

单片机 C 语言交叉编译器存贮分配研究*

侯济恭 余 坚 李铮铮

(华侨大学计算机科学系, 泉州 362011)

摘要 介绍单片机存贮结构的特点和 C 语言编译器的存贮分配原理, 提出单片机交叉 C 语言编译器的存贮分配方法, 并给出一个实用存贮分配算法。

关键词 编译器, 存贮分配, 单片机

分类号 TP 314

将 C 语言(下称为 C)引入单片机的应用领域, 利用 C 的“高级汇编”优势, 必然会缩短单片机应用系统的开发周期, 降低其开发费用。在移植和生成单片机交叉 C 编译器时, 由于单片机的存贮结构与微、小型机差异甚大。因此, 存贮分配便成为首先必须解决的问题。为此, 本文首先介绍 8031 单片机的存贮结构及其寻址特点, 以及 VAX-11 机 C 编译器存贮分配原理, 然后提出单片机交叉 C 编译器的存贮分配设计方案及其实现模型。

1 单片机存贮结构与寻址方式简析

8031 单片机存贮空间, 分为内部 RAM, 外部 RAM 和外部程序区。各区功能分叙如次。

1.1 内部 RAM 区

地址空间为 00~FFH, 其中 80~FFH 为特殊寄存器区, 如累加器 A、寄存器 B 和栈指针等。一般而言, 此区不作为一般的数据存取区。00~7F 区为片内工作区, 该区又分为: (1) 寄存器组区 00~1F。每 8 个对应 1 组寄存器 R0~R7, 共 4 组, 见图 1。工作寄存器组可用 R0~R7 寻址, 也可用直接地址寻址。寄存器 R0, R1 还可作为间址寄存器; (2) 位寻址空间。20~2FH 为寻址空间(也可字节寻址); (3) 字节寻址。30~7FH 为仅可字节寻址空间。对片内 RAM, 指令系统提供的寻址操作有下述四种。

(1) 立即数, 如: MOV A, #03; 3 送累加器 A。(2) 直接地址间操作, 如: MOV 6F, 3FH; 单元 3F 值送 6F。(3) 间址操作, 如: MOV @R0, #07; 值 7 送 R0 所指单元。(4) 位寻址操作, 位址从 00~7FH, 如: SETB 7FH; 位址 7F 置 1。各种寻址操作关系, 如图 2 所示(图中实

FF	SFR
7F	字节寻址
30	位寻址
20	寄存器组 3
18	寄存器组 2
10	寄存器组 1
08	寄存器 0
00	

图 1 片内 RAM 结构

* 本文 1993-10-18 收到; 福建省自然科学基金资助项目

线表示数据传送类,虚线表示运算类)。

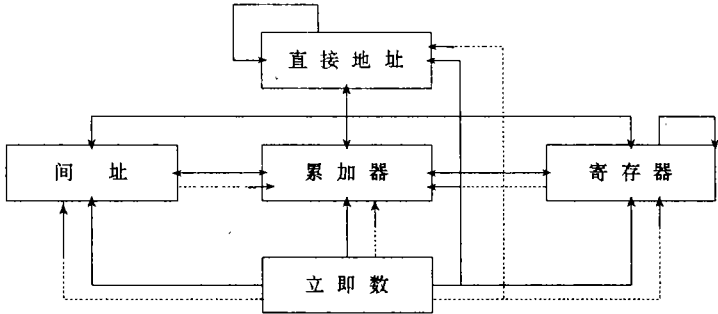


图2 指令寻址关系

1.2 外部 RAM

地址空间从 0000~FFFF,计 64kB,它有两种寻址方式。(1)直接 16 位寻址,可通过寄存器 DPTR 进行,如:MOVX A, @DPTR;读外部数据。(2)地址空间 0000~00FFH,可通过间址寻址,如:MOVX @R0, A。

1.3 外部程序区

只有 1 条变址寻址指令,如:MOVX A, @A+PC ;(PC+A) 送入 A。
可见,片内 RAM 区操作极其丰富,它具有寄存器功能、寻址多样、存取灵活、存取速度快于外部 RAM 等优点。单片机的主要应用领域是工业控制,其执行速度至关重要。因此,单片机 C 编译器首先必须解决存贮空间分配问题,其核心则是如何充分利用片内 RAM 区。

