記 号 処 理　41－6
（１９８７．６．２６）

３２ビットアドレス向きＬＩＳＰ処理系の実現

橋本ユキ子　内田誠二　小川雄司　高橋哲夫
日本電気株式会社　基本ソフトウェア開発本部

東京大学和田研究室で、ＭＣ６８０１０上に作成された高速なＬＩＳＰ処理系Ｕｔｉｌｉｓｐ（Ｕｎｉｖｅｒｓｉｔｙ　ｏｆ　Ｔｏｋｙｏ　Ｉｎｔｅｒａｃｔｉｖｅ　ＬＩＳｔ　Ｐｒｏｃｅｓｓｏｒ）を３２ビットアドレスをもつＭＣ６８０２０上に移植した処理方式について述べる。

ＭＣ６８０１０では、アドレスバスが２４ビットであるため、この上に作成されたＵｔｉｌｉｓｐは、ポインタデータ内にアドレス表現とは別にタグを設けることができる。このＵｔｉｌｉｓｐを４Ｇバイトのアドレス空間を扱えるようにするため、ポインタデータ内のアドレス表現の一部を直接タグとして使用する方式を考案し、処理系作成、評価を行った。

IMPLEMENTATION OF LISP ON 32-BIT-ADDRESSING CHIPS

Yukiko HASHIMOTO, Seiji UCHIDA,
Yuji OGAWA and Tetsuo TAKAHASHI
Basic Software Development Division, NEC Corporation
1-10, Nisshin-cho, Fuchu-shi, Tokyo 183, JAPAN

This paper describes an implementation of Utilisp※ on MC68020 using 32 bit addressing. This version originates the MC68010 version of Utilisp designed and maked at the University of Tokyo, Wada Laboratory.

Because MC68010 has 24 bit address-bass, it is possible to use the left-most 8 bits of a pointer data as a tag field. In case of MC68020, we can not provide a tag field in a pointer data in the same way as MC68010. The authors will propose the way to recognize every lisp object with some of middle bits of the address expression.

※ University of Tokyo Interactive LISt Processor

1．　はじめに

近年３２ビットアドレスを持ちアドレッシング範囲が４Ｇバイトのアーキテクチャを持つコンピュータが増えている。このようなマシン上にＬＩＳＰ処理系を実現しようとした場合、必ず問題となるのがタグの扱いである。ＭＣ６８０１０のようにアドレッシングの範囲が２４ビットである場合には、ポインタデータ内にアドレス表現とは別にタグを設けることが多く、そうして実現されたＬＩＳＰ処理系では４Ｇバイトの空間を扱うことは不可能となる。３２ビットアドレスで表現できるアドレス空間を扱えるようなＬＩＳＰ処理系の実現方法としては、ポインタデータとは別にタグ領域を設ける方法やポインタデータのさすオブジェクトデータ側へタグを持たせる方法など幾つかの方法が考えられる。本稿ではポインタデータ内にアドレス表現とは別にタグを含める方式と同等の効率を維持することを目的として、東京大学和田研究室でＭＣ６８０１０上に作成された Utilispを３２ビットアドレッシングをもつＭＣ６８０２０上へ移植した際の処理方式について報告する。以下従来方式としてはＭＣ６８０１０上の Utilispを基に説明を行う。

2．　３２ビットアドレシングのための技法

2．1　ポインタ形式とタグ

３２ビットアドレッシングが可能なＬＩＳＰ処理系を実現する場合、ＭＣ６８０１０上の Utilispで実現されているようにポインタデータ内にアドレス表現とは別にタグを設定することはできない。この方式でＬＩＳＰ処理系を３２ビットアドレス空間上に実現した場合はＬＩＳＰ処理系がタグ領域を１バイトとすると１６Ｍバイトより大きなアドレスをもつ場所へローディングされない保証が必要であり、またアドレッシングのときには必ずタグクリアが必要となってしまう。そこで本処理系ではアドレス表現自身の一部を直接タグとして使用する方式を採用した（以下本方式をアドレスタグ方式と呼ぶ）。

