2E-2

Common Lisp サブセットの試作

佐治 信之
日本電気 (株) ﾏｲｸﾛｺﾝﾋﾟｭｰﾀｿﾌﾄｳｪｱ開発本部

1. はじめに

近年のＬＩＳＰの応用範囲の拡大に伴って、ＬＩＳＰの使える環境の整備及び拡大が求められている。最近のマイクロプロセッサの発達によって、パーソナルユースにおいても３２ビットのプロセッサが使われるようになることが予想され、より広い分野においてＬＩＳＰの使用が今後本格化してゆくと思われる。

当社では今年３２ビットの汎用マイクロプロセッサとしてＶ６０を発表した[1]。Ｖ６０は、広いセル空間と十分な実行速度のＬＩＳＰを提供することが可能なプロセッサであり、Ｖ６０上にＬＩＳＰを実現することでＬＩＳＰ分野のニーズに効果的に対応することができると思われる。またＶ６０は３２本の汎用レジスタ、豊富なアドレッシングモードを備えており、効率的なリスト処理が可能なアーキテクチャでもある。そこでＶ６０をターゲットとするＬＩＳＰ処理系を試作することにした。この試作ではＬＩＳＰの関数を網羅することよりも、自己拡張が容易なカーネルを提供することを第一に考えた。

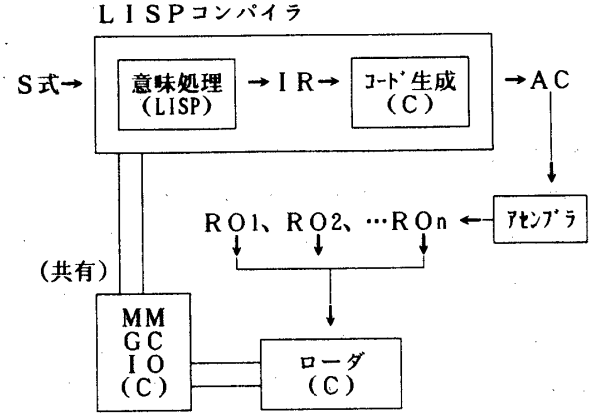
本論文においては、このＬＩＳＰシステムとその中核的存在であるコンパイラの構成法について考察する。

2. 設計方針

このＬＩＳＰシステム及びコンパイラを、以下の方針に基づいて設計することとした。

1) ＬＩＳＰの言語仕様としては世界的な標準を目指して検討されている Common Lisp を採用する。しかし Common Lisp の仕様はいまだ揺れているという状況であり、ここではこの Common Lisp の言語仕様書と言える Common Lisp the Language [2] (以下ＣＬｔＬと略す) をそのまま実現しようとはせず、Common Lisp のエッセンスともいえる、

- スコープとエクステントのルール



ＬＩＳＰシステム

ＩＲ … 中間表現、ポータブルＣコンパイラに準拠
ＡＣ … アセンブラソースコード
ＲＯ … リロケータブルオブジェクト
ＭＭ … メモリ（データオブジェクト）管理
ＧＣ … ガーベージコレクタ
ＩＯ … 入出力管理（ＬＩＳＰリーダを含む）

図 ＬＩＳＰシステムの構成

- スペシャルフォーム
- データ型階層

を極力保存した、小さなサブセットを想定する。

2) ＬＩＳＰシステム及びコンパイラはリターゲッタブルな構成とする。すなわちコンパイラのコード生成部及びローダは容易に置き換えができる形にする。

3) まずこのＬＩＳＰシステム及びコンパイラを既存のミニコンあるいはワークステーションの環境を利用して作成する。構成法が確立された時点で、コード生成部とローダを置き換えて、Ｖ６０のＬＩＳＰとする。

4) 他言語とのインターフェースを十分考慮して作成する。特にＣとは相互呼び出しが容易な仕様にする。

5) データオブジェクトの管理、ガーベージコレクション、入出力等はＣで記述する。つまり、低レベル操作はＣのライブラリに依存する。

6) コンパイラ自身の記述は、ＬＩＳＰでもＣでも良い。ただし今回は、他の目的で開発したＣで記述された簡単なＬＩＳＰインタプリタ（５Ｋステップ）が流用できるので、これを用いる。

An Experimental Implementation of a Common Lisp Subset.
Nobuyuki SAJI
NEC Corporation, Microcomputer Software Development Lab.