2 C 语言编译器存贮分配机制

C 语言存贮分配采用栈式动态分配和静态分配^[1]。栈式动态分配是为每一个运行中的函数建立一个栈帧,它包含四部分:实参区,寄存器保护区,链接数据区和局部变量区。图 3 是 VAX-11 机 C 语言的栈帧结构^[2]。图中,sp 为栈(顶)指针,寄存器 fp 为自动变量基址,寄存器 ap 为实参基址。对自动变量的访问,形如(左为 C 语言语句,右为相应的目标代码)

x=3; movl \$3, -4(fp);

对实参的访问则为

a=6; movl \$6, 4(ap);

栈帧的生成由编译器、目标指令和硬件共同完成。实参区由函数调用前的进栈指令生成,寄存器保护区和链接数据区由硬件自动完成,自动变量区则由编译计算而成。
函数返回时,必须作恢复环境的工作。即将链接数据区的数据返填入原寄存器,把 sp 指针调至调用者函数栈帧顶部,然后执行返回(RET)指令。

3 单片机 C 编译器存贮分配设计

由上述可知,单片机片内 RAM 太小,不可能把栈帧放在此处。如果把栈帧放在外部

RAM 中,则外部 RAM 寻址功能太差,系统的开销也太大. 因此,需要一个可行的方案:分裂栈帧,使处于不同 RAM 区. 为方便讨论,先定义有关术语:(1)工作帧,是当前正在执行的函数之栈帧;(2)帧库,是活跃函数之栈帧的有序集合. 考虑单片机内 RAM 区具有寄存器功能,本 C 编译器可略去 C 中寄存器变量. 因此,栈帧由自动变量区、实参区和链接数据区这三个部分组成(图 4):(1)自动变量区.

由内部 RAM 承担,由 7FH 向下分配;(2)实参区. 由内部 RAM 承担,由 1F 向下增长;(3)链接数据区. 置于外部 RAM 中. 链接数据区包括:(a)老 R0,为调用者函数自动变量区指针;(b)老 R1,为调用者函数实参区指针;(c)老 PSW,为调用者函数程序状态字.(d)老 PC,为调用者函数返回地址. 由于片内 RAM 区 20H~2FH 可位寻址,非常宝贵,故此区域作为外部变量分配区.

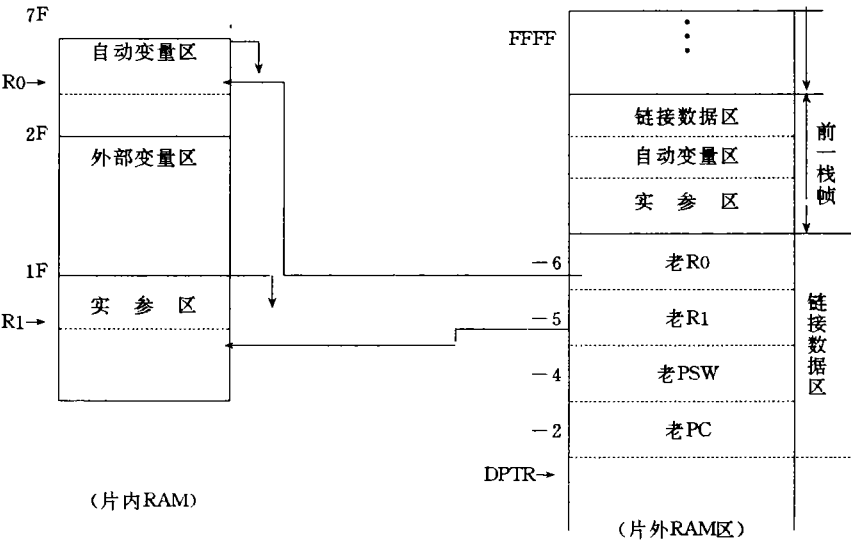


图5 工作帧结构

3.1 工作帧生成

工作帧的物理结构,如图 5 所示. 工作帧生成分两步完成. 第一步,在翻译函数调用 $f(x,y,z)$ 时,执行如下算法:(1)将 7FH~R0 单元的自动变量移入栈帧库;(2)将 1FH~R1 单元的实参区移入栈帧库;(3)保护 R0,R1;(4)把实参传入 1F 始单元,生成新 R1;(5)调函数 f .

这些算法,保护老工作帧于帧库,然后生成新函数的部份工作帧. 第二步,进入被调用函数 f ,生成函数头码及其余链接区. 其算法:(1)将老 $R0,R1$ 送外部 RAM 的帧库;(2)PSW 送帧库;(3)PC(返回地址)送帧库;(4)根据自动变量总额 L ,产生自动变量区 $7F-L \Rightarrow R0$. 至此,整个工作帧生成完毕(图 5).