この場合アドレス表現のどの部分でもタグとして使用することができるが　ＭＣ６８０２０ではワード（１６ビット）を扱う命令が豊富であり実行時間もロングワード（３２ビット）を扱う場合より速い。そこでこれを有効に利用するため図１に示すようにポインタデータの第１１ビットから第１５ビットまでの５ビットを基本的にタグとして使用している。

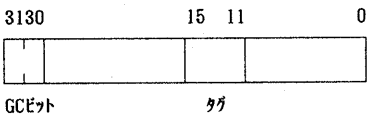

図１．　ポインタデータの形式

ＧＣはmark&sweep法を使用しており、ＧＣビットは２ビット必要なため第３０、３１ビットを使用する。本システムではユーザ空間が１Ｇバイトに制限されているためＧＣビットとして第３０、３１ビットを使用しているが、ＬＩＳＰ処理系を４バイト境界に整列して作成しておくことにより第０、１ビットを使用することができ事実上４Ｇバイトのアドレッシングが可能となる。

Utilisp本来の高速性を損なわないように設計を行うためタグの決定についても従来の２４ビットアドレス空間上の Utilispと同じ効率で各タグが扱えるように考慮する必要がある。タグ決定のポイントは　各データタイプ中出現頻度の高い順に領域の割合を定めることとアクセス頻度の高いデータをより高速に扱えるように工夫をすことである。

データタイプとタグの関係を表１に示す。

ﾃﾞｰﾀﾀｲﾌﾟ	ﾀｸﾞ（16進）
list	00 ～ 78
symbol	80 ～ 88
flonum	90 ～ 98
vector	A0 ～ A8
code piece	B0 ～ B8
string	C0 ～ D8
reference	E0
bignum（用）	E8
extra	F0
stream	F2
undefined	F4
fixnum	F8

表１．　データタイプとタグの関係

listは各データタイプ中で最も大量に割り付けられるオブジェクトであり、listか atom かの判定回数も非常に多い。そのためlistかどうかの判定は従来通り正負の判定で可能とし、リスト領域もヒープ領域全体の１／２を占めるようになっている。また、symbol, fixnumのアクセス頻度も高いためやはり高速に判定が行えるように定められている。　stream,undefined, extra の各タイプについては必要最小限の領域が確保できればよいので更に細かく領域を分けポインタデータの第７ビットから第１０ビットもタグとして使用することにする。

ＭＣ６８０１０版とＭＣ６８０２０版におけるタグ判定法の代表例を表２に紹介する。

ﾃﾞｰﾀﾀｲﾌﾟ	68020 判定法	68010 判定法
list	mov.w R1,R2 bpl yes	mov.l R1,R2 bmi yes
symbol	cmp.w Rn,SY bgt yes (SY=0x8fff)	cmp.l N,Rn bge yes
string	mov.w R1,R2 and.w 0xe000,R2 cmp.w 0xc000,R2 beq yes	mov.l R1,R2 swap R2 and.w 0xff00,R2 cmp.w 0x4000,R2 beq yes
fixnum	cmp.w 0xf800,Rn bcc yes	cmp.l maxfix,Rn bls yes

表２．　タグ判定法

２．２　メモリ構成

２．１図１で示したようにポインタデータ内の第１５ビットから第１１ビットをタグとして使用した場合ヒープ領域は６４Ｋバイト毎のセグメントに分割され更に各セグメントがタグの種類に従ってＬＩＳＰの全データタイプに分割されることになる。

ＭＣ６８０２０版Uitlisp のメモリ構成を図２に示す。（次頁）

本方式では当然のことながら predefined objectも各々決まったタグをアドレスとしてもつように配置されなくてはならないためメモリへロードされた時点で第２図のように展開されるように、また無駄な領域が作られることのないよう調整を行っている。
本方式における問題点として　例えばＬＩＳＰで記述したアプリケーションが floating データは使用していないがシンボルの数が非常に多いというように１セグメントのデータタイプ毎の分割の割合がアプリケーションのオブジェクト使用量に合っていない場合が考えられる。仮想記憶管理を備えているＯＳ上では全く使用されていない領域は実メモリに割り付けられることはないのでメモリ使用効率はあまり問題にはならないが、それでも実メモリが少ない場合は仮想記憶空間を圧迫することにはなる。それをできるだけ解消し最適な構成のＬＩＳＰ処理系を作成するためにタグの再編成方式を考案した。（詳細は「４．タグ再編成方式」参照）

