マイクロコンピュータ 48－10
計算機アーキテクチャ 69－10
（１９８８．１．２１）

32ビットAIチップELISのアーキテクチャ

渡辺和文*、石川 篤*、山田康宏**、日比野靖*
*NTT研究所、**NTTインテリジェントテクノロジ株式会社

知能処理研究において広く使われているリスト処理言語LISPを高速に実行する専用のLSIチップELIS－VLSIを設計、完成した。ELIS-VLSIは、スタックアーキテクチャ、タグアーキテクチャのほか、リストセルのたどりと生成、文字列の照合、分解、合成の操作を効率よく実行するための専用のアーキテクチャをその特徴とする。チップは、トランジスタ数80Kで、2ミクロンC-MOS 2層配線技術を用いて15mm角に収め、16.67MHzのクロック周波数で動作する。ELIS－VLSIは、マイクロプログラム制御方式を採用しており、LISPインタプリタは完全にマイクロプログラム化され、DEC-2060上で、コンパイラを介して実行した速度と同等の性能を得た。

The Architecture of the ELIS 32-bit AI-chip

Kazufumi WATANABE*, Atsushi ISHIKAWA*,
Yasuhiro YAMADA**, and Yasushi HIBINO*

*NTT Laboratories
1-2356 Take Yokosuka-shi Kanagawa 238-03 Japan

**NTT Intelligent Technology
223-1 Yamashita-cho Naka-ku Yokohama-shi Kanagawa 231 Japan

The ELIS 32-bit AI-chip, ELIS-VLSI, has been developed. It is designed for a fast LISP interpreter, which is important for productive program development in a research field of artificial intelligence. ELIS-VLSI's architectural features are stack architecture, tagged architecture, and a dedicated datapath for following/creating list data and for string manipulation. The chip, fabricated using 2-micron CMOS double metal layer process, contains 80K transistors in 15mm die size and operates at the clock rate of 16.67MHz. The chip is controlled by microprograms. The LISP interpreter is fully microcoded and its speed is equal to the execution speed of compiled code on DEC-2060.

1. はじめに

知能処理研究の発展に伴い、プログラミング言語として、リスト処理言語 LISP が今日広く使われるようになってきた。LISP は、会話型の言語であると同時に、関数型の言語でもあることから、計算機と対話しながらアルゴリズムの検証を進めるプロトタイピングに向いた言語であることが、その大きな特徴である。

しかし、LISP はそのデータ構造が、2 進木リスト構造を持つことから、本質的に多量のメモリを消費する。このためユーザの間で資源を共有する、汎用計算機の TSS (Ti me Sharing System) の下では、大型機でさえも対話的な処理において充分な実行速度を得ることが難しい。効率の良いソフトウェア開発のためには、早い応答性能をもつ計算機システムが求められることから、LISP 専用の計算機が研究開発され [1,2,3]、専用チップも出現している [4]。

筆者らは、1978 年にインタプリタの高速化を指向した実用的な計算機、ELIS(Electrical Communications Laboratories List Processor) の設計、開発に着手した [5]。まず TTL 素子を用いてプロトタイプ機の試作を進めた。試作は 1982 年に完成し、これを用いて、LISP 言語 TAO の研究と、その ELIS 上へのインプリメントを行った。TAO のインタプリタは、全面的にマイクロコード化され [6]、汎用大型機上にソフトウェアでインプリメントされたインタプリタの速度に匹敵する性能を示し、ELIS のアーキテクチャの有効性が実証された。さらに ELIS を小型化し、かつ価格性能比の優れた知能処理用ワークステーションとして実現するために、主プロセッサ部の 1 チップ VLSI を開発、完成した [7]。LSI の設計は、1985 年に着手し、1986 年にチップを完成した。

