用几率的通用方法。“合理利用指针增减的思想”是一种提高程序效率的编程思想。对于内层循环次数随着外层循环递增或者递减的二重循环，可用“ARO 作为循环次数法”提高其效率。四种方法是从实际工作中抽取出来，具有通用性，其优化思想对其它具有流水线结构的 MCU 的汇编程序优化也具有一定指导作用。

关键词 DSP；汇编；优化；TMS320C54X

Some Optimization Methods for TMS320C54X DSP Assembly Language Programs

Li Zhanglin[1] ,Wu Yue[2] , Lu Guizhang[1]

(1 Institute of Robotics and Automatic Information System, Nankai University, Tianjin 300071; 2 Communication Engineering Department, Nankai University, Tianjin 300071)

ABSTRACT: Through analyzing the instructions and pipeline feature of TMS320C54X DSP, we promoted four optimization methods for TMS320C54X DSP assembly language programs, which were “Techniques of Partly Expanding Rotation”, “The Techniques about Using Parallel Instructions”, “The Proper Utilization of Increase or Decrease Pointers”, “Use AR0 as a Rotation Counter”. “Techniques of Partly Expanding Rotation” is a method to avoid redundant NOP instructions in rotation; “The Techniques about Using Parallel Instructions” provides a method to increase the probability of using parallel instructions; “The Proper Utilization of Increase or Decrease Pointers” is a way to increase efficiency of programs; “Use AR0 As a Rotation Counter” will be applied to the optimization of two-layer rotations whose inner rotation time increases or decreases with its outer rotation. The 4 methods were abstracted from practice, and had general-purpose nature. Its optimization idea may also be helpful for program-optimization of other MCUs with pipeline feature.

Key word: DSP; Assembly language; Optimization; TMS320C54X

引言

TMS320C54X DSP（简称 54xDSP） 芯片是美国 TI 公司的一款定点 DSP 芯片[1]，它在信号处理领域使用非常广泛，但是对于该 DSP 的汇编程序的优化技巧方面的资料却比较少。对于特定的程序结构，例如循环结构、分支结构、二重循环等，存在一套比较通用的优化策略，本研究就是旨在找出这些程序结构的通用优化策略，这里根据 54xDSP 的指令特点和其流水线特性提出了 4 种优化方法，4 种方法主要针对循环结构的优化。

我们知道程序的速度优化关键点在于优化循环体的效率。对于循环体的优化这里主要从几个方面入手：①减少为了避免流水线冲突而加入的 NOP[2]指令（空操作指令）。②用一条并行指令代替两条指令。③采用指令周期少的指令。接下来的 4 种方法用于解决以上问题。

1 部分循环展开法

在 54xDSP 汇编程序中，经常会出现 NOP 指令，例如为了防止流水线冲突用 NOP 隔开相邻的指令，又如延迟跳转指令后常跟两个 NOP。为了提高程序效率，特别是 NOP 出现在循环体内部时，我们往往通过调整语句（即汇编指令）顺序的方法将后面或者前面的指令调整到 NOP 指令所在位置，以替换这个 NOP 指令。部分循环展开技术就是用于调整循环体内语句顺序从而替换 NOP 指令的一种方法。请看例 1（54xDSP 汇编

程序代码[3])。

```
                例 1
    loop1:
        STM     #1, AR1;将 1 放入 AR1
        BANZD   loop1, *AR2-;跳转
        NOP                 ;空指令
        NOP                 ;空指令
```

例 1 无法通过调整指令顺序替换循环体内的两个 NOP。以下采用部分循环展开法来解决这个问题。

部分循环展开法有两种：前向部分循环展开和后向部分循环展开。先看一下基本循环体结构如图 1 所示。对该基本循环体进

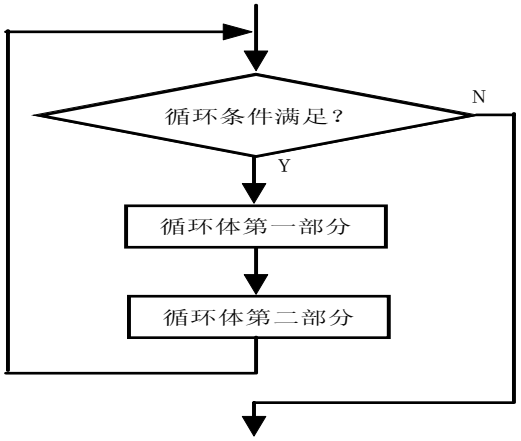


图1. 基本循环体结构
(Structure of a basic rotaion)