２．３　データオブジェクトの特長

本節ではアドレスタグ方式の採用により元の Utilisp処理系とは異なった形式で実現しているデータオブジェクトについて説明する。変更が加えられたデータは fixnum,reference,extra である。

（１）　fixnum
従来fixnumはオブジェクトを持たずポインタデータの代わりに直接値をタグとともに格納している。アドレスタグ方式を実現するとポインタデータがさすデータ側にfixnum値を格納しなければならなくなるがそれではfixnumをアクセスする際常に間接参照となり明らかに効率が低下する。またfixnum領域も大量に必要となってしまう。そこで従来と同様にポインタデータの代わりに値を格納する手段としてfixnumは次の構造をとっている。

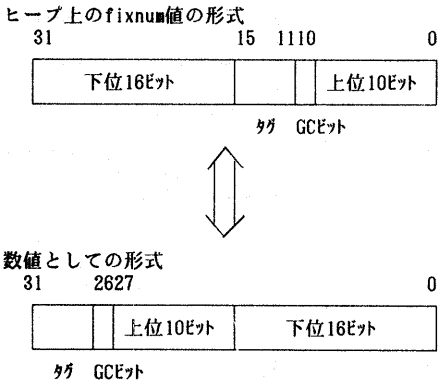

図３．　fixnum形式

ヒープ上のfixnum値の形式と数値としての形式の間の変換はＭＣ６８０２０上のSWAP命令１命令で行うことができる。（間接参照よりSWAP命令を使用したほうが高速である。）
メモリ領域には図２に示す通り各セグメントにfixnum領域２Ｋバイトが確保されている。この領域はfixnumオブジェクトが存在していないため実際に使用されることはない。そこでこの領域はＩ／Ｏバッファとして現在利用している。

（２）　extra
図２のようなメモリ構成をとっているため可変長オブジェクト　ストリング，ベクタは各々のサイズを越えてアロケートすることができない。そこでこれを越える大きさのストリング，ベクタを作成したい場合 extraオブジェクトと呼ぶ別のデータをアロケートしストリング，ベクタの実体へのポインタを格納する。

