計算機アーキテクチャ 74－4
（１９８９．１．２６）

ＡＩワークステーション（ＷＩＮＥ）の評価

河込和作　小泉善裕　中村明　星野康夫　斎藤光男
（株）東芝

我々は、ＥＷＳ「ＡＳ３０００」にバックエンドプロセッサＡＩＰを付加した構成のＡＩワークステーション「ＷＩＮＥ」を開発し、その上で動作するＬｉｓｐ処理系を構築した。

開発されたＬｉｓｐ処理系は「ＡＩＰ－Ｌｉｓｐ」と呼ばれ、現在の標準である「Ｃｏｍｍｏｎ　Ｌｉｓｐ」に準拠した言語仕様を持っている。

Ｇａｂｒｉｅｌベンチマークによる性能評価の結果、「ＡＩＰ－Ｌｉｓｐ」の速度性能が当初の開発目標をほぼ達成していることが確かめられた。

本稿では、「ＡＩＰ－Ｌｉｓｐ」について、性能評価の結果と共に、開発戦略および実現の概要を報告する。

Performance Evaluation of AIP-Lisp on AI workstation (WINE)

Kazusaku KAWAGOME, Yoshihiro KOIZUMI, Akira NAKAMURA, Yasuo HOSHINO, and Mitsuo SAITO
TOSHIBA Corporation
Information and Communication Systems
Laboratory, TOSHIBA, Fuchu-shi, 183, Japan

WINE is an AI workstation which is built as a combination of a conventional EWS and an AI-oriented back-end processor AIP.

In this paper, we report the issues of "AIP-LISP" on this WINE, its performance, design strategy, and outline of implementation.

AIP-Lisp is a Common Lisp system which runs on the AIP of WINE. Performance evaluation of AIP-LISP has been done for the Gabriel's benchmark programs.

For almost all of the benchmark programs, AIP-Lisp is several times faster than other Lisp systems on conventional EWS or traditional Lisp machines.

0.　はじめに

近年、人工知能の研究が実用化に向けて急速に進み、さまざまな応用システムの開発が行われている。それにともなってＡＩワークステーションにも、実行速度の高さがまず求められるようになってきた。

我々は、ＡＩワークステーションＷＩＮＥ（Ｗｏｒｋｓｔａｔｉｏｎ ｗｉｔｈ Ｉｎｔｅｌｌｉｇｅｎｔ ａｎｄ Ｎａｔｉｖｅ Ｅｎｖｉｒｏｎｍｅｎｔ）を開発中である。現在ＷＩＮＥ上でＬｉｓｐ、Ｐｒｏｌｏｇ言語の処理系が稼働を開始している。

本報告では、ＷＩＮＥの２次試作ハードウェア上で動作しているＣｏｍｍｏｎ Ｌｉｓｐに準拠したＬｉｓｐ処理系ＡＩＰ－Ｌｉｓｐの概要と、Ｇａｂｒｉｅｌベンチマークによる評価結果を中心にして述べる。

1.　ＷＩＮＥの概要

1．1　開発の背景

ＡＩワークステーションＷＩＮＥは、当社のエンジニアリングワークステーションＡＳ３０００シリーズに、ＡＩ言語処理用のプロセッサＡＩＰを付加したものである。

我々がＷＩＮＥを開発するにあたって、ＡＩワークステーションに求められる特性として次の５つを想定した。

（１）　演算速度が速いこと
（２）　目的に応じた言語を使用できること
（３）　既存のソフトウェア資産を活用できること
（４）　入出力機器等の拡張が容易であること
（５）　開発環境が優れていること

我々はこのうち、（１）、（２）、（５）の実現を重視することにして実現方式を検討した。その結果、既存のＥＷＳ（エンジニアリングワークステーション）ＡＳ３０００シリーズに、ＡＩ機能を強化したプロセッサシステムＡＩＰをバックエンドに付加した構成を決定し、ＷＩＮＥとして実現した。