本報告では、ELIS アーキテクチャの特徴、LSI の設計、およびその性能について述べる。

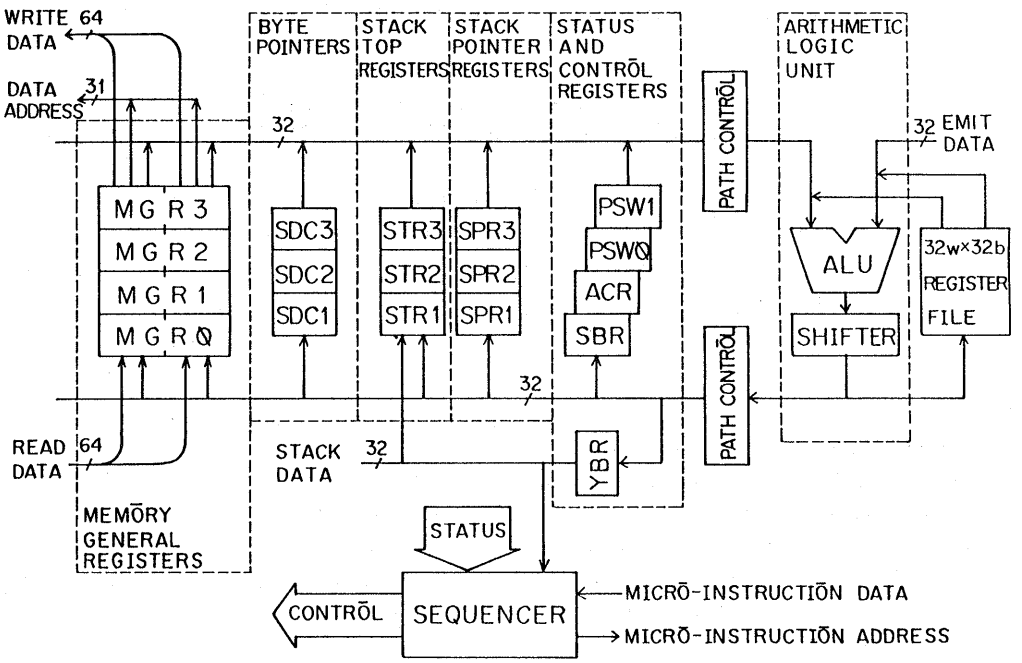


図 1. バス構成

2. データパスの構成と制御

2.1 内部バス構成

ELISのバス構成を図1に示す。内部バス幅は32ビットである。ALUは32ビットデータまたは24ビットデータに対して算術論理演算を行う。24ビットの演算では、第3.1節で述べるタグ付きデータのポインタデータを扱う。このときタグビットは、そのまま出力バスに出力されるか、クリヤされる。 SHIFTERはデータの1ビットシフトを行う。 REGISTER FILEは、32語(32ビット/語)のALU汎用レジスタである。 PATH CONTROL はバイト/半語(16ビット)のデータの切り出しと挿入を行う。 PSWはプロセッサ状態レジスタ、ACRはI/Oレジスタ、またSBRはスタック境界レジスタである。YBRは毎サイクルごとに演算結果を格納するレジスタである。 SPRはスタックポインタレジスタ、STRはスタックトップレジスタである。 SPRとSTRは、チップ外部に置かれたスタック専用のメモリと共にプッシュダウンスタックを構成する。 MGRはメモリ汎用レジスタと呼ばれ、メモリデータレジスタ、メモリアドレスレジスタ、そしてALUの汎用レジスタのいずれにも使うことができる。SDCは、一種のカウンタレジスタであって、MGR内のバイト位置を指定するバイトポインタである。メモリとの間のデータ幅は64ビットと内部バス幅の2倍とし、汎用プロセッサで通常2回のメモリ操作を要するリストセルの読みだし/書き込みを一回のメモリ操作で可能としている。

2.2 マイクロ命令

2.2.1 命令形式

データパスは、外部におかれた制御記憶から送られてくるマイクロ命令によって制御される。マイクロ命令は64ビット長の水平型である。その形式を図2に示す。 TYPE-<I>,<II>,<III>ともに、ALUの演算を3アドレスモードで行う。 A-Bus Source、 Y-Bus Destinationとして前節で述べたレジスタをすべて指定できる。B-Bus Sourceは、TYPE-<I>,<II>では32語のALU汎用レジスタと小整数(-16から15)の発生に用いる。TYPE-<III>では 32ビットの整数を発生し、これをB-Busから入力する。 TYPE-<I>命令により、メモリ操作が、またTYPE-<II>命令により、