例としてXが１０Ｋバイトのストリング　”ＡＢＣ・・・”を値とするシンボルであるときのエキストラオブジェクトの構成を図４に示す。

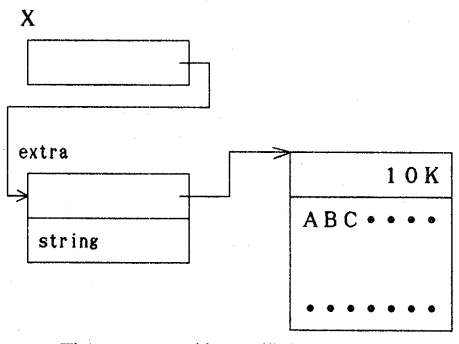

図４．　extraｵﾌﾞｼﾞｪｸﾄの構成

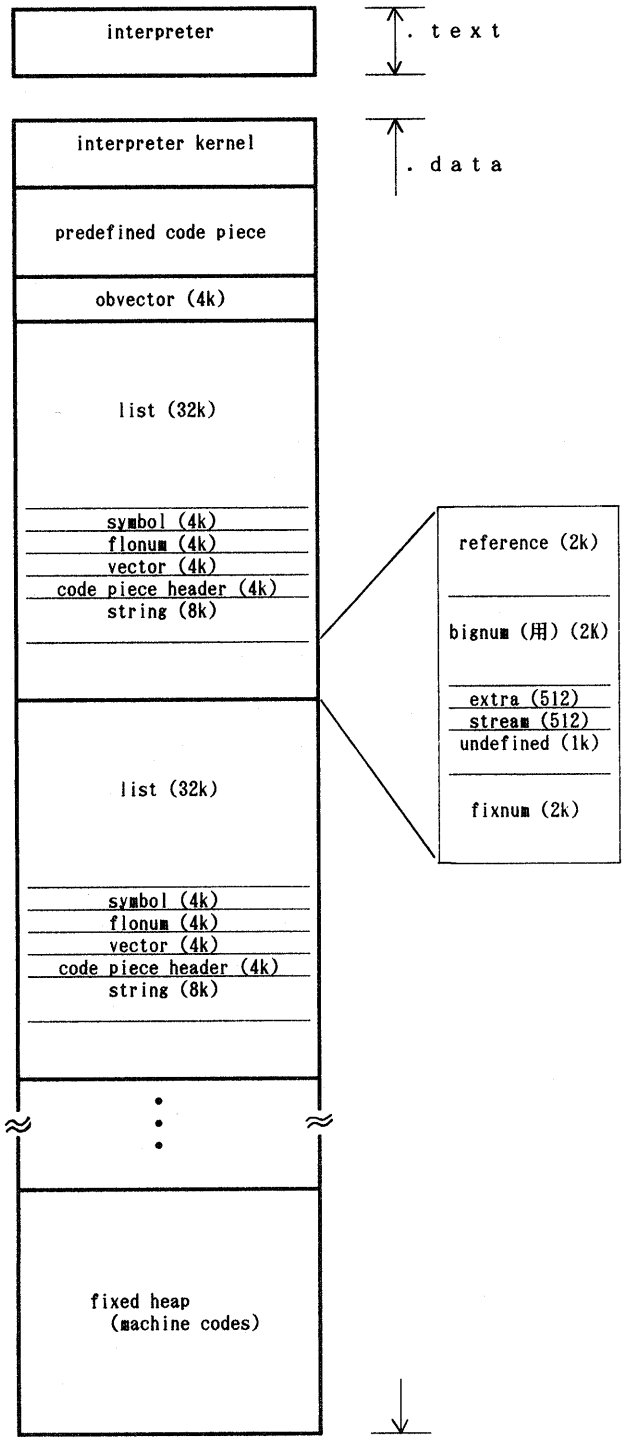

図2. Utilisp のメモリ構成

（３）　refernece

従来 referenceはオブジェクトを持たずポインタデータは直接ベクタ要素をさしていた。本方式においては　reference オブジェクトが存在しベクタの要素をさすポインタとベクタの先頭をさすポインタから構成されている。

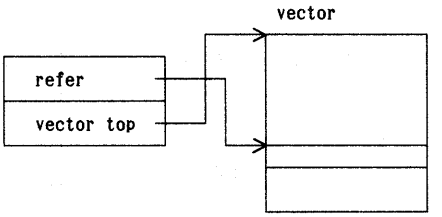

図５．　refernceｵﾌﾞｼﾞｪｸﾄの構造

３．ガーベージコレクション

本処理系ではヒープ領域は図２に示すようにデータタイプ別に分類されているため、各領域の管理方式は固定長データ、可変長データにより異なっている。可変長オブジェクトは各領域の先頭から順に必要な大きさを割り付けているが固定長オブジェクトは未使用領域をフリーチェーンによって管理している。各データのセル自身をフリーチェーンとして使用しており当該セルから次のセルへの相対距離を格納している。

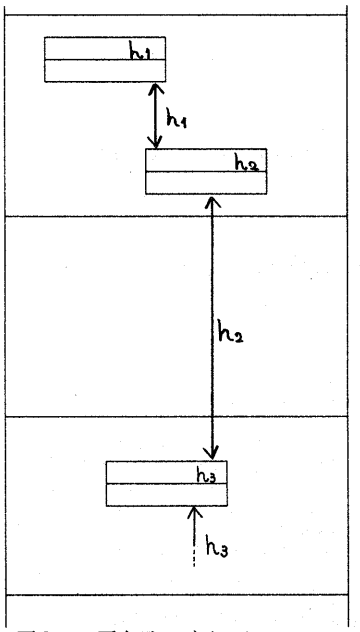

図６．　固定長オブジェクトの
管理方式

ＧＣ（ガーベジコレクション）はこのヒープ領域管理に従い固定長オブジェクトと可変長オブジェクトで処理が異なる。　固定長オブジェクトのＧＣはゴミとなったセルをフリーチェーンに繋いでいく処理を行い、可変長オブジェクトのＧＣはmark&sweep法により使用中のオブジェクトを各ヒープ領域の先頭に寄せ集める。

