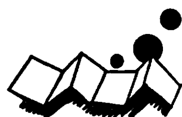
解 説 3. ハードウェアから見た命令セットアーキテクチャ



3.4 高水準言語計算機の命令セットアーキテクチャ†

中田 登志之†† 新 淳†† 中島 震††

1. はじめに

高水準言語計算機は，その命令セットアーキテクチャのセマンティックレベル[1]を高級言語に近いレベルにすることにより，言語の翻訳に要する時間を短縮したり，プログラムの実行効率を良くしようとしたものである[2].

高水準言語計算機の研究としては1960年代の初頭から Algol マシンや Cobol マシンなどの研究が行われてきた．これらの高水準言語計算機では，高級言語の構文や意味に対応した命令セットが定義されていた．このような命令セットを直接ハードウェアで実現するのは難しいため，マイクロプログラム技術が積極的に採用されてきた[3].

近年では以下に述べる理由から，高水準言語計算機の研究は Fortran やCなどの手続き型言語用の高水準言語計算機から，Prolog, LISP, Smalltalk などのAI用言語マシンに中心が移っているように思われる．

（1） 初期の高水準言語計算機の一つの目的であった，コンパイラの負担の軽減化よりも，目的コードの実行効率が重視されるようになってきた．

（2） もう一つの目的であったデバッギングの支援も，ワークステーションやパーソナルコンピュータ上のデバッギングツールが整備されるにつれ，ソフトウェア上の機能で十分実現されるようになってきた．

（3） コンパイラ技術（特に最適化技術）の進歩にともない，高級言語の各命令に近い命令セットよりも，高級言語の各命令の機能を実現する一連のプリミティブな命令セットを提供する方が，より実行効率の良いプログラムが得られるようになってきた．(RISC[4]はその顕著な例である．)

（4） 最近のマイクロプロセッサのアーキテクチャにおいてはスタック向き命令，配列データアクセス用のアドレッシングモードなど，高級言語処理に必要なプリミティブがすでに提供されている．

一方 AI 用の高級言語においては

（1） 動的タイプチェックやメモリ管理など，コンパイル時には決定できない処理の比率が大きい．

（2） インクリメンタル・コンパイル／リンクを可能にしながらかつ，関数呼び出しを効率よく実現しなければならない．

など実行時の負荷が大きく，それだけ，アーキテクチャによる処理の支援が期待できる．

本稿では AI 用の高水準言語（LISP, Prolog ならびに Smalltalk）計算機において，各言語を処理するのに必要な基本機構，それを実現するための命令セットの特徴について述べていくことにする．

なお紙面の都合上本解説では Multi-LISP マシンや EVLIS マシン，PIM などの並列言語計算機については触れず，逐次型の高水準言語計算機に解説を限る．

2. LISP マシンの命令セットアーキテクチャ

本章では AI 用高級言語の中では最も歴史の古い LISP マシンの命令セットに付いて解説する．

2.1 LISP マシンの命令セットアーキテクチャの特徴

当初 LISP はインタプリティブな言語として位置づけられていた．しかし AI の実用化が進み，大規模な LISP のプログラムが作成されるにつれ，プログラムの実行効率が重要視され，効率のよい LISP コンパイラが開発されるようになった．特に従来の LISP の標準化を目指した Common LISP[5] においては，lexical-scoping の採用，型の宣言など，コンパイラを重視した言語仕様がなされている．その反面 & rest, & key 引き数の扱いなど従来のスタックモデルでは

† Instruction Set Processor Architecture of High-Level Language Machines by Toshiyuki NAKATA, Atsushi ATARASHI and Shin NAKAJIMA (C&C Systems Research Laboratories, NEC Corporation).
†† 日本電気(株) C & C システム研究所

実現しにくい仕様となっている．

他方 LISP システムにおいては，開発環境が重視される．したがって LISP マシンは実行時のオーバヘッドを最小限に止めながらインクリメンタルコンパイル機能や，実行時のエラーチェック，デバッギングサポートを含む，統合的なプログラミング環境を実現しなければならない．

LISP マシンは，LISP 言語同様，他の AI 用マシンと比べるとその歴史は古い．また，対象としている LISP 言語の処理系の仕様も，処理効率を重視する MacLISP 系の言語や，実行環境を重視する Inter-LISP 系の言語では異なる．したがって，LISP マシンの命令セットアーキテクチャには後述する Prolog マシンの WAM や Smalltalk マシンのバイトコードといった標準的なモデルは存在しない．

ほとんどの LISP マシンの命令セットに共通する特徴を以下にあげる．

1) タグアーキテクチャならびにジェネリックな演算命令の採用

これらの特徴は LISP では，変数のタイプが実行時に初めて決まることに起因する．このためにデータのタイプはプログラムではなくて，データの方に保持させておく必要がある．また，逆にこの性質を利用して，実行時のエラーチェックを厳密に行い，システムの信頼性を向上することが可能となる．特に LISP によるプログラム開発は試行錯誤的な性格のものが多いため，実行時のエラーチェックは不可欠である．