1．2　ハードウェアアーキテクチャ

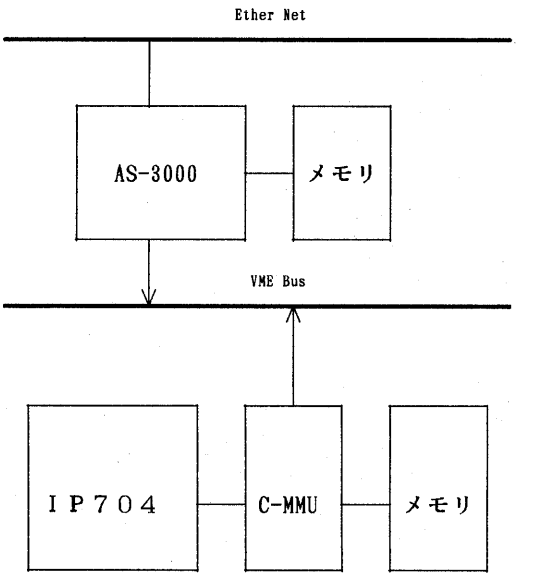
ＷＩＮＥの中心をなすＡＩＰは、（図１）に示すような３種類のボードから構成されている。

（１）　ＩＰ７０４プロセッサボード
（２）　ＣＭＭＵ（キャッシュ／ＭＭＵ）ボード
（３）　ＭＥＭ（３２ＭＢ／枚）ボード、最大１２８ＭＢ

プロセッサＩＰ７０４の内部構造を（図２）に示す。高速化を図るため、データバスの２重化、型タグ取り扱いのハードウェア化等を採用し、ＲＩＳＣ思想とマイクロプログラム制御が統合されたアーキテクチャを備えている。

ＩＰ７０４は２次試作において、主としてＬｉｓｐの動作速度を改善するため型タグの取り扱い等に改良が加えられている。

図１　<<ＷＩＮＥの構成>>



2. ＡＩＰ－Ｌｉｓｐ

2．1　開発方針

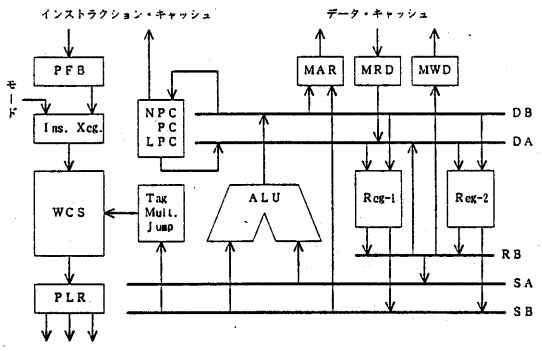
ＡＩＰ－Ｐｒｏｌｏｇでは、ＡＳ３０００上で動作するＣ－Ｐｒｏｌｏｇインタプリタの制御のもとでコンパイル述語をＡＩＰ上で動作させ、ホストとＡＩＰが協同してＰｒｏｌｏｇシステムを構成する方法を採用したが、Ｌｉｓｐ（ＡＩＰ－Ｌｉｓｐ）はこれと異なり、全実行時システムをＡＩＰ上に配置し、ホストであるＡＳ３０００には、起動・停止処理、低レベルのＩ／Ｏ、ＯＳコマンド実行等だけを行わせる方式を採用している。

この方針は、次にあげる２つの条件によって決定されたものである。

（1）　Ｐｒｏｌｏｇと比べてＬｉｓｐの基本演算は原始的であり関数間のデータ結合も密であるため、通常の処理系では２つのプロセッサ間で制御やデータを受け渡すのに適さない。

（2）　何よりも、ＡＩＰの全機能を用いて高速化を達成することに目的があった。

図２　＜＜ＩＰ７０４＞＞の内部構造



2．2　Ｌｉｓｐサポート命令

ハードウェアの項で述べたように、ＡＩＰはＲＩＳＣ思想とマイクロプログラム制御を統合したアーキテクチャを備えている。ＡＩＰ－Ｌｉｓｐの開発はＬｉｓｐサポート命令の検討から開始された。

Ｌｉｓｐサポート命令の仕様決定において前提条件となったのは、次のような点であった。

（1）　命令は３２ビット固定長のみ。
（2）　マイクロプログラム容量をできるだけ減らす。
（3）　マイクロプログラム化することで大幅に高速化するものだけを選ぶ。
（4）　必要なら、例外発生機構を利用して処理をソフトウェアに引き渡す。

ＡＩ用高級言語のうち、ＰｒｏｌｏｇではＷＡＭのような広く認められた高水準の仮想命令体系が知られており、ＡＩＰも含め多くのＰｒｏｌｏｇマシンはそれに従って設計されている。しかしＬｉｓｐは、基本的には非常に低水準な“基本関数”から全体系が構成される関数指向型言語である。

そこで、Ｌｉｓｐサポート命令の基本的な体系として、Ｌｉｓｐの基本関数のうちマイクロプログラムで実現することによって速度向上に大きく寄与すると期待されるものについて命令化する、という基本戦略をとることにした。