ＧＣによって使用するビット位置は次のようになっている。fixnumの場合は第３０、３１ビットはfixnum値を表しＧＣビットとしては使用できないため第１０ビットを使用している。

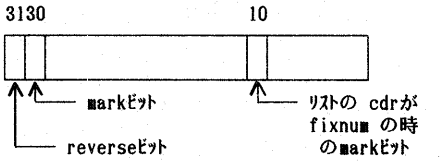

図７．　ＧＣビット

ＧＣ処理手順は次の通りである。

（１）マーキング処理

現在使用中のオブジェクトのマークビットをオンにする。ただしリストの cdrがfixnumのときは cdr 部の第１０ビットがＧＣビットであり０のときがオン、１のときがオフである。（fixnumではタグとＧＣビットが全て１であれば負の数のときにタグの設定が不要になる。）またシンボルオブジェクトは pname部にマークをしその他のオブジェクトはヘッダ部にマークをする。

（２）ポインタの逆転処理

既定義のオブジェクト、スタック等から参照されいているヒープ領域中の可変長オブジェクト（ストリング、ベクタ）へのポインタを逆転する。

（３）未使用オブジェクトの回収

ａ．固定長オブジェクト

各オブジェクトのヒープ領域を走査しフリーチェーンを作成する。使用中オブジェクト内にストリングまたはベクタをさすポインタがあれば逆転する。

ｂ．可変長オブジェクト

各オブジェクトのヒープ領域を走査し使用中オブジェクトは領域の先頭へ寄せ集める。このとき逆転ポインタをオブジェクトの移動先へ戻す。

本方式のＧＣは固定長オブジェクトはフリーチェーンで管理し移動をしないため従来のmark&sweep法と比べると逆転ポインタ処理がかなり省略されておりＧＣは高速になっている。実際に逆転ポインタ処理を行うのはストリングとベクタだけである。また　従来のmark&sweep法ではヒープ領域に対し順方向と逆方向の２度の走査処理が必要であった。本方式ではベクタ自身がポインタをもっているためコンパクションをするには２回の走査処理が必要となるが、他のオブジェクトは１度の走査で済ますことができる。

本ＧＣでは固定長オブジェクトはヒープ上に寄せ集められず分散してしまうが　アドレスタグ方式によりヒープ領域は６４Ｋバイトのセグメントに分割されており本来ローカリティはあまり上がらない。従って高速にＧＣが行えるほうがメリットが高い。

4．タグ再編成方式

本節では、ヒープ領域内のメモリ構成を変更する方式について述べる。

4．1　背景

第2節で述べたように、本処理系ではアドレスデータの内部の数ビットをタグとして使用している。　したがってヒープ領域は各64kバイトの領域に分割され、図2に見るようにそのそれぞれの領域がさらにタグの値に対応する小領域に分割される。　すなわちタグの編成がそのままヒープ領域の使用区分となっている。　最も標準的と思われる編成をもって既定値としているが、特定の種類のデータを多用するアプリケーションプログラムを効率よく動作させるためには、多用される種類のオブジェクトを格納する領域がより大きくとれることが必要であり、そのためにはタグの編成を変更することが不可欠である。

4．2　再編成の対象となるデータ領域

本処理系において、list,fixnum,symbolの3種のデータは特に高速に判定されるタグに対応する領域に格納されており、セグメント内の位置を変更することは困難である。一方、たとえばstreamオブジェクト用の領域は、オブジェクトの総量に上限があり、既定値のタグ編成において全てを使用可能としており、特に再編成の対象とする必要はない。　再編成の対象としないデータ属性およびその理由は次の通りである。