このように動的にデータのタイプをチェックすることを効率よく行うために，LISP マシンではデータ（アドレス）の一部にデータの種類を示すビット（タグ・ビット）を装備することが標準的である．また，算術演算命令(加算など)などはオペランドとして，任意のデータタイプ（整数，浮動小数，BigNum など）を扱える必要がある．

2) CDR-コーディングの支援

LISP の基本的なデータ構造は，2進木を構成するための2個のポインタからなるセルである．CDR コーディングでは，リストの各要素が連続した領域に割り付けられた場合に，タグを用いて次のリストの要素がすぐ後ろにあることを示すことにより，必要なメモリ量を減らしている．この考え方はメモリが高価であった時点で盛んに用いられたが，現在でもページヒット率の向上などの目的で用いられている．

3) 効率の良い関数呼び出し命令ならびにスタックマシンの採用

LISP の計算モデルの基本機構は式の評価と，関数呼び出しである．LISP でのプログラミングスタイルの特徴として，関数呼び出しの多用があげられる．

関数呼び出し時にはスタック上に関数を適用するためのフレームが生成され，その後，その関数内での演算はそのスタックフレーム上で行われる．したがって効率のよい関数呼び出し機構と，スタック操作命令を命令セットで支援する必要がある．また，Common LISP では &rest 引き数や多値関数の扱いなど，従来の LISP 処理系よりも関数の扱いが強力になっている．その分関数呼び出し・復帰時のスタックフレームの操作の機構が複雑になっている．LISP マシンによってはこれらの処理も効率よく実現するような関数呼び出し命令を有するものも存在する．

4) システム関数の命令セットへの組込み

前述したように LISP では関数呼び出しが基本機構である．そのため，よく用いるシステム関数を機械命令として取り入れることが比較的容易である．システム関数を機械命令として，実現することにより，不要なスタック操作を除去するとともに，関数呼び出しのオーバヘッドを減らすことが可能となる．

この考え方を延長して，ユーザ定義関数までもマイクロプログラムにコンパイルして，機械命令として組み込むことを可能にした LISP マシンも存在する[6]．

5) Garbage Collection のサポート

LISP では明示的なデータ領域解放命令は存在しない．したがって定期的に不要にデータ領域を再利用のために集める（Garbage Collection : GC と呼ぶ）必要が生じる．GC を支援するためのデータ構造や命令を有する LISP マシンが存在する．

2.2 命令セットの実現方法

LISP マシンを命令セットの実現方法という観点から分類すると，以下の3種類のものに分類できる．

a） インタプリタ指向 LISP マシン

神戸大 LISP マシン[7]，Facom ALPHA（富士通）[8),9]，ELIS (NTT)[10),11] など

LISP のインタプリタをマイクロプログラムで記述する．（概念的には eval という関数を機械命令で有していると考えてもよい．）この種のマシンでは，LISP 関数の多数をマイクロプログラムで記述して，機械命令として有しているものが多い．一方ユーザの関数をすべてインタプリティブ形式（S式）でもっているのは

効率が悪い．そこで，このようなマシンにおいても，スタックマシンタイプの機械命令をコンパイラ用に用意している．

Facom ALPHA (富士通)[8),9)] では LISP インタプリタ，コピイーング GC，ならびに 107 種類もの標準関数をマイクロプログラムで記述し，機械命令として装備している．

一方コンパイラ用の命令として，スタックマシン用の命令を有している．特徴としてはマルチユーザ用の仮想スタックを用意していることがあげられる．コンパイラ用の命令の中では，特に関数呼び出しを効率よく実現するための，終端コール用命令や CATCH 命令などを装備している．

ELIS (NTT)[10),11)] は TAO という，LISP を基にして，オブジェクト指向言語や論理型言語のパラダイムを取り入れたマルチパラダイム言語を対象とした，LISP マシンである．特にインタプリタとコンパイラの両立をシステムの特徴としている．LISP インタプリタ，400 個もの組込み関数，ならびにスタックマシン用のバイトコード命令 (LAP コード) を装備している．大域脱出を効率よく実現するために，多段フレームたたみこみ用の LAP コードを用意している．

b） コンパイラ指向 CISC 型 LISP マシン

Symbolics 3600[12),13)], Ivory (Symbolics), Explorer, CLM[14)] (TI), FLATS[15)] (理化学研究所) など

スタック指向の命令セットをマイクロプログラムで解釈・実行する．特に CLM (Compact LISP Machine) や Ivory では高速化のために機械命令をパイプライン化して，マイクロ命令で実行する．GC のサポートのための命令やアドレッシング・モードを有する．

Symbolics 3600 (Symbolics)[12),13)] は MIT の CADR マシンの流れを汲む典型的な CISC 型の LISP マシンである．データは 1 語 36 ビットで，2 ビットの cdr コード，2 ビッドの主タグ，ならびに 32 ビットのデータあるいは 6 ビットの副タグと番地情報から構成される．命令は 17 ビット長で，9 ビットの命令コードと 8 ビットのオペランド部からなる．Symbolics 3600 では，関数呼び出しを必ず関数定義セルをとおして間接的に行うことにより，実行時に呼び出す関数が再定義された場合に対応する．また Symbolics の CALL 命令は，a）引き数の個数のチェック，b）& rest 引き数以外の引き数のコピー，c）& optional 引き数のデフォルト値の書き込みを行った後，制御を呼び出された関数に渡す．