Ｌｉｓｐサポート命令の体系は、汎用系命令、Ｐｒｏｌｏｇサポート命令と同様に、レジスタ指向のものとした。数値演算は（一部の即値を除いて）レジスタ間のみで行なわれ、メモリアクセスはロード・ストアアーキテクチャである。

既存のＬｉｓｐマシンでは、スタックマシン指向のアーキテクチャをとるものが多い。ＡＩＰにおいてもマイクロプログラムによってスタックマシンをエミュレートすることは可能であったが、スタックマシンでは中間結果のｐｕｓｈ／ｐｏｐがどうしても多くなり最適化の障害になるため、レジスタ指向の命令セットが採用された。

今回、Ｌｉｓｐサポート命令として取り込まれたのは主として次の５つの範疇に属するＬｉ

表1　<<Ｌｉｓｐサポート命令の例>>

（1）　型タグ指定付きロード、ストア ｃｘｒ　Ｒｄ　Ｒｐ　Ｄｉｓｐ Ｄｉｓｐ＝0のとき、ＲｐのｃａｒをとってＲｄへ転送する。 Ｄｉｓｐ＝4のとき、ＲｐのｃｄｒをとってＲｄへ転送する。 ｓｔｏ　Ｔｙｐｅ　Ｒｓ　Ｒｐ　Ｄｉｓｐ Ｒｐの型タグとＴｙｐｅが一致すればＲｐによって指されるオブジェクトのＤｉｓｐ番目のスロットにＲｓの内容を書き込む。そうでなければ例外を発生する。
（2）　ｃｏｎｓ ｃｏｎｓ　Ｒｄ　Ｒｓ１　Ｒｓ２ ヒープ上にコンスセルを割り付ける。ｃａｒをＲｓ１、ｃｄｒをＲｓ２の内容で初期化し、割り付けられたコンスセルの型タグ付きポインタをＲｄに転送する。
（3）　型タグの一致、不一致による分岐 ｂｔｅ　Ｔｙｐｅ　Ｒｓ　Ｄｉｓｐ Ｒｓの型タグの内容がＴｙｐｅと一致すれば、Ｄｉｓｐで指定されるアドレスへ相対ブランチする。一致しなければ、次の命令を実行する。
（4）　ジェネリックな数値演算 ｇａｄｄ　Ｒｄ　Ｒｓ１　Ｒｓ２ Ｒｓ１とＲｓ２が表す内容を加算して、結果をＲｄに転送する。Ｒｓ１、Ｒｓ２の少なくともいずれかがｆｉｘｎｕｍでもｓｈｏｒｔ－ｆｌｏａｔでなければ例外が発生する。 演算結果がｆｉｘｎｕｍで表現できない場合にも例外が発生する。
（5）　２８ビット幅の非ジェネリック算術論理演算 ｌａｎｄ　Ｒｄ　Ｒｓ１　Ｒｓ２ Ｒｓ１とＲｓ２の値フィールド（ワードの下位２８ビット）の論理積を求め、Ｒｄに転送する。Ｒｄの型タグフィールドには、Ｒｓ１の型タグフィールドの内容が転送される。

Ｒｘ　　：　レジスタ
Ｔｙｐｅ　：　型タグ（定数）
Ｄｉｓｐ　：　変位（定数）

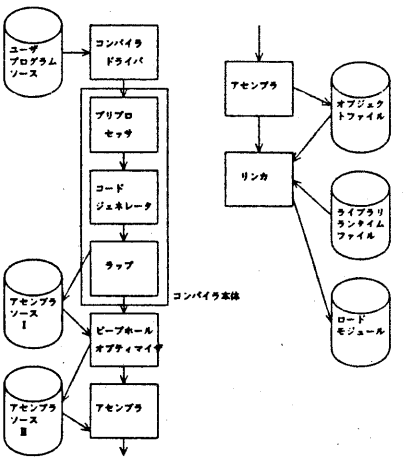
ｓｐの基本操作である。（表１）に代表的なＬｉｓｐサポート命令の形式と仕様の例を示す。

（1）　型タグ指定付きロード、ストア（ｃａｒ、ｃｄｒを含む）
（2）　ｃｏｎｓ
（3）　型タグの一致、不一致による分岐
（4）　ジェネリックな基本数値演算（加減乗除とａｓｈ）
（5）　２８ビット幅の非ジェネリック算術論理演算