（1）　list
　タグの符号のみで判定が行われており、これが最高速のタグ判定であることによる
（2）　fixnum
　判定の高速化のため、各セグメントの最終位置をあらわすタグがついており、最終位置であることを変更することはできない。また、実際にこのタグに対応する領域は2kバイト長のI／Oバッファとして使用されているため、サイズも変更することができないことによる。
（3）　streamおよびundefined
　増減がないため。
（4）　extra
　さほど多用されず、現行のままで十分であると思われるため。

上記以外のデータに対するヒープ領域、すなわち図8の領域に対して再編成処理を行う。

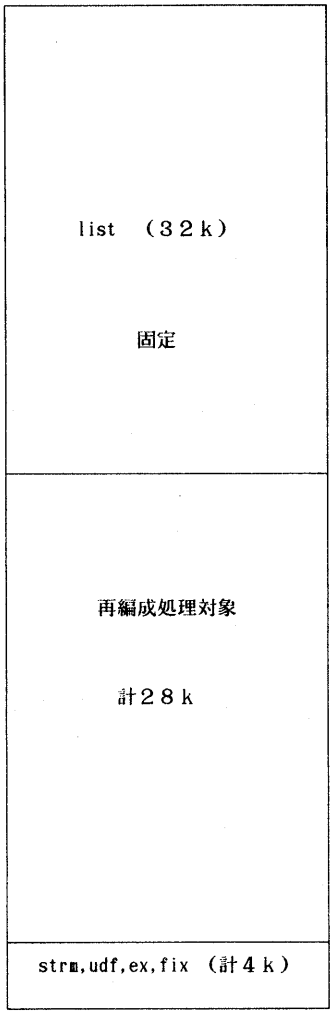

図8　再編成処理の対象となる領域

4．3　再編成を行う契機

utilisp コマンドの通常の実行形態は、コマンドのオプションパラメータとしてヒープ領域の大きさ等が入力され、十分な大きさの領域が割り付けられた後、インタプリタ本体に制御がわたり、その初めの処理としてヒープ領域の初期化すなわちメモリ管理のためにヒープ上にチェーンを作成する処理を行い、システムファイルをロードした後、read-eval-print ループに入るという形態をとっている。　ヒープ領域等の拡張割り付け自体はタグの編成とは無関係であり、また、ヒープ領域の初期化処理はヒープが拡張されているか否かにはよらず、常にタグの編成に依存する定数を用いて行われる。　これらから、タグの再編成処理の契機は、utilisp コマンドのオプションパラメータによって与えられる。　契機が与えられなかった場合には

そのまま通常の実行形態をとる。　タグの再編成を行ったutilisp の立ち上げは４．５で述べるが、このことによりインタプリタ自身は自分が既定値のutilisp であるのかそれとも再編成をされたutilisp なのかを知らずに動作する。　すなわち、インタプリタ本体に制御が渡った後は、タグの再編成の有無による処理速度の変化は全くない。

４．４　処理概要

４．２で述べた再編成処理対象領域は、既定値のタグ編成において２ｋ，４ｋあるいは８ｋバイトの小領域７つからなり、合計２８ｋバイトである。　そのうち、symbolのタグ判定においてはレジスタとの比較が１回行われ、その他のタグ判定においては即値とのAND および即値比較が１回行われる。

即値を書きかえるのみでタグチェックを更新するためには、編成後の各領域の大きさは２ｋ，４ｋあるいは８ｋバイトのいずれかでなくてはならない。　合計２８ｋバイトの領域を７種のデータ領域に分割する可能な組み合わせは、大きさの順に、ｋバイト単位で