FLATS (理化学研究所)[15)] は数式処理に重点をおいた LISP マシンである．命令はすべて 32 ビットで，8 ビットの命令コードと各々 8 ビットの 3 個のオペランド部からなる．

CALL 命令はフレームポインタを指定された値だけ，増加して，新たなフレームを直前のフレームと一部重複する位置に取り，この重複部分を引き数及び結果の引渡しに使用する．基本 LISP 関数，ハッシング操作，ビット操作，GC 用の命令など，230 種の機械命令を装備している．

c） コンパイラ指向 RISC 型 LISP マシン SPUR[16)] (UCB)

SPUR (Symbolic Processors Using Riscs)(UCB)[16)] は RISC II や SOAR の流れを組む LISP 用の RISC チップである．スタックの替わりにオーバラッピング・レジスタウィンドウを用い，Load/Store 命令以外のオペランドとしてはレジスタしか指定しない．データは 8 ビットのタグと 32 ビットのデータ部の合計 40 ビットからなり，タグ付き・無しの load/store 命令を有する．

SPUR ではジェネリック演算を機械命令で実現すオペランドのタイプが最も頻度の高い整数であると仮定して，演算を行い，タグを調べて整数でなかった場合はトラップを発生する．また，世代別 GC をハードウェアで支援しており，Store-40 命令実行時にタグを用いて古い世代のセルに新しい世代のセルの番地を書き込むことを検出するとトラップを発生する．

3. Prolog マシンの命令セットアーキテクチャ

Prolog マシンの歴史は比較的浅いが，大きく二つに区切ることができる．第 1 期のマシンは，ICOT の PSI[17)] や神戸大の PEK[18)] に代表され，D. Warren の作成した DEC 10-Prolog の仮想命令[19)] を基にしている．第 2 期のマシンは，1983 年に同じく Warren が提案した仮想アーキテクチャ[20)] (以下，WAM：Warren Abstract Machine と略する）を基にしており，以後現在までに発表された Prolog マシン[23),25),26),28)] はほぼ例外なく WAM をベースにしている．本章では，まず WAM の概要および拡張について説明し，WAM を基にした代表的な Prolog マシンのいくつかの命令セットアーキテクチャ上の特徴について紹介する．

3.1 WAM

WAM の特徴は，次のようにまとめることができる．

- レジスタを用いた引数の受け渡しの高速化．
- 変数の分類の詳細化．
- 構造体のコピー方式による表現．
- 終端再帰呼び出しの徹底．
- インデキシングの強化．

WAM は，次のような要素からなる．

引数レジスタ：述語呼び出しの際の引数を設定する領域であり，ゴール間にまたがって値を保持する必要のない変数の領域としても利用する．

ローカルスタック：制御情報（後述する環境フレーム，選択肢フレーム）を積むスタックである．

グローバルスタック：構造体（リストを含む）を作成するためのスタックである．

トレイルスタック：バックトラックするときに束縛を解く必要のある変数を記録するスタックである．

WAM の命令は，次のように分類できる．まず，一つの節を翻訳する場合に必要なのが，以下の命令である．

get 命令：レジスタ上の引数のユニフィケーションを行う．

put 命令：レジスタ上に引数を設定する．

unify 命令：構造体の要素のユニフィケーションを行う．unify 命令は，既に存在している構造体の要素のユニフィケーションを行うための read モードと，新たに作成する構造体の要素を設定するための write モードの 2 とおりに解釈される．

実行制御命令：述語呼び出しを行う命令，環境フレームを作成／廃棄する命令からなる．環境フレームとは，節からの戻り番地を退避し，節の中で使用する変数の領域を割り当てるものである．これは C や LISP で関数呼び出しの際に割り当てるフレームと同様のものである．

一方，複数の節から構成される述語の実行を制御するのが，次の命令である．

選択肢制御命令：選択点フレームを作成，修正，廃棄する．選択点フレームとは，複数の選択肢（節）がある場合，選択肢を実行する前の状態を保存するためのもので，選択肢の実行に失敗した場合に，この選択肢の実行直前の状態を復元することを可能とする．

インデキシング命令：述語が呼び出された時の実引数を動的に解析することによって，かならず失敗する節の実行を抑制する．

3.2 WAM のインプリメント

実際の Prolog マシンの命令セットとしては，WAM の原論文で述べられているものだけでは不十分であり，次のような点を考慮しておく必要がある．

- 構造体どうしのユニフィケーション

ユニフィケーションを行う命令で，ユニファイする二つのデータが構造体である場合がある．このような場合，もしも構造体が循環構造をもっていれば，（循環構造を陽に意識しないかぎり）命令の中で無限ループをおこす．このような事態を避けるためには，機械語命令の中からソフトウェアルーチンを呼び出すようなメカニズムを用意するような工夫が必要である．

