システム記述のためのC言語の仕様拡張の検討

3D-9

細田泰弘　遠藤　真　星野民夫　　ＮＴＴ電気通信研究所

１．　はじめに

我々はこれまでメインフレーム上に様々なＤＡプログラムを開発してきたが、最近グラフィックや会話処理の比重が高まり、きめ細かい処理を行うためにアセンブラを利用する機会が増えた。

プログラムの生産性、保守性、拡張性を考えた場合、システム全体を高級言語で記述できることが望ましい。Ｃ言語はアドレスが見えるなどシステム記述用言語として優れているが、機械に依存した部分やプログラムの中核で特に高速に実行する必要がある部分についてはアセンブラを使用する必要がある。

ここではシステム全体を統一的にＣ言語で記述するために、言語仕様の拡張について検討した。

２．　拡張点

主な拡張点は次の通りである。

アセンブラを使用せずに機械に依存した部分を記述するために(1)ある特定のレジスタをＣの変数に割り付けることができるようにする。(2)特殊な機械語命令を発行するための関数を用意する。(3)関数呼出しのプロローグ／エピローグ処理をユーザ定義により変更可能にする。

アセンブラに匹敵する実行速度を得る最適化を可能とするために(4)文字列操作等、幾つかの関数をアセンブラにインライン展開する。(5)コンパイラの最適化を前提とした新たな型を追加する。

さらに(6)機械語命令やアセンブラ命令を直接記述する必要がある場合，#asm,#endasm文で括って記述することを可能とする。

次に各々の拡張点について説明する。

２．１　レジスタ割り付け

レジスタがプログラムから直接参照できることは、機械語とのインタフェースを考えた場合必須である。本仕様では次の構文を採用する。

標準Ｃとの互換性のため、レジスタ番号は#pragma文により指定する。

構文	意味
#pragma register(n)<name>	レジスタnを変数<name>で使用する
#pragma restrict(n)	コンパイラに対してレジスタnの使用を制限する
#pragma release(n)	レジスタnを解放する。

例

```
#pragma register(5) i     (レジスタの５番を
 foo(){                    iに割り付ける)
  register int i;
#pragma restrict(5)       (レジスタの５番の
        for (i=0;;i++)    使用を制限する)
          .....
#pragma release(5)        (レジスタの５番を
  }                        解放する)
```

２．２　特殊な機械語命令の発行

特権命令を使う場合など、ある命令を明示的に発行する必要がある場合、これを関数呼び出しを用いて実現する。

例えば、３７０のSIO(Start I/O)命令は

```
SIO(10);
```

という形で発行する。本仕様ではSIOをはじめとするいくつかの関数名は予約語とする。これらの関数名はANSI標準規格案[1]と同様に標準ヘッダファイルで次のように定義する

```
#define  a(int b)  _builtin_a(int b)
```

これらの関数名をユーザが使いたいときは

```
#undef   a(int b)
```

とする。

２．３　関数のプロローグ／エピローグ処理

リンケージ規約の異なる他言語とのリンクを容易にするため、関数のプロローグ／エピローグ処理をユーザが変更できるようにする。

#pragma prologue <name>	プロローグ処理用アセンブラマクロ名
#pragma epilogue <name>	エピローグ処理用アセンブラマクロ名

これらの指定がない場合はコンパイラの用意するリンケージ規約を使用する。

A Proposal of C Language Extension for System Development

Yasuhiro Hosoda, Makoto Endo and Tamio Hoshino

NTT Electrical Communications Laboratories

２．４　文字列関数

文字列の比較、複写等の操作はＤＡプログラムの中でかなりの比重をしめており、高速で実行することが望まれる。メインフレームの代表である３７０系のマシンをはじめとして多くのマシンは文字列処理命令を持っており、これを活用してアセンブラにインライン展開すれば最適化が図れる。

例えば、文字列s1に文字列s2をn文字分コピーする関数

```
strncpy (char *s1,char *s2,int n)
```

は、s1,s2がそれぞれレジスタＲ１，Ｒ２に割り付けられておりnが定数であるとすると、３７０のMVC命令を用いて

```
MVC  0(n,R1),0(R2)
```

の1命令で済ますことができる。これらの関数(命令)名は機械語命令発生のための関数と同様に、予約語とし、最適化を行う。

２．５　型の追加

コンパイラによる最適化を前提としたconst型とvolatile型を追加する。

読みだし専用の変数を定義するためにconst型を設ける。

例　static char const s[]="abc";

コンパイラによる検査を強化することにより、不用意な変数の書き換えを防ぐことができる。

コンパイラの最適化により副作用が生じる可能性がある場合、volatile型を使用する。

例　char const volatile s[]="abc";

この指定により配列sは確実にメモリ部に割りつけられ、最適化によりレジスタに割りつけられる可能性を排除することができる。

２．６　機械語命令の直接記述

他マシンに対するポータビリティのためアセンブラ命令や機械語命令を#asm,#endasmで括って記述できるようにする。

```
foo(){
 ......
 #asm
  WTO 'JOB ENDED'
* MSG OUTPUT TO CONSOLE
 .....
 #endasm }
```

３．適用例

本仕様を既存のアセンブラによるモジュール（仮想カードリーダからテキスト入力を行う）に適用した例を示す。

アセンブラ記述

```
START    BALR  15,0
         USING *,15
         LA    1,BUFFER
         MVI   BUFFER,X'00'
         LA    3,READCCW
         ST    3,CAW
         LH    2,=X'000C'
LL1      SIO   0(2)
L3       TIO   0(2)
         BE    L4
         CLI   CSW+4,X'02'
         BNE   L3
         MVI   DEAD+7,X'FF'
L6       LPSW  DEAD
L4       CLC   BUFFER+1(3),=C'TXT'
         BE    L5
         ........
READCCW  CCW   X'02',BUFFER,X'00',80
BUFFER   DS    CL80
DEAD     DC    X'0002000000000000'
CSW      EQU   64
CAW      EQU   72
         LTORG
         END
```

C記述

```
#pragma register r0 (0)
  ....
#pragma register r3 (1)
int readccw;
char buffer[80];
int dead=0x2000000000000,caw=72,csw=64;
start(){
   register int r0,r1,r3;
   register short r2;
#pragma restrict(0)
  ....
#pragma restrict(3)
   r1=(int)&buffer;
   buffer[0]=0;
   r3=(int)&readccw;
   caw=r3;
   r0=0x0c;
   sio(2+r0);
   if (tio(2+r0) != 0) {
      for(;*(&csw+1)!=0x02;);
      *((char *)&dead+7)=0xff;
      lpsw(dead);
      clc(buffer+r1,3,"text");
      ..........
   }
}
```

４．まとめ

これはメインフレーム系を念頭に置いた仕様であるが、それ以外のマシンにも幅広く適用可能であり、またシステムからアプリケーションに至るまで使用可能であると考えられる。

５．参考文献

1) "C Standard Draft", ANSI, Nov. 1985