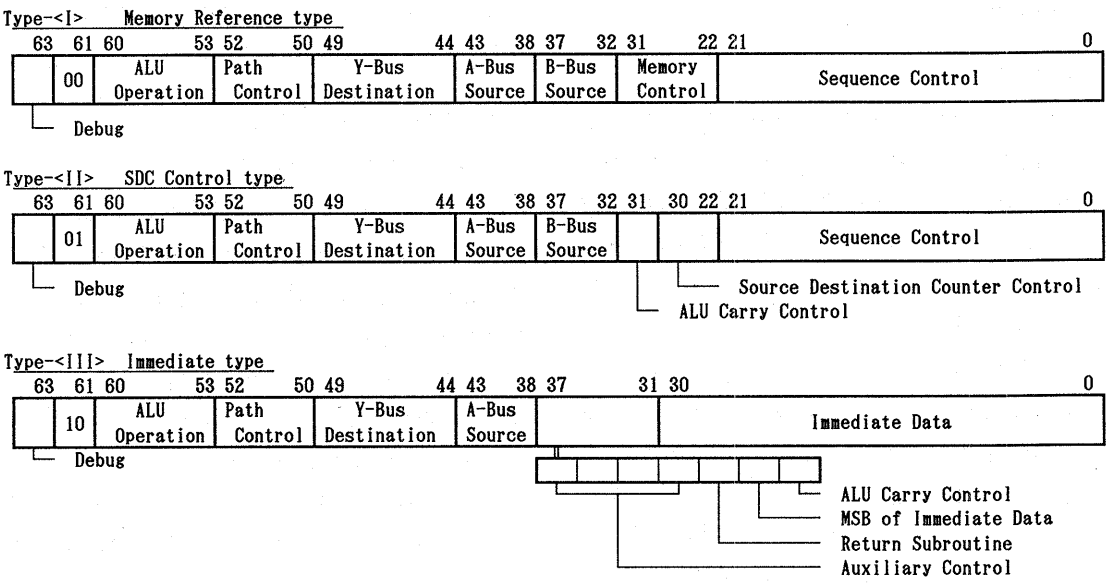


図2. マイクロ命令形式

3本の SDCのカウントアップの制御がALUの演算と並行して可能である。Memory Controlフィールドは、4本のMGRからアドレスレジスタと、データレジスタを選択し、メモリ操作の種別を指定する。

スタックのプッシュ/ポップは、スタックプッシュをA-Bus Sourceにスタックポップを Y-Bus Destinationに指定したときに行われる。

Sequence Controlフィールドは、次に実行するマイクロ命令のアドレスを発生するために用いられ、14ビットの次マイクロ命令アドレスフィールドと分岐の種別を指定するフィールドから成っている。アドレスフィールドをタグビット、YBR、ALUの演算結果フラグなどの分岐ソースで修飾することによって、多方向分岐を行う。

ELISは、分岐の種類の豊富なことが一つの特徴である。具体的な分岐命令の種類については文献[8]に詳しい。TYPE-<III> 命令は、Sequence Controlフィールドを持たない。このためTYPE-<III> 命令の次の命令のアドレスは、+1されたものとなる。

TYPE-<IV>は、図2には示していないが、マイクロ命令レベルでの割り込みが発生した際のレジスタの退避と回復のために用い、基本的にはTYPE-<II>と同じ形式である。

2.2.2 実行制御

1マイクロ命令は通常180nsec.で実行が完了する。分岐を行う場合には、分岐ソースを直前の命令の実行結果から取るか、分岐命令自身の実行結果から取るか、どちらかを選択できる。後者の場合をカレントジャンプといい、その実行は1サイクルよけいにかかり、360nsec.かかる。

ELISでは、スタックアクセスとメモリアクセスとをALUの演算と出来るだけ並行化させ、アクセス待ちを演算の裏に隠して実行できるようにハードウェアを組んでいる。実行の遅れはスタックアクセス、またはメモリアクセスの完了待ちを必要とする場合に発生するが、これはマイクロ命令のシーケンスに依存する。簡単にまとめると:

(1) 連続したスタックのプッシュまたはポップは、サイクルの遅れはなく、180nsec.で実行される。

(2) スタックへの書き込み(プッシュを含む)のち直ちにポップを行うと、ポップ命令は60nsec. 実行が遅れる。

(3) スタックポインタを書き換えたのちにただちに、スタックを読み出すと、120nsec.遅れる。ただし3本のスタックの内、異なったスタックを用いる場合には、遅れは60nsec.である。

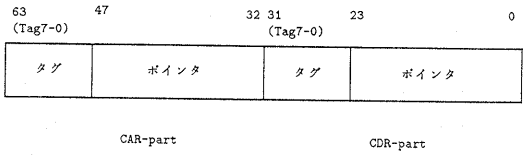
(4) メモリ操作命令実行後、メモリ操作を行おうとした場合、またはメモリからのデータを演算のソースに指定した場合(メモリリード命令を指定した後に)に、メモリアクセスが終了するまで、60nsec.の整数倍で実行が遅れる。

上記(1)から(3)までは、カレントジャンプを伴っているときには遅れは発生しない。これら実行の遅延条件の検出は、すべてハードウェアが行うので、マイクロプログラマは、実行タイミングを気にすることなくプログラミングができる(ただし実行速度を気にしなければである。)。したがってマイクロプログラミングは、マイクロ命令の形式がアセンブリ言語に近いこともあって、容易である。インタプリタを全面的にマイクロコード化するうえで、このことは効率的なプログラミングに大きく結びついている。