- 組込み命令の導入，特にカット

WAM は pure-Prolog の実現に必要な命令についてしか述べておらず，実用的な処理系を実現するには，Prolog の提供する組込み述語を実現するための命令を用意する必要がある．ここでは，最も重要な組込み述語の一つのカット（！）について，二つの方式及びそのために必要な命令について述べる．

Choice-Commit 方式：カットすべき時点での選択点を記録しておき，カット時に選択点をこの記録された選択点に戻す方式である．これを実現するためには，現在の選択点フレームへのポインタをレジスタに格納する命令，及びカットを実行する命令，

```
choice   Ri
commit   Ri
```

を考えることができる．また，通常のカットを実現するには，述語が呼び出された時点での選択点を記録するメカニズムを備えていれば十分であるので，述語呼び出しを行う call/exec 命令で現在の選択点を退避することもできる[22]．

レベル方式：述語呼び出しのレベルを記録しておき，カットを実行する時点で指定されたレベルまでの選択点を刈り取る方式である．後述する PSI では，任意のレベルのカットを行うために，命令

```
relative_cut    Level
absolute_cut    Level
```

を用意している[24]．通常の手続きのカットは，relative_cut（1）で実現される．

3.3 WAM の改良

ここでは，WAM を基にした命令セットの改良の例について述べる．

- インデキシングの強化

WAM のクローズインデキシングは，頭部ゴール

の第1引数のみに基づいているので，次のような決定的なプログラムでも浅いバックトラックが必ずおきる．

```
foo (X) :- integer (X), !, ①
foo (X) :- ②
```

このような場合を考えて，レジスタ Ai が整数でなければ Lab 番地に分岐する命令

```
if_not_integer    Ai, Lab
```

を用意することによって次のような浅いバックトラックがおきないようなコードを生成できる．

```
if_not_integer    A0, Lab
    ①に対するコード
Lab :
    ②に対するコード
```

このほかにも，コンパイラの最適化の観点から検討を加えることによっていろいろな命令を考えることができる．

● 複合命令

翻訳して得られた命令列を解析すると，特定の命令のパターンを抽出することができる．このような命令パターンを一つの命令に統合することは，マイクロプログラマブルな高級言語マシンでは速度向上の面からもメモリ消費量の点からも有効である．たとえば，Prolog では次のような頭部ゴールでリストをユニフィケーションするプログラムを多用する．

$f([X|Y])$:- ,…

これは次のような命令列に翻訳されるが，

```
get_list          A0
unify_variable    Ax
unify_variable    Ay
```

これらをまとめて行う命令

```
get_list_var_var    Ai, Aj, Ak
```

を用意することができる[26]．

● インタプリタ用命令の導入

Prolog で実用規模の応用プログラムを作成する場合，大域データを実現するために述語 assert/retract を用いて述語に動的に節を追加／削除することが多い．このような述語はインタプリタを介して実行されるが，応用プログラムの全体の速度に影響するので，高速実行に対する配慮が必要である．文献27) はこのような問題に取り組んだ例である．この例では，assert 時に節を WAM をベースとした命令列にコンパイルし，得られたコードをポインタでつなぐ方式を採用しており，さらにこれらのコード間の実行制御を行う命令を導入することによってインタプリタの高速実行を達成している（最適化コンパイルを行った場合の 30～80% の速度）．また，clause などで使用するための assert 時のソースイメージを取り出すために，通常の実行を行うモードとソースイメージを取り出すためのモードとを設け，このモードに応じて命令の処理内容を切り替えることによって実行コードからソースイメージを取り出す工夫を行っている．

3.4 Prolog マシンの実例

1) PSI-II（三菱電機，ICOT）

第5世代コンピュータ研究開発プロジェクトの一環として開発された逐次型推論マシン PSI を改良，小型化したマシンである[23]．PSI はマイクロインタプリタによって表形式のオブジェクトを解釈実行する方式をとっていたが，PSI-IIではコンパイラの最適化を生かすために基本命令セットとして WAM を採用している．この基本命令セットは WAM をベースにしているが，**3.3** で述べたようなインデキシングの強化や複合命令の導入を行っている．さらに，PSI-IIは PSI と同じく，

● オブジェクト指向プログラミングの支援
● 強力な実行制御メカニズムの提供
● オペレーティングシステム構築に必要な機能の提供

のために，組込み述語 KL0[24] を直接機械語として提供している．

2) CHI（日本電気）

PSI と同じく，第5世代プロジェクトの一環として開発されたマシンである．現在までに試作版 CHI[25] と，これを改良小型化した小型化版 CHI[26]（以下 CHI-IIと略）とを開発している．PSI がスタンドアロン型のワークステーションであるのに対し，CHI はバックエンド型の構成を取っており，ホストマシン（CHI-II の場合は Unix ワークステーション）上のファイル，ウィンドウ，ネットワークなどの資源をそのまま利用できる環境を提供している．以下に CHI-IIの命令セットアーキテクチャ上の特徴を述べる．

汎用命令の導入：試作版 CHI は PSI の KL0 と同様な論理型言語風の高レベルの機械語を設定しているが，きめ細かなデータ操作や実行制御は記述し難いことから，CHI-II ではタグアーキテクチャを有効に利用しつつ，通常の汎用計算機で用意されているような四則演算命令，比較命令，転送，実行制御命令を用意している．