7）その他の関数は、できあがったＬＩＳＰコンパイラを使用して、ＬＩＳＰで記述する。

このＬＩＳＰシステム及びコンパイラの全体構成を前頁の図に示した。ＬＩＳＰコンパイラはＳ式をアセンブリ言語にコンパイルし、それをそのマシン上のアセンブラでアセンブルし、ＬＩＳＰシステムがこれをロード、実行する。メモリ管理、入出力等の基本ルーチンは図のように共有する。コード生成部については次節で述べる。

3．リターゲッタブルコンパイラ

既にＣコンパイラに関しては、コード生成部をターゲットマシンの仕様から自動生成するツールであるＣＯＯ（compiler object code generator generator）を開発し、実際のＣコンパイラ開発に適用している［3、4］。ここではこのＣＯＯを用いて作成したＶ６０用のＣコンパイラのコード生成部を流用することを考えた。設計方針にあるように、他のマシンにおける前段階試作においても既に作成済の各々のコード生成部を使用すればよい。

しかしながら、これらのＣ用のコード生成部はそのままの形では流用できない。それは以下の点に起因する。

a）関数呼び出しにおいて、ＬＩＳＰでは必ず間接呼び出しになる点でＣと異なる。

b）ＬＩＳＰにおいてはデータ型チェックのコードが必要である。

c）イミディエイトデータ及びポインタのワード内からの抜き出しと格納のためのコードが必要である。

そこで、「通常のプログラミング言語のコンパイラを作成する」という観点から見直し、これらの問題点に対処した。

a）この点は仕方がないが、コード生成部の中でこのための修正は数行で済みかつ極めて局所化されている。

b）型チェック無しのコードを出力する。関数エントリでのみ引数の型チェックを行ない、基本的に型チェックを省略する。チェックの必要な箇所には、明示的に型チェック関数を挿入しておく。つまり、このコンパイラはＬＩＳＰシステム記述者用であり、記述者はコンパイルコードについて熟知していることを前提とする。

c）イミディエイト及びポインタの格納方法としてワード全体を占有するようなデータ構造を与える。型情報は、そのワードの外側に保持する。これと b）により、Ｃなどによるコーディングと遜色のないオブジェクト生成を期待できる。

4．サブセット仕様

Common Lisp の言語仕様については、国内外で議論が盛んである。電子協のＬＩＳＰ技術専門委員会では、特に Common Lisp のサブセット試案作成の作業を行なってきた。これは Common Lisp のエッセンスを保ちつつ、現在あるいは近い将来のパーソナルコンピュータ上の標準ＬＩＳＰを目指したものである。またパーソナルコンピュータということで教育上の意義も大きいと考えられる。

ただし、今年の７月にまとまった第一次案［5］は、ＣＬｔＬの６００関数に対して３５０関数を有するもので依然として大きく、今回の試作の主旨には合わないため、採用は見合わせた。このＬＩＳＰシステムのカーネルが出来上った後、徐々にこの仕様に合わせてゆく予定である。その頃にはCommon Lisp の（フルセットの）仕様も収束していることを期待している。

現在検討しているサブセットは、以下のような仕様である。

- 型階層の概念の保持
- LEXICAL SCOPE の採用
- コンパイルで有効と思われる型宣言の保持
- 全般に関数の大幅削除（総数100～150程度）

5．おわりに

Common Lisp の試作について考察した。本システムはコンパイラ主導型のシステムであり、今回はコンパイラについてのみ述べた。インタプリタについては今後検討してゆく予定である。プログラミング支援環境を考えると、インタプリタ、デバッガ等は重要な要素であり除去するわけにはいかない。しかし、ＬＩＳＰを実行効率面で実用的なシステムならしめるには、コンパイラのより一層の洗練が必要不可欠であろう。

Ｖ６０のアーキテクチャを生かしたシステムの構成法についても機会を改めて述べることとする。

参考文献

［1］Ｖ６０アーキテクチャマニュアル、NEC、1986
［2］Common Lisp the Language、Digital Press、1984
［3］Code Generator Generator、情処ﾌﾟﾛｸﾞﾗﾐﾝｸﾞ言語研究会3-2、1985年12月
［4］v60ｱｰｷﾃｸﾁｬを生かした言語処理系、情処全国大会1C-1、1986年10月
［5］Common Lisp/Core 仕様、電子協LISP技術委資料、1986年7月