3.2 工作帧的恢复

当函数执行完毕,在返回调用者之前必须恢复工作帧. 其算法:(1)保护返回值;(2)计算实参区长度 $L1$;为 $L1=1FH-老 R0$;(3)将 DPTR-8 始 $L1$ 个单元移入片内 RAM $1FH$ 始区域(恢复实参区);(4)计算自动变量区长为 $L2=7FH-老 R1$;(5)将 DPTR-8- $L1$ 个单元移入片内 RAM $7FH$ 始区域(恢复自动变量区);(6)调整栈帧库指针,使指向当前工作帧链接数据区,即 $DPTR=DPTR-L1-L2-6$;(7)恢复返回值 $R0,R1,PSW,PC$;(8)返回.

4 自动变量与形式参数的翻译

自动变量区和形式参数区确定在内部 RAM 中,自动变量区以 $7F$ 为始点,形式参数以 $1F$ 为始点,逐次向下分配(表 1). 参数 x,y,z 和自动量 a,b,c 存贮空间分配,如表 2 所示.

表 1 C 程序与目标代码

C 程序	目 标 代 码
$f(x,y,z)$.
char x,y,z ;	.
char a,b,c ;	.
$a=3$;	MOV 7FH, #03H
$b=4$;	MOV 7FH, #04H
$c=x+y$;	MOV A, 1FH
⋮	ADD A, 1EH, $A=x+y$
}	MOV 7DH, A ; $c=A$

表 2 参数与自动量存贮空间分配

变 量	地 址
x	1F
y	1E
z	1D
a	7F
b	7E
c	7D

根据上面分析,修改 C 编译器的自动变量分配算法(自动变量分配指示器 $autooff=80H$)如次:(1)每进入一层分程序, $autooff$ 进栈;(2)每识别 1 个自动变量,根据其类型确定其所需的空间大小 tse , $autooff=autooff-tse$;(3) $autooff$ 即为新变量的存贮地址;(4)退出一层分程序,执行 $minoff=\min(minoff, autooff)$. 即记住当前所需的最小空间地址(最大占用空间), $minoff$ 的初值由系统设定,恢复上层 $autooff$;(5)计算自动变量所需空间,生成自动变量区指针 $R0$. 即 MOV $R0, minoff$;(6)结束返回 $autooff$.

5 实参区生成

在翻译函数调用语句 $f(x,y,z)$ 时,于完成老工作帧的保护后,用以下指令便可完成实参区的生成. 为方便讨论,令 x,y,z 均 char 型,则其存贮地址分别为 $7F,7E,7D$. 即

MOV R1, #1FH; 实参区首址 1F 送 R1
MOV @R1, 7FH; 传 x 值至实参, 间址操作
DEC R1
MOV @R1, 7EH; 传 y 值
DEC R1
MOV @R1, 7DH; 传 z 值
LCALL $-f$; 走调用函数 f
运行结果, 实参区如图 6 所示.

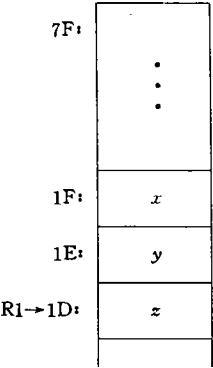


图6 实参区

6 结 束 语

本文提出工作帧的概念, 它在物理上将一个函数的栈帧分为三部分, 并分配于单片机不同的 RAM 区, 是本存贮分配方法的特色. 同时, 该存贮分配方法还充分利用单片机存贮结构特点, 有助于提高目标程序的运行效率, 并使自动变量的存贮分配变得十分简单.

参 考 文 献

1 侯济恭. C 语言编译器结构分析. 小型微型计算机系统, 1992, 4: 60~61
2 王博文, 侯济恭. VAX-11 机 C 编译器分析报告与移植概要. 计算机工程与应用, 1990, 2: 11~13

Storage Allocation of C Language Cross Compiler on
A Single Chip Microprocessor

Hou Jigong Yu Jian Li Zhengzheng

(Dept. of Computer Science, Huaqiao Univ., 362011, Quanzhou)

Abstract The authors deal with storage allocation of C language compiler. The peculiarity of storage structure of a single chip processor and the storage allocation principle of C language compiler are brieded. A storage allocation method of C language cross compiler implemented on a single chip microprocessor is proposed. A practical algorithm for storage allocation on a single chip microprocessor is given finally.

Keywords compiler, storage allocation, single chip microprocessor