インタプリタ用命令：CHI-II では，インタプリタの高速実行のため，**3.3** で述べたように，動的に追加／削除される節を assert 時にインクリメンタル・コンパイルする方式を採用しており，このための実行制御命令，ソースイメージ取り出し命令を用意している．

3) その他の WAM マシン

その他の WAM ベースの Prolog マシンとしては，UCB の PLM[28]，東芝の WINE[29] がある．また汎用プロセッサに WAM の命令を付加するアプローチも試みられており，汎用ミニコンをベースとした日立の IPP[30] や，VAX 8600 に WAM 命令を追加した例[31]がある．

4) RISC プロセッサによるアプローチ

近年の RISC プロセッサの研究の進展にともない，記号処理向きのタグ付き RISC プロセッサ，及びその上の Prolog 処理系の研究も進められている[32]．この場合もコンパイラの中間言語として WAM を利用し，WAM コードを RISC 命令に展開することによって処理系を作成するのが一般的である．ところが，機能の高い WAM の命令を単純に機能の低い RISC 命令に展開しただけでは，コンパイルコードの量が爆発する，といった問題がある．文献 33) は，このような問題を評価した例であり，WAM をマイクロコードで実現した PLM 上の処理系と，RISC アーキテクチャの SPUR 上に作成した Prolog 処理系とのベンチマークプログラムの解析を行い，コード量は SPUR の処理系が PLM の処理系の 14 倍であるが，総実行ステップ数では SPUR が PLM の 16 倍，総マシンサイクルは SPUR が PLM の 2.3 倍であるという結果を得ている．

4. Smalltalk マシンの命令セットアーキテクチャ

代表的なオブジェクト指向言語，Smalltalk を高速実行する専用マシンの命令セットアーキテクチャについて解説する．まず，Smalltalk のメッセージ送受に関する考え方について簡単に述べる．ついで，Smalltalk 仮想マシンが定義しているバイトコード命令セットについて解説し，最後にレジスタマシン上で Smalltalk システムを作動させる研究を紹介する．

4.1 メッセージ送受

オブジェクト指向計算モデルの基本機構は，メッセージ送受であるといわれている[34]．特に，Smalltalk では，メッセージの受け手であるレシーバ・オブジェクトのクラスとメッセージの内容を示すセレクタから，実行すべきメソッドを動的に求める手続き呼び出し機構と考える[35]．たとえば，3+4 という式は，セレクタが＋でメッセージ引数が 4 のメッセージを整数オブジェクト 3 に送ることである．このとき，レシーバとなった整数オブジェクト 3 のクラスが管理するメソッド辞書を検索して，実行すべき加算メソッドを求める．この加算メソッドを実行した後，本メッセージを起動した地点に計算結果を戻す．この間，メッセージの送出元は実行結果を待つため，メッセージ実行に並列性はない．

4.2 仮想マシン・アーキテクチャ

仮想マシンの概要

Smalltalk 仮想マシンは，実行モデルとして，スタックマシンを採用し，バイトコード命令セットとプリミティブ・メソッドの仕様を定めている．バイトコードは仮想マシン命令を，基本的に 1 バイトの情報にコード化したものである．また，プリミティブ・メソッドは，入出力機能，グラフィックス機能やハードウェアの機能を用いるなど，メッセージ送受のみでは実行できない処理を行う組込み関数といえる．

バイトコードの長所として，コンパイラやデバッガといったシステム・ソフトウェアが作りやすいこと，インタプリタを作成するだけでシステムを移植できること，の二点がある．逆に，命令読み出しと解読の負荷が大きく，専用ハードウェアの支援がないと性能向上が図れないこと，実行制御が複雑なため低レベル機械語命令へのコンパイルが難しい，といった短所がある．

一方，Smalltalk システムのターゲット・マシンとして最初に選ばれた Dorado (Xerox) は，ECL テクノロジの高性能ワークステーションであった．大容量のマイクロメモリや命令プリフェッチ回路などをもつ汎用のバイトコード・エミュレータであるため，仮想マシンの命令セットとしてバイトコードを選んだのも，当然のことといえる[36]．

バイトコード命令セット

バイトコード命令セットを，次の 4 つに分類することができる．

（1） スタック操作命令
（2） ジャンプ命令
（3） センド命令
（4） リターン命令

スタック操作命令は，メモリ中のオブジェクトが内部に格納する情報と，評価スタックとの間の情報交換を行う．

Smalltalk では，すべての式をメッセージ送受により解釈するため，ジャンプ命令があることに奇異な感じがするかもしれない．しかし，条件判断式や繰り返し式は，プログラミング言語の基本的な制御構造であり，Smalltalk のメソッドにおいても多用される．そのため，高速に実行しなければならないし，このような基本的なメッセージのプロトコルをプログラマが変更することはない．そこで，Smalltalk コンパイラが，基本的な制御構造を示す特定セレクタを解釈して，ジャンプ命令を生成する．