行前向部分循环展开和后向部分循环展开以后的程序流程如图 2 和图 3 所示。从

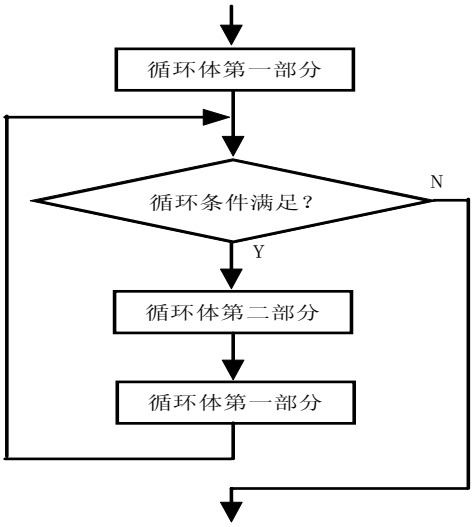


图2. 部分循环前向展开后的循环体结构
(Rotation structure after forward "Partly Expanding Rotation")

图 2、图 3 可知，前向部分循环展开后的流程和展开前的流程的区别是：展开后最后一次循环再执行一次“循环体第一部分”模块（见图 2）；后向部分循环展开后和展开前的流程的区别是：展开后，第一次循环先执行一次“循环体第二部分”模块。它们的共同的效果是使得循环体内两模块调换了顺序，从而为替换 NOP 提供了条件。将例 1 中的 STM #1, AR1 语句视为“循环体第一部分”，对例 1 使用前向部分循环展开。展开后 STM #1, AR1 的位置调整到两个 NOP 的后面，由于 STM #1, AR1 为双周期指令[4]正好可以替换这两个 NOP，最后的结果如下：

```
STM    #1, AR1;提取到循环体外，见图 2
loop1:
   BANZD  loop1, *AR2-
   STM    #1, AR1;已经替换 2 个 NOP
```

展开后程序执行结果不受影响，因为展开后和展开前程序流程的唯一区别是：展开后，最后一次循环多执行一次 STM #1,AR1 语句，该语句改变了 AR1 的内容，但在本程序中 AR1 的内容不被后续程序使用，所以多执行一次 STM #1,AR1 语句并不影响程序执行结果。

设例 1 的循环体循环 N 次，那么使用部

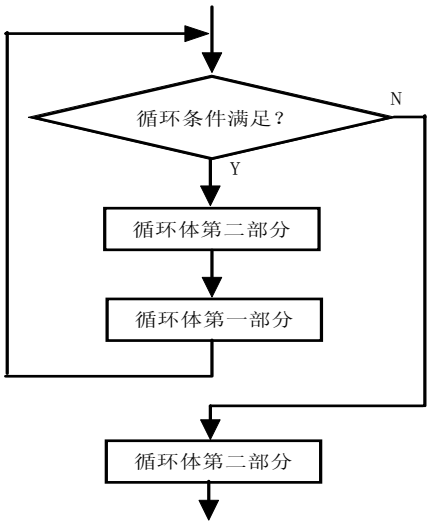


图3. 部分循环后向展开后的循环体结构
(Rotation structure after backward "partly Expanding Rotation")

分循环展开共节省了 2×N－2 个指令周期。

部分循环展开法使得循环内部前后语句顺序可调换，这不仅可用于消除 NOP，也可用于其它需要调换语序的场合。

2 并行指令使用技术

使用并行指令可用一条指令实现两条

指令的功能，将提高程序效率。但是只有在两条特定的语句挨在一起时，才能合并这两条语句。为了提高合并机会，一个简单的方法是通过调整语句顺序，让特定的语句挨在一起，但是调整语句顺序是在不改变程序逻辑的前提下进行的，调整的自由度有限。这里我们介绍一种增大调整语句顺序自由度的方法，用该方法将增大使用并行指令的机会。另外 54xDSP 的并行指令有 3 种类型[5]：①LD 和 ST 的合并② ST 和算术指令的合并③LD 和算术指令的合并。其中第 3 种情况要求指令中两个累加器必须不同。所以为了提高合并语句的机会还应尽可能让有可能合并的语句使用不同的累加器。

假设一个 N 次循环体，其前 N/2 次循环和后 N/2 次循环不相关，即无论哪个先执行结果都一样，此时可以使用“将循环体视为两个并行执行的循环体”的方法来提高使用并行指令的机会。其具体做法是：先将 N 次循环体分为一个前 N/2 次的循环体和一个后 N/2 次的循环体，并且在这两个循环体中使用不同的累加器和辅助寄存器(ARx)，以减小两个循环的相关性。然后前后两个 N/2 次循环体的语句可以以任意顺序交织合并为一个 N/2 次循环体，选择合适的交织顺序使得能够合并的语句挨在一起。