２．３　ＡＩＰ－Ｌｉｓｐコンパイラ

ＡＩＰ－Ｌｉｓｐコンパイラ第１版は既存のＬｉｓｐシステム（ＣＬＩＳＰ；ＫＣＬの当社製品化版）の上につくられ、ＡＩＰの標準アセンブラソースプログラムを出力するクロスコンパイラである。コンパイル処理は（図３）に示すように行われる。

図３　<<コンパイル処理の流れ>>



本クロスコンパイラには、あまり高度な最適化機構は実装されていないが、組み込み関数のインライン展開やデータフロー解析等により、出力されるオブジェクトコードの実行速度を十分なものとすることに重点をおいて設計されている。（図４）に本コンパイラによるコンパイルの例を示す。

あとで述べるように、ＡＩＰ－Ｌｉｓｐ実行時システムの大部分は本クロスコンパイラを使用して構築されている。

図４　＜＜コンパイル結果の例＞＞

```
(defun fact (n)
  (if (zerop n)
    1
    (* n (fact (1- n)))))
```

```
#include        "alglobals.h"

.ecode
AL$!FACT:
        lpsh    r6,r1
        lpsh    r0,r1
        mv      r6,r18
        cmpni   r6,0
        bc      nz,L1167
L1205:
        ldfx    r1c,1
        lpop    r0,r1
        lpop    r6,r1
        bai     r0,4
L1167:
        gsubi   r18,r6,1
        ldfx    r1c,1
        call    AL$!FACT
        call    vget1st
        gmul    r1c,r6,r1c
        lpop    r0,r1
        lpop    r6,r1
        bai     r0,4

        .globl  AL$!FACT
```

２．４　実行時システム

２．４．１　開発方針

ＡＩＰ－Ｌｉｓｐの実行時システムはＬｉｓｐによって全面的に記述されている。ＡＩＰ－Ｌｉｓｐ実行時システムのソースコード全体をみても、９０％以上がＬｉｓｐで記述されておりＣ言語やアセンブリ言語で記述されているのは、低レベルの例外処理、入出力、超越関数等に過ぎない。このことにより、単一の制御スタック、レジスタ使用、手続き呼び出しコンベンション等を採用することができ、効率が良く保守性の優れた実行時システムを作成することができた。

実行時システム全体をＬｉｓｐで記述する場合の問題点としてよく指摘される実行性能であるが、我々は、コンパイラの基本性能を高いものとする努力を払うとともに、一部の数値演算ルーチンに対しては低レベル記述向きの言語仕様拡張を行っている。

２．４．２　データ表現

ＡＩＰ－Ｌｉｓｐにおける型タグの実現は、ＡＩＰのハードウェアに合わせたタグ付きポインタ方式を採用している。

ハードウェアの項で述べたように、ＡＩＰには３２ビットのワードデータの上位４ビットを型タグ、下位２８ビットを値（ポインタ）フィールドとして切り分けて使用するハードウェア機構が装備されている。そのため、１６種類の型をハードウェアによって直接サポートすることができるが、Ｃｏｍｍｏｎ　Ｌｉｓｐの豊富なデータ型に対しては決して十分な数とはいえない。

型タグの割り当ては、Ｃｏｍｍｏｎ　Ｌｉｓｐのデータ型の実現方式、処理系内部で必要とされる特別なデータ型の必要性の検討に基づき（表２）に示すようになっている。

２．４．３　関数呼び出しのコンベンション

Ｌｉｓｐは他の言語と比べて関数呼び出しが非常に多いことが知られている。そのため関数

呼び出し方式が性能に与える影響は大きい。ＡＩＰ－Ｌｉｓｐでは下記のようなコンベンションを採用し、効率的な関数呼び出しを実現している。

（１）　引数はレジスタ渡し

引数は、最初の５個までがレジスタに入れて渡される。残った引数はスタックに積まれて渡される。引数の数は関数値レジスタに入れて渡され、引数のポップアップは呼び出し側によって行われる。

引数をレジスタ渡しにした場合、Ｃｏｍｍｏｎ　Ｌｉｓｐでは、呼び出される関数が、＆ｏｐｔｉｏｎａｌ、＆ｒｅｓｔ、＆ｋｅｙ等のラムダキーワードを持つ場合の処理がやや複雑になるという問題点がある。