センド命令は Smalltalk のメッセージ送出を実現する命令であり，以下のアルゴリズムにより作動する．

（1）　レシーバ・オブジェクトのクラスを求める．

（2）　求めたクラスが管理するメソッド辞書中を探して，セレクタに対応するメソッドを求める．

（3）　メソッドが見つからない場合，該クラスのスーパクラスに対して(2)を繰り返す．スーパクラスが無い場合，メソッド未定義のエラーを発生する．

（4）　メソッドが見つかった場合，新しい実行コンテクストを生成して，メソッドを実行する．

また，センド命令には，スーパ・センドを実現する変形がある．この命令は，Smalltalk のクラス継承機構と密接な関係があり，メソッドコンパイル時に決定するクラスから探索を開始することを示す．

リターン命令は，センドによって起動された実行コンテクストの終了を表し，メッセージ起動箇所，すなわち対応するセンド命令の次命令に制御を戻す．

4.3　レジスタマシン上での実現方式

前節で述べた，バイトコード・エミュレータによるシステム以外に，レジスタマシンへのコンパイラを開発することにより高性能を得よう，というアプローチがある．本アプローチでは，MC 68000 をターゲットとした Deutsch-Schiffman の研究[37]が先駆的で，25 MHz の MC 68020 を CPU としてもつ UNIX ワークステーション上で，Dorado の2倍にあたる性能を引き出すことに成功している．また，RISC 系プロセッサ[43]をターゲットとした SOAR プロジェクト[39]があり，コンパイラとアーキテクチャ・サポートの研究を並行して進め，Dorado と同等の性能を達成している．

コンパイラによるアプローチ

コンパイラによるアプローチの長所としては，バイトコードの解読などによる実行時負荷がないこと，インタプリタによってスタックマシンのシミュレートを行う必要がなく，レジスタマシンに適した実行モデルを採用できること，などをあげることができる．また，RISC プロセッサでは，操作対象データをチップ内レジスタに割り付けることができ，メモリ参照を極力減少させることが高速化の前提であるため，高度なコンパイラの開発が必須となる．

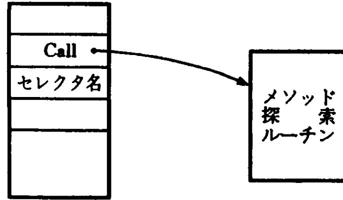
一方，大きな問題点として，バイトコード方式に比べて使用メモリ容量が増加する点がある．Deutsch-Schiffman は，メッセージ送受要求に応じて，必要なメソッドのみをコンパイルし，その後キャッシングして再利用する方式によりメモリ使用量の増加を抑えている．また，SOAR ではコンパイラが生成するコード量が，バイトコードに比べて約 3.78 倍の増加ですんだ．そのため，Smalltalk 標準システムの全メソッドをコンパイルしても，約 0.4 MB 増加するだけであり，メモリ容量はあまり問題にならないと報告している[42]．

もう一つの問題は，Smalltalk 標準システムが提供するデバッガなどのシステム・プログラムを，利用できなくなるという点である．特に，命令カウンタやスタックポインタなどのデバッガに見せる情報と，プロセッサのハードウェア状態の対応関係を，動的に管理しなければならない．Deutsch-Schiffman では，デバッガ使用時などにより必要になった時点で，両者の情報を動的に変換する方式をとって，Smalltalk 標準デバッガをそのまま使用できるようにした．一方，SOAR では専用のデバッガを別途開発している．

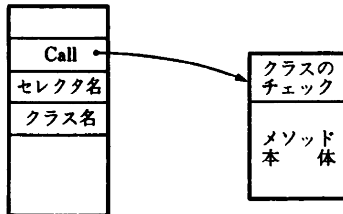
また，メソッド探索を効率化する技法として，インライン・キャッシュと呼ぶ方式[38),39]を採用することが多い．一般に，あるメッセージに着目した場合，同一クラスのオブジェクトをレシーバとしてメッセージを実行することが多い．コンパイラは，メッセージ送出を実現するコードとして，メソッド探索手続きを呼び出す call 命令を生成する．実行時にメソッド探索を行い，実行すべきメソッドの実行開始番地を得る．同時に，次のような手続きを実行する命令に書き換える（図-1 参照）．

（1）　レシーバがメソッド探索を行ったときのクラスに属するかどうかのチェックを行う

（2）　属すれば，キャッシングしてある番地のメソ



（a） コンパイラが生成するコード



（b） 書き換え後のコード

図-1 インライン・キャッシュ

ッドを実行する

（3） そうでなければ，メソッド探索を行う

以降のメッセージ送受時には，書き換えた命令を実行するので，メソッド探索を省略することができる．

SOAR

SOAR プロセッサ (UCB)[39] は，固定長形式の命令，メモリ参照を Load/Store に限定，3オペランドの論理演算や整数演算命令，大容量のレジスタウィンドウなど RISC-II[43] の流れを汲む RISC プロセッサである．さらに，Smalltalk を高速に実行する機能として，即値表現の整数 (Small Integer) とポインタ表現を区別するタグ情報の識別回路，自動記憶管理機構として採用した世代別回収型コピー方式を支援するタグ識別回路，などを追加した点が特徴である．また，多態実行環境を実現する線形スタックをレジスタウィンドウに割り付ける技法により，高速性能を実現した．