例如例 2 所示的 N 次循环体：

例 2

```
STM  #(N-1), BRC  ;计数值放入 BRC 中
RPTBD loop2  ;粗体表示的语句循环 N 次
STM #table1, AR2; table1 指针到 AR2
    LD  *AR2,16, A   ;表内容加载到 A
    SFTA A     , 1   ;A 左移一位
    ADD  *AR2,16,  A ;加到 A
Loop2:
    STH A, *AR2+     ;A 存储到表中
```

从中我们看不出有能够合并的语句，且无法调整语句顺序。保证程序逻辑不变的情况下，将其分为一个前 N/2 次的循环体（称为 A 循环）和一个后 N/2 次的循环体（称为 B 循环），并且在这两个循环体中使用不同的累加器和辅助寄存器，即 A 循环使用 A 累加器和 AR2 辅助寄存器，B 循环使用 B 累加器和 AR3 辅助寄存器。分成前后两个循环体后的代码如下（A 循环用细体字表示，B 循环用粗体字表示）：

```
    STM       #(N/2-1),  BRC
    RPTBD  loop3
    STM    #table1,   AR2
        LD     *AR2,16,   A
        SFTA   A,         1
        ADD    *AR2,16,   A
Loop3:
        STH    A,        *AR2+

    STM    #(N/2-1),  BRC
    RPTBD  loop4
    STM    #table1+N/2,AR3
        LD     *AR3,16,   B
        SFTA   B,         1
        ADD    *AR3,16,   B
Loop4:
        STH    B,     *AR3+
```

从程序逻辑关系可知 A 循环和 B 循环不相关。现在将 A、B 循环体合并为一个 N/2 次循环体，由于 A、B 循环不相关，所以合并时在保证 A、B 循环体自身语句顺序不变的情况下，A、B 循环体之间语句可以以任意顺序交织，将能够合并为并行指令的语句挨在一起。A、B 循环体交织以后的结果如下（粗体为来自 B 循环的语句）：

```
STM       #(N/2-1),  BRC
STM       #table1,   AR2
RPTBD     loop5
STM       #table1+N/2,  AR3
    LD        *AR2,16,   A;第 5 行
    SFTA      A,         1;第 6 行
    ADD       *AR2,16,   A
    LD        *AR3,16,   B;第 8 行
    STH       A,     *AR2+;第 9 行
    SFTA      B,         1
    ADD       *AR3,16,   B
Loop5:
    STH       B,     *AR3+;第 13 行
```

此时，程序的第 8 行和第 9 行可以合并为 ST||LD 并行指令。

再使用 “部分循环展开法”将上例中第 6 至 13 行所有语句和第 5 行调换顺序，

调换后最后两条指令能够合并为并行指令。

至此我们用两种调整语序的方法分别产生一个合并的机会，共节省约 N 个指令周期，其中 N 是循环次数。

对于任何前 N/2 次循环和后 N/2 次循环不相关的循环体可以尝试用该方法增加使用并行指令的几率。

3 合理利用指针增减的思想

这里指针是指进行间接寻址的辅助寄存器[6]（ARx）。54xDSP 汇编中专门用于增减 ARx 的指令为 MAR[7]指令，但是任何包含通过 ARx 间接寻址的指令都能够附带地改变 ARx 的值，如果能够合理利用这一特性能够减少指令周期数和节省 ARx 的使用个数。合理利用指针增减的关键点是，在当前指令使用 ARx 的同时修改 ARx 的值为下面的指令使用 ARx 做好准备。下面用例 3 来说明这种思想的应用思路：

例 3

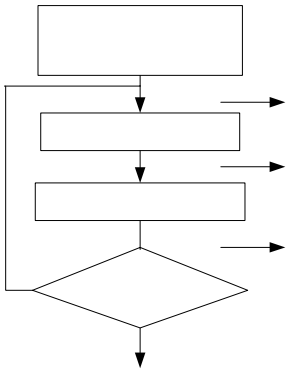
```
while(num--){
 for(j=0;j<N;j++ )
   Acc0=L_mac(Acc0,t1[j],t[j]);
 for(j=0;j<N;j++ )
   Acc0=L_msu(Acc0,t2[j],t[j]);
 t[0] = round( Acc0 ) ;
}
```

使用合理指针增减以后的汇编代码如下：

```
STM  #(-(N-1)),AR0; 设置 AR0 的值
Loop6: ;以下所有指令循环，构成 C 循环体
    RPTZ A, #(N-1-1);下一条指令循环
        MAC  *AR3+,*AR2+, A;D 循环体
    MAC  *AR3+0%, *AR2, A;AR3+AR0

    MAS  *AR2-, *AR4, A ;乘减
    RPTZ B, #(N-1-1);下一条指令循环
        MACD *AR2-,#t2,B; E 循环体
    SUB     B,  A ;相减
    STH     A,  *+AR2; 存储到 AR2
    BANZ    loop6, *AR1-;跳到 loop6
```