ＡＩＰ－Ｌｉｓｐではこれらの特殊な場合の処理効率よりも、大部分の場合にメモリアクセス回数を低減することを優先し、現方式を採用することにした。

（２）　関数値はレジスタ返し

関数呼び出しの効率をあげるため、関数が返却する値はレジスタに格納されるようにした。しかしこの場合、Ｃｏｍｍｏｎ　Ｌｉｓｐは通常の１つの値を返却する関数の他、一度に複数の値を返却する関数（多値関数）を定義することができるので、その処理をどのように行うかという問題が発生する。

多値を実現する方法としては値をスタック返しにする等が知られているが、ＡＩＰ－Ｌｉｓｐでは多値の場合に多少のオーバヘッドはあるものの通常に使用される単値の場合が高速に処理できることから、多値の場合には関数値レジスタに多値格納領域へのポインタを設定し、呼び出し側でのオーバヘッドを省くために単値と多値でリターンアドレスをずらす方式を採用した。

ＡＩＰ－Ｌｉｓｐでは単値の場合のリタンアドレスを１ワード（４バイト）ずらし、多値操作は全て実行時ルーチンが行うようにしている。多値の格納にはヒープではなく固定領域が使用されている。また、関数内部で使用するスタックフレームの構造を工夫することにより、ｕｎｗｉｎｄ－ｐｒｏｔｅｃｔや

表２　＜＜型タグの割り当て＞＞

タイプコード	タイプ名	データ型
0	external-pointer	スタックポインタ・リタンアドレス等
1	subtype-code	オブジェクト先頭のサブタイプ識別コード
2	character	文字
3	short-float	短形式浮動小数点数
4	fixnum	固定長整数
5	Reserved	未使用
6	Reserved	未使用
7	GC-forward	ＧＣ用のフォワードポインタ
8	simple-string	単純文字列
9	symbol	シンボル
１０	Reserved	未使用
１１	structure	構造体
１２	extended-number	非即値形式の数（ｂｉｇｎｕｍ、ｒａｔｉｏ）
１３	simple-vector	単純ベクタ
１４	cons	コンスセル
１５	indirection	クロージャ変数用間接ポインタ

ｍｕｌｔｉｐｌｅ－ｖａｌｕｅ－ｃａｌｌ等を使用した場合でもヒープを消費しないようになっている。

（３）　関数は原則として直接呼び出し

ＡＩＰ－Ｌｉｓｐでは、コンパイル時に呼び出される関数の名称を確定できる限り、言い替えれば明示的に間接呼び出しを行うようなコーディングを行わない限り、すべての関数呼び出しに対してｃａｌｌ命令を使用した直接呼び出しを行うようなオブジェクトコードが生成されるようになっている。

そのため、インクリメンタルローディングを実装した場合に機構が複雑化し関数の再定義に時間がかかることになるが、一部の処理系に見られるブロックコンパイル（同一のファイル内で定義された関数間の呼び出しを直接呼び出しとし、そうでない呼び出しを間接呼び出しとする方式）よりも意味的に明確であり、何よりも関数呼び出しが常に高速化されるため現在の方式に決定した。

２．４．４　レジスタの使用

ＡＩＰは３２個の汎用レジスタを持つ。レジスタは固定であり、レジスタウィンドウ等の機構は実装されていない。ＡＩＰ－Ｌｉｓｐではこのアーキテクチャの特性を活用すべくレジスタ割り当てを設計した。現在のＡＩＰ－Ｌｉｓｐでのレジスタ使用を（表３）に示す。

２．５　評価

ＡＩＰ－Ｌｉｓｐの評価には、Ｌｉｓｐのベンチマークとしては最も著名であるＧａｂｒｉｅｌベンチマークを使用した。

（表４）に示す結果は、１１０ｎｓのクロックで動作するＡＩＰの実機によって実測された値であり、単位は秒である。

得られた結果を他の処理系について公表された値と比較してみると、大部分の項目ではＡＳ３０００等の既存の汎用ワークステーション上の処理系やＬｉｓｐマシン上の処理系と比較して、数倍から十数倍の性能を得ることができた。特にｄｅｓｔｒｕｃｔｉｖｅ等のリスト操作では、リスト操作用に導入された命令が十分な効果をあげていることを確認できた。