SOAR コンパイラは，Smalltalk メソッドのソースコードをバイトコードに変換するオリジナル・コンパイラの拡張として，Smalltalk により作成された[45]．すなわち，オリジナル・コンパイラが出力するバイトコード系列を入力として受付け，メソッド実行時に必要な一時変数や評価スタックなどをレジスタに割り付けるとともに，バイトコードを記号実行して評価スタック上の動きをシミュレートすることにより，SOAR の命令列を生成する．

もう一つの特徴は，ハードウェアが提供するタグ識別機構を利用できるので，コンパイラのコード生成が簡単になった点である．たとえば，$c \leftarrow a+b$ という式を展開した次のような疑似バイトコード列を，3オペランドの加算命令一語で実現することができる．

```
push :     a
push :     b
send :     +     →     add rA, rB, rC
pop into : c
```

ここで，add はレジスタ rA と rB が即値整数 (Small Integer) であると仮定して，整数の加算演算を行い，その結果を rC に格納する SOAR の加算命令である．SOAR では LISP マシンの SPUR と同様に，加算対象が即値整数でない場合はトラップを発生し，起動されたトラップ処理ルーチンが，通常のメソッド探索を行って処理を進める．また，加算結果が即値整数で表現可能な大きさを越えた場合にもトラップを発生し，大きな整数を表現できるオブジェクト (Large Integer) に変換する．

最適化コンパイラの可能性

最適化コンパイラを開発するためには，ソーステキストから得られる静的な情報を基に，実行すべきメソッドをコンパイル時に決定するアルゴリズムの確立が重要となる．現在のところ，Smalltalk の言語仕様をまったく変更しないで，タイプ推論手法により，レシーバのクラスを決定するアプローチ[44]と，変数に対してクラス情報を注釈としてプログラマが与えられるように Smalltalk 言語仕様を変更し，本注釈を基にしてシステムが推論するアプローチ[45],[46] の双方が試みられている．今後，この方向に関する研究の進展により，高性能な最適化コンパイラの実現が期待されている．

5. 終わりに

本稿では，AI 用の高水準言語計算機に絞って，その命令セットアーキテクチャについて述べた．

高水準言語計算機の命令セットアーキテクチャを議論するに当たっては VLSI 技術の進展と，コンパイラ技術の進歩を無視するわけにはならない．

たとえば SPUR や SOAR のような，RISC チップは，従来の高水準言語計算機の反省から出現したものであるといえる．一方，Ivory, CLM のような第3世代の高水準言語計算機の命令セットでは，実行時チェック機能など，AI 用言語マシン以外のマシンでも重要な機能を実現しながら，1チップ化を図っている．

