Ｌｉｓｐ０５　コンパイラの設計

1D-1

井田昌之　(青山学院大学)

１．はじめに ――― 試作の目的と概要 ―――

8086系用のLisp処理系であるLisp05[1][2]に対してコンパイラを設計し、その初版を作成したので報告する。Lisp05は主記憶256KB以上のCP/M-86上で動作する。

試作の目的は、86系パソコンのハードウェアでの実用規模と速度に対する指標を得ることである。速度的には筆者らの以前のシステムであるALPS/Iの100倍が目標である。インタプリタでの速度はALPS/Iの約16倍で、INTERLISP(PDP10)に匹敵する値もでている[2]。規模の指標となるデータはReduce2.0の実行データ(ALPS/I)及び、その後入手したReduce3.0のソースをもととし、100kセル程度のLISPプログラムコードと数十kセルの作業域を想定する。この想定が8086系でのLISPの実質的な上限であるという仮説に基いて設計した。すなわち、ソフトによる仮想化はせず、16ビットのままで扱う。コンパイラは、速度的な要件としては勿論であるが、フリーストレージが32Kセルであり、プログラムコード格納のための必然性もある。

表1 レジスタの割付け

Reg.	用途
CS	CS0,CS1,CS2
DS	car セグメントに固定
ES	cdr セグメントに固定
SS	スタック用に固定
AX	値レジスタ
BX	第一引数
CX,DX	作業用
SI	第二引数
DI	第三引数
BP	フレームポインタ
SP	スタックポインタ

２．メモリの構成とレジスタの割付け

表1にレジスタの割付けを示す。CS0はCP/Mコード(32k),CS1はLISP05システム(64k),CS2はコンパイルコード(最小16kb,640kbまで可)に対する領域である。DS,ESは各64kbを固定使用する。SSは現在の所、CS1とオーバーラップしている。256kb構成でも、DDTやLISP05内部からのcpmコマンド[2]実行用の領域として、16KBを残す。BPによりフレームを形成する。

３．ＬＩＳＰコンパイラの基本的な構造

コンパイラは次のような特徴を持つ。

(１) 完全な機械語コードを生成する3パスコンパイラ

pass 1：中間コードへのコンパイル(基本関数のいくつかは展開する)

pass 2：中間コードのオプティマイズ

pass 3：8086機械語命令の生成

(２) コンパイルコードは、lexical bindを基本とする。

(３) 引数はレジスタに、ローカル変数(ラムダ変数、プログ変数)はフレームスタックに、グローバル変数は素情報領域におく。

(４) 他の関数とはInter-segment callを経由して行き来する。[1]ではシステムコールはプログラム割込み経由としていたが、52clockと遅いので設計を変更し20clockで済むcallfで統一した。(システム内ではIntra-callを使用)

(５) コンパイルコードスペース(CS2)に対してはメモリの余裕に応じて独立したセグメントを割りつける。CS2の大きさは、64Kを越えてよい。

3．中間コード命令セット

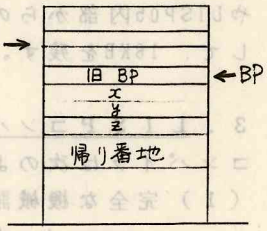
中間コードは仮想アーキテクチャではなく8086のアーキテクチャに基づくものとした。pass2 は現在、スタックフレーム処理などの最適化を行なっているが、[3][4]などを参考に改良中である。現在のLisp05コンパイラの中間コードは次の通り

1	(ENTRY 名前) (ALLOC n)	sub sp,n ! push bp ! mov bp,sp
2	(LABEL ラベル)	ローカルラベル
3	(MOVFG レジスタ グローバル変数)	グローバル変数⇒レジスタ
4	(MOVTG グローバル変数 レジスタ)	グローバル変数⇐レジスタ
5	(MOVR レジスタ AX)	MOV レジスタ, AX
6	(MOVI レジスタ 即値)	MOV レジスタ, 即値
7	(MOVFBP レジスタ オフセット値)	MOV レジスタ, offset[BP]
8	(MOVTBP オフセット値 レジスタ)	MOV offset[BP], レジスタ
9	(JMP ラベル)	
10	(JZ ラベル)	zero flag-onで分岐
11	(JNZ ラベル)	zero flag-off で分岐
12	(ATOMJ ラベル)	shl bx,1 ! jnc label
13	(NATOMJ ラベル)	
14	(PUSH レジスタ)	
15	(POP レジスタ)	
16	(CAR)	shl bx,1 ! mov ax,[bx]
17	(CDR)	shl bx,1 ! mov ax,es:[bx]
18	(SYSCALL 名前)	mov bx,name ! callf syscall
19	(FARCALL 名前)	callf usrcseg:name
20	(DALLOC n) (RET)	pop bp ! add sp,n ! retf
21	(NULLAX ラベル)	cmp ax,nil ! jz label

4．例

コンパイルの概略を次に示す。

```
func:(LAMBDA (x) (prog (y z)...(setq z x)...))
                    ⇩   pass1
((ENTRY func)(ALLOC 2)(MOVTBP 2,AX)(ALLOC 4)...
  (MOVFBP AX,2)(MOVTBP 6,AX)...(DALLOC 4)(DALLOC 2)(RET))
                    ⇩   pass2
((ENTRY FUNC)(ALLOC 6)(MOVTBP 2,AX)... ...(DALLOC 6)(RET))
                    ⇩   pass3
sub sp,6! push bp! mov bp,sp! mov 2[bp],ax! ... ! pop bp! add sp,6! retf
```



5．おわりに

初版における fib(20) の実行時間は1.9sec（コンパイラ）及び、9.25sec(インタプリタ）である。（同一アルゴリズムをハンドコードで実現した場合、1.1sec．であった）今後も最適化は改良が必要である。しかし、コンパイラの具備により、比較的大きなLISPプログラムの実行への道を開くことができた。速度的には、ALPS/Iの80倍程度までの向上が実現できた。

参考文献 (1)井田：8086系試作Lispの0Sプリミティブについて；情処27回大会 pp1333-1334, 1983 (2) 井田：8086用試作Lispの設計とその機能；記号処理研究会27-2 1984
(3) M.Griss & A.C.Hearn : A Portable LISP Compiler, soft.prac.& exp.11 pp541- 1981
(4) L.M.Masinter and L.P. Deutsch : Local Optimizaion in a Compiler for stack-based Lisp Machine, Xerox PARC SSL-80-4 pp11-23 1980