例 3 可以用图 4 表示其流程图，例 3 整体是一个大循环（称之为 C 循环），C 循环内含有两个内循环（依次称为 D 循环和 E 循环）。这里主要注意 AR2 的使用技巧，程序中用 AR2 指向表 t，在 D 循环中对表 t 中每个单元做 MAC 操作，在 E 循环中做 MACD 操作。通常的做法是在进入 D 循环和 E 循环前给 AR2 赋初值为表 t 的开头，然后在循环里递增 AR2 来遍历每个单元，但是如果我们假设在进入 D 循环前 AR2 指向表 t 的开头，在 D 循环内用递增 AR2 遍历每个单元，D 循环结束时 AR2 指向表 t 的末尾，在 E 循环中改用递减 AR2 来遍历（同时事先需改变表 t2 的存储序顺序），这样 E 循环结束后并重新回到 D 循环时 AR2 正好又指向表 t 的开头，于是就不必在 D、E 循环前赋初值，只用在进入 C 循环前初始化即可（见图 4）。另外，为了使得 D 循环执行



后 AR2 指向表 t 末尾，这里将 D 循环的最后一次循环独立出来，并且不使用*AR2+寻址而使用*AR2 寻址。总之“合理利用指针增减的思想”使得前面的代码执行的同时为后面的代码的执行做好了指针初始化的工作，节省了额外进行指针初始化的操作。

4 AR0 作为循环次数法

对于内层循环次数随着外层循环递增或者递减的二重循环，可用“AR0 作为循环次数法”提高其效率。这种循环结构在卷积等多种 DSP 常用算法中经常出现。以如下例 4 代码为例：

例 4

```
for(i= N-1;i>=1;i--){
   for(j=0;j<=i;j++)
Acc0=L_mac(Acc0,t3[j],t4[j]);
}
```

该程序由外层循环和内层循环构成，内层循环次数随着外循环递减。例 4 采用 AR0 作为循环次数法时的汇编代码如下：

```
STM  #(N-2), BRC    ; 外层循环次数
STM  #(t3+N-1), AR3; 指向表 t3
STM  #(t4+N-1), AR4; 指向表 t4
RPTBD loop7;
STM  #(N-2), AR0    ;AR0 初始化
    RPT  *(AR0) ;AR0 作为循环次数
        MAC  *AR3-,*AR4-,A ;内循环体
    MAC  *AR3+0%,*AR4+0%,A;改变指针
Loop7:
    MAR     *AR0- ;内层循环次数递减
```

这里用 AR0 作为内循环次数有两个好处：①内循环次数正好是内循环执行后指针(AR3、AR4)的改变量，这样语句 MAC *AR3+0%,*AR4+0%,A 执行乘加操作的同时，可以将 AR0 加到 AR3 和 AR4 上，使指针恢复为原值（也就是下次内循环开始时需要的值），避免了在内层循环开始时需要初始化指针的操作。AR0 起了双重作用。②使用 AR0 作为内循环次数时，当内循环次数递减时可以用 MAR　*AR0-指令，它是单周期指令，而相比之下若采用 ADDM #(-1), Smem[8]指令来递减则需两个周期。

5 结语

DSP 汇编程序编写时，需要选择不同的指令组合来实现程序功能，它的组合形式是多样的，各种组合的效率也可能有明显的差别。于是对于特定的程序结构，有一套较优的实现方式，本研究的目的是找到这些实现方式供编程时参考。其它的程序结构，例如条件分支结构、码本搜索结构由于篇幅所限将在以后推出。本文只是抛砖引玉，针对特定程序结构的汇编优化方案的寻找还需要不断补充、完善，该研究内容不仅对 54xDSP 且对于其它 MCU 也是一个很实用的研究内容。

参考文献

[1] TMS320C54x DSP Functional Overview [M]. Texas Instruments Incorporated, 2001.

[2] TMS320C54x DSP Reference Set Volume 2: Mnemonic Instruction Set[M]. Texas Instruments Incorporated, 2001.p235

[3] TMS320C54x Assembly Language Tools User's Guide[M]. Texas Instruments Incorporated, 2001.

[4] TMS320C54x DSP Reference Set Volume 2: Mnemonic Instruction Set[M]. Texas Instruments Incorporated, 2001, p290

[5] TMS320C54x DSP Reference Set Volume 2: Mnemonic Instruction Set[M]. Texas Instruments Incorporated, 2001. p291-300

[6] TMS320C54x DSP Reference Set Volume 1: CPU and Peripherals [M]. Texas Instruments Incorporated, 2001. p128

[7] TMS320C54x DSP Reference Set Volume 2: Mnemonic Instruction Set[M]. Texas Instruments Incorporated, 2001. p206

[8] TMS320C54x DSP Reference Set Volume 1: CPU and Peripherals [M]. Texas Instruments Incorporated, 2001. p125