（１）　８，８，４，２，２，２，２
（２）　８，４，４，４，４，２，２
（３）　４，４，４，４，４，４，４

の３種であり、データ属性の組み合わせにより、２１１種類の異なった編成が作成可能である。

タグの再編成処理は、

（１）　新しいタグ編成の決定
（２）　タグ値に依存する大域データの更新
（３）　タグチェックに関するマスク値、タグ値の即値データの更新
（４）　タグ値による多分岐用テーブルの更新
（５）　predefinedオブジェクトに対する参照の再解決
（６）　再編成済みutilisp をファイルに出力

の６つの処理から構成されている。

新しいタグ編成の決定は対話的に行われる。　利用者は各データ属性およびそれらの領域サイズを入力する。　システムはそれらの値に基づき可能なタグ編成を決定し、利用者の確認を得る。　続く４つの更新処理に対しては、書きかえるべき即値のあるアドレス等をあらかじめ持っている必要がある。　これはLAP 展開において、書きかえられる可能性のある箇所にラベルを出力し、同時に別のファイルにラベル名からなる表を作成し、utilisp 作成と同時にアセンブル／リンクすることにより得る。　この表を格納した領域はutilisp 本体が動作するときには不要であり、クリアされた後、ヒープの一部として使用される。　新しいタグ編成が決定されれば、上記の表を用いて処理（２）－（４）は容易に行われる。

処理（５）に対しては、再編成対象となるヒープ領域を２ｋバイトごとのブロックで管理し、各ブロックへの参照をLAP 展開時に表にすることにより、参照の再解決を行う。

最後に処理（６）において、タグの再編成されたutilisp をファイルに出力することにより、タグの再編成処理を終了する。

４．５　再編成されたutilisp の立ち上げ

４．３で述べたように、utilisp コマンドが起動されると、既定値のutilisp がロードされる。　タグの再編成を行ったutilisp を使用する旨のオプションパラメータと、前項で出力されたファイル名をコマンドの引数に指定することにより、そのファイルの内容が既定値のutilisp に上書きされ、utilisp 本体の初期化ルーチンに制御が渡る。以後、タグの再編成されたutilisp が動作する。

５．性能評価

本処理系はＥＷＳ４８００（UNIX SYSTEMV）上に作成した。表３にＬＩＳＰコンテストの幾つかのベンチマークの測定結果を示す。従来の２４ビットアドレスをもつ Utilisp処理系と比べほぼ同等の性能を上げており Utilispの高速性は維持されているといえる。

（MS）

ベンチマーク	インタプリタ	コンパイラ
TARAI-4	2108.3	196.6
LIST-TARAI-4	3733	1033
STRING-TARAI-4	4434	2384
SEQ-100	91.51	48
BITA-5	25.83	5.16
TPU-7	2150	416
BOYER	141366	18232

表３．　第３回ＬＩＳＰコンテスト測定結果

アドレスタグ方式を採用しているためデータのローカリティが上がらないので大きなアプリケーションを実行した場合仮想記憶領域と実メモリの間のスラッシングの問題が非常に心配であったが、使用されていない領域は実メモリに割り付けられることはないく、ワークステーションで実現したため実メモリがふんだんに使用可能となり、心配していた問題は起こらなかった。

６．おわりに

３２ビットアドレス空間上に高速なＬＩＳＰ処理系Utilispを実現した方法について述べた。ポインタデータのアドレスの一部をタグとして使用しているため幾つかの欠点もあるが、おおむね満足のいく処理系が作成できた。拡張機能として　他言語呼び出し機能（Ｃ，ＦＯＲＴＲＡＮ）、日本語処理機能を追加している。今後の課題としては　コンパイラの性能改善、プログラミング環境の充実などがある。

最後に、本処理系を作成するにあたりＭＣ６８０１０上の Utilispを提供して頂き、方式設計の際有益なご意見を賜った東京大学和田教授ならびに研究室の方々に深く感謝致します。

参考文献

(1) 近山隆：“Utilisp システムの開発”，情報処理学会論文誌，Vol.24 No.5,Sep.1983

(2) ＭＯＴＯＲＯＬＡ：
"MC68020 32-Bit Microprocessor User's Manual", Second Edition

(3) 近山隆：“UTILISP MANUAL”，
TECHNICAL REPORTS,METR 81-6,September 1981