表３　＜＜レジスタの割り当て＞＞

（１）　グローバルレジスタ　１１個
Ｌｉｓｐシステム全体として役割の固定されたレジスタ。 ｒ０　リタンアドレス ｒ１　制御スタックポインタ ｒ２　フレームポインタ ｒ３　束縛スタックポインタ ｒ４　引数ポインタ ｒ５　定数ベクタポインタ ｒ１ｂ　ヒープポインタ ｒ１ｃ　値レジスタ ｒ１ｄ　キャッチャフレームポインタ ｒ１ｅ　定数　Ｔ ｒ１ｆ　定数　ＮＩＬ
（２）　ローカル変数レジスタ　１０個
ｃａｌｌの前後で値が保証されるレジスタ。入口でセーブ、出口でリストアされる。
（３）　一時変数レジスタ　１１個
ｃａｌｌの前後で値が保証されないレジスタ。 ｒ１３－ｒ１８は引数レジスタとしても使用される。

表4　<<ベンチマークの結果>>

項目	結果
TAK	0.16
STAK	1.26
CTAK	0.83
TAKL	0.98
TAKR	0.20
BOYER	2.51
BROWSE	3.52
DESTRU	0.34
DERIV	0.83
DDERIV	0.88
DIV2-I	0.26
DIV2-R	0.40
FFT	4.15
PUZZLE	2.64
TRIANG	47.95
TRAV-INIT	2.05
TRAV-RUN	36.64
FRPOLY FIX	0.14
FRPOLY BIG	2.20
FRPOLY FLO	0.17

注：　FRPOLYは　n=10

しかし、一部ではｔｒａｖｅｒｓｅやｂｉｇｎｕｍ演算のように極端に悪い結果が得られている。これらについてはアセンブラソースレベルで解析した結果、ｔｒａｖｅｒｓｅについては構造体アクセスのインライン化が未実装であること、ｂｉｇｎｕｍ演算については測定に使用されたｂｉｇｎｕｍ演算ルーチンが暫定版であり調整が不足していることが主な原因であることが判明している。これらの不具合は今後の改良によって取り除かれる予定である。

全体的にみて、測定に使用したＷＩＮＥのクロックが１１０ｎｓ（９ＭＨｚ強）に過ぎないことを考えると、所期の目標は達成されたと考えられる。

3.　結論

我々は、ＡＩワークステーションＷＩＮＥ上にＣｏｍｍｏｎ　Ｌｉｓｐに準拠したＬｉｓｐ処理系ＡＩＰ－Ｌｉｓｐを実装し、Ｇａｂｒｉｅｌベンチマークによる評価を実施した。

評価の結果、速度性能に関しては一部に問題があるものの大部分の項目に関しては目的としていたレベルに到達していることを確認することができた。

また、今後の改良の指針となるようなデータを得ることができた。

4.　おわりに

以上、ＡＩワークステーションＷＩＮＥについて、ＡＩＰ－Ｌｉｓｐの概要及び評価結果を報告した。

現在、我々は今回の実装の経験に基づいて、プログラム開発機能の充実を目指した新バージョンの開発を検討している。

今後、よりバランスのとれたシステムの実現を目指して研究を続けていきたい。

参考文献

［1］ G.L.Steele Jr.,"Common Lisp the Language",Digital Press,1984

［2］ R.P.Gabriel,"Performance and Evaluation of the Lisp Systems",MIT Press,1985

［3］ S.Wholey,S.E.Fahman,"The Design of an Instruction Set for Common Lisp",ACM Proceedings of the 1984 Symposium on Lisp and Functional Programming",p.p.150-158,1984

［4］ S.Wholey,S.E.Fahman,J.Ginder,"Revised Internal Design of Spice Lisp",Spice Document S226,Carnegie-Mellon Univ. Dept. of Computer Science,1983

［5］ M.Hill,et al,"Design Decisions in SPUR",IEEE Computer,19(11):p.p.8-22,Nov. 1985

［6］ R.A.Brooks,et al,"Design of An Optimizing, Dynamically Retargetable Compiler for Common Lisp",ACM Proceedings of the 1986 Symposium on Lisp and Functional Programming,p.p.67-85,1986

［7］ 湯浦 他、「高速Ｃｏｍｍｏｎ Ｌｉｓｐ － ＨｉＬｉｓｐの実現」、情報処理学会第３３回全国大会、１９８６

［8］ 斉藤 他、「ＡＩワークステーションの開発思想」、第１回人工知能学会、１９８７

［9］ 宮本 他、「ＡＩＰ－Ｌｉｓｐの実現（Ｉ）」、情報処理学会第３６回全国大会、１９８８

［１０］河込 他、「ＡＩＰ－Ｌｉｓｐの実現（ＩＩ）」、情報処理学会第３６回全国大会、１９８８

［１１］五十嵐 他、「ＡＩワークステーション（ＷＩＮＥ）の改良」、情報処理学会第３７回全国大会、１９８８