ソフトウェアの信頼性が重視されるにつれ，これら

の高水準言語計算機の命令セットが他の計算機の命令セットにどのような影響を与えていくのかについて，注目していきたい．

参 考 文 献

1) Myers, G. J.: Advances in Computer Architecture, John Wiley and Sons (1978).
2) 箱崎，山本：高級言語マシンの実際，産報出版 (1981).
3) 柴山：プログラミング言語処理におけるマイクロプログラミング技術，情報処理，Vol. 28, No. 12, pp. 1595-1609 (1987).
4) Patterson, D.: Reduced Instruction Set Computer, CACM, 28(1), (Jan. 1985).
5) Steele, G.: Common LISP, Digital Press (1984).
6) LMI Inc: The Microcompiler
7) 瀧，金田，前川：LISP マシンの試作，情報処理学会論文誌，Vol. 20, No. 6, pp. 481-493 (1979).
8) 服部他：ALPHA-高性能 LISP マシン，情報処理学会研究会報告，Vol. SM-23, No. 1, pp. 1-8 (1983).
9) Yuhara, M. et al.: Evaluation of the FACOM ALPHA Lisp Machine, 13th Annual Int. Symp. on Comp. Architecture, pp. 184-190 (1986).
10) 日比野他：LISP マシン ELIS のアーキテクチャ・メモリレジスタの汎用化とその効果，情報処理学会研究会報告，Vol. SM-24, No. 3, pp. 1-8 (1983).
11) 神尾他：ELIS における Lisp コンパイラ，情報処理学会研究会報告，Vol. SM-40, No. 4, pp. 1-6 (1983).
12) Moon, D. A.: Architecture of the Symbolics 3600, 12th Annual Int. Symp. on Comp. Architecture, pp. 76-83 (1985).
13) Moon, D. A.: Symbolics Architecture, IEEE Computer, pp. 43-52, Jan. (1987).
14) Krueger, S.D. and Bosshart, P.W.: High LISP Performance on a High-Level Language Processor, COMPCON Spring 87, pp. 120-132.
15) 平木，後藤：数式処理計算機 FLATS のアーキテクチャ，情報処理学会論文誌，Vol. 27, No. 1, pp. 81-89 (1986).
16) Taylor, G. S. et al.: Evaluation of the SPUR Lisp Architecture, Proc. 13th Annual Int. Symp. on Comp. Architecture, pp. 444-452 (1986).
17) Yokota, M., Yamamoto, A., Taki, K., Nishikawa, H. and Uchida, S.: The Design and Implementation of a Personal Sequential Inference Machine: PSI, New Generation Computing, Vol. 1-2 (1983).
18) Kaneda, Y., Tamura, N., Wada, K. and Matsuda, H.: Sequential Prolog Machine PEK Architecture and Software System, Proc. of the International Workshop on High-Level Computer Architecture (1984).
19) Warren, D. H. D.: Implementing Prolog-Compiling Predicate Logic Program Vol. 1-2, D. A. I. Research Report, No. 39-40, Univ. of Edinburgh (1977).
20) Warren, D. H. D.: AN ABSTRACT PROLOG INSTRUCTION SET, Technical Note 309, SRI International (1983).
21) 横田 実：逐次型 Prolog マシン，人工知能学会誌，Vol. 2, No. 4 (1987).
22) Carlsson, M.: On Compiling Indexing and Cut for the WAM, SICS R 86011, Swedish Institute of Computer Science (1986).
23) 中島克人他：マルチ PSI 要素プロセッサ PSI-II のアーキテクチャ，情報処理学会第 33 回全国大会論文集，(1986).
24) 新世代コンピュータ技術開発機構：KL0 組込述語説明書 (1986).
25) Nakazaki, R., Konagaya, A., Habata, S., Simazu, H., Umemura, M., Yamamoto, M., Yokota, M. and Chikayama, T.: Design of a Highspeed Prolog Machine (HPM), Proc. of the 12th International Symposium on Computre Architecture (June 1985).
26) Habata, S., Nakazaki, R., Konagaya, A., Atarashi, A. and Umemura, M.: Co-operative High Performance Sequential Inference Machine: CHI, Proc. of 1987 IEEE International Conference on Computer Design (Oct. 1987).
27) 小長谷明彦：高速 Prolog インタプリタの構築法とその評価について，情報処理学会記号処理研究会報告 46-4 (1988).
28) Dobry, T. P., Patt, Y. N. and Despain, A. M.: Design Decisions Influencing the Microarchitecture for a Prolog Machine, 17th Annual Microprogramming Workshop of IEEE Computer Society (Oct. 1984).
29) 斉藤光男他：AI ワークステーション (WINE) の開発，情報処理学会第 35 回全国大会論文集 (1987).
30) Abe, S., Bandoh, T., Yamaguchi, S., Kurosawa, K. and Kiriyama, K.: High Performance Integrated Prolog Processor IPP, Proc. of Int'l Symposium on Computer Arthitecture (1987).
31) Gee, J., Melvin, S. W. and Patt, Y. N.: The Implementation of Prolog via VAX 8600 Microcode, 19th Annual Workshop Microprogramming (1986).
32) 瀬尾和夫，横田隆史：Prolog 指向 RISC プロセッサ "Pegasus"，信学会電子計算機アーキテクチャ研究会資料，CAS 86-121 (1986).

33) Borriello, G., Cherenson, A. R., Danzig, P. B. and Nelson, M. N.: RISCs vs. CISCs for Prolog: A Case Study, Proc. of the 2nd International Conference on Architectural Support for Programming Languages and Operating Systems (1987).
34) 米澤明憲：オブジェクト指向プログラミングについて，コンピュータソフトウェア，1(1) (Jan. 1984).
35) Goldberg, A. and Robson, D.: Smalltalk-80: The Language and Its Implementation, Addison-Wesley (1983).
36) Krasner, G. (ed.): Smalltalk-80: Bits of History, Words of Advice, Addison-Wesley (1983).
37) 鈴木則久，小方一郎：「多態実行環境」：高級言語の制御機械の高性能実現法，情報処理，Vol. 26, No. 11 (1985).
38) Deutsch, L. P. and Schiffman, A. M.: Efficient Implementation of Smalltalk-80 System, 11th POPL (Jan. 1984).
39) Ungar, D. and Patternson, D.: What Price Smalltalk?, IEEE Computer (Jan. 1987).
40) Suzuki, N., Kubota, K. and Aoki, T.: Sword-32: A Bytecode Emulating Microprocessor for Object-Oriented Languages, FGCS' 84 (Nov. 1984).
41) Lewis, D. M. et al.: Swamp: A Fast Processor for Smalltalk-80, OOPSLA' 86 (Sep. 1986).
42) Bush, W. et al.: Compiling Smalltalk-80 to a RISC, ASPLOS-II (Oct. 1987)
43) Patterson, D.: Reduced Instruction Set Computer, CACM, 28(1) (Jan. 1985).
44) Suzuki, N.: Inferring Types in Smalltalk, 8 th POPL (Jan. 1981).
45) Borning, A. H. and Ingalls, D. H. H.: A Type Declaration and Inference System for Smalltalk, 9th POPL (Jan. 1982).
46) Atkinson, R. G.: Hurricane: An Optimizing Compiler for Smalltalk, OOPSLA' 86 (Sep. 1986).

(昭和 63 年 9 月 29 日受付)