3. ELISアーキテクチャの特徴

3.1 タグアーキテクチャの採用

LISPではデータタイプの宣言が無いため、実行時にデータの型を逐一判定しなければならない。判定処理を高速に行うために、データに属性を示す情報をつけておき(これをタグという)、ハードウェアでタグのビットを直接参照/操作できるようにしている。リストセルの形式を図3に示す。タグはポインタタグであり、ポインタで指されているデータの型を表す。このタグビットは、TAOの大きな特徴であるマルチパラダイム言語(LISP、オブジェクト指向言語、論理型言語の三つ言語が、S式という、統一した形で記述でき



タグのビット割当:

Tag7　ガーベジマーキング
Tag6　データタイプの拡張
Tag5　リストたどり可 (car-cdr-able)
Tag4-0　データタイプ

図3. リストセル形式

る) で書かれたS式を実行する際に、言語間の処理速度の相違をなるべく少なくするように、活用されている。データ型のデコードは、6ビットのデータタイプフィールド (Tag5-0) による多方向分岐命令1命令で実行できる。この多方向分岐命令を 1回から3回行うだけで必要な処理ルーチンを呼び出せる。

さらにELISでは、このタグビットとメモリの操作制御との間にハードウェア上、密接な関係を持たせている。メモリ操作においては、特殊な命令を設け、Tag5 ビット (car-cdr-able) の内容によって実際のメモリへのアクセスを抑止できるようにした。この機能を用いると、リストの末尾の判定、CAR/CDR オペレーション禁止の検出において、Tag ビットによる分岐と同時にメモリアクセス起動がかけられるため、コーディングの際、ステップ数を削減でき、処理速度の向上に繋がっている。

3.2 大容量のハードウェアスタックのサポート

LISP は関数型の言語である。LISP のプログラムは入れ子構造をもった関数定義によって構築される。関数に引数を与えて計算を行い、次々と入れ子を解きほぐしながら計算を進めるが、この処理を効率良く行うために通常プッシュダウンスタックを多用する。ELISでは、スタック操作を高速化するためにスタックをハードウェア化した結果、内部にある汎用レジスタと同じ速度でスタックをアクセス出来る。さらに容量を32K語 (32bit/語) と大容量化することにより、次のようにマルチユーザ / マルチプロセスを効率良く実現している。

TAOでは8台までのユーザ端末操作および128個までのプロセス走行を可能としている。これらユーザ / プロセスの処理を進めるために必要な環境情報 (変数束縛情報、関数呼び出しのための制御情報など) を可能な限りスタックに常駐させ、ユーザ / プロセスの切り替え時にスタックの入れ替えをなくし、高速の切り替え (40 マイクロ秒) を実現する。スタック領域が不足した場合にはスタックのデータの入れ替えを伴うが、メモリとスタックへの同時アクセスが可能なため、入れ替えに伴うオーバヘッドは小さい (1.6 ミリ秒)。 プログラミング容易化を狙い、スタックポインタは3本設けた。 3本の間には機能的な差は全くないので、その使用目的をプログラマが自由に決めることができる。スタック領域は16の2K語のブロックに分割され、3本の境界はSBRによって自由に設定出来る。スタックオーバフローは自動的に検出される。

3.3 メモリ汎用レジスタの概念の導入

4組のMGRは機能的にいずれも全く同一である。MGR設計の基本的な考えは、「同一の動作概念に属するものは同一の回路で実現する」ということにある。第2.1節でMGRの機能を述べたが、どのMGRをどういう目的で使用するかはプログラムの流れに応じて自由に決められる。MGRの主な使用法は、

(1) CAR-CDR レジスタ
(2) コンパイルコードのための8バイトのバッファ
(3) 文字データバッファ

である。

例えば、(1) のCAR-CDR レジスタとしての使用では、 2組のMGRをアドレスレジスタまたはデータレジスタとして、交互に切り換えて用いることにより、アドレスの無用な転送がなくなり、リスト連鎖を効率よくたどることができる。

(2)、(3) の使用については次のようになる。ひとつのMGRは8バイト長である。MGRからは半語 (16ビット) またはバイトデータを任意の位置から読みだし、また任意の位置へ書き込むことが出来る。位置の指定にはポインタレジスタSDCを用いる。SDCは

5ビット長であり、そのうちの上位2ビットでMGRを選択し、下位3ビットでそのMGRの中のバイト位置を指定する。SDCは3本あるが、いずれも機能は同じである。SDCは、1マイクロサイクルで1、2、または4、自動増加でき、modulo8、16、または32を生成できるように制御が可能である。MGRとSDCとを組み合わせることにより、MGRがあたかも連結したバイトバッファとして使用でき、可変長のバイト処理を効率よく行える。文字列の参照、照合、生成、さらにコンパイルコードの取り出しがこの機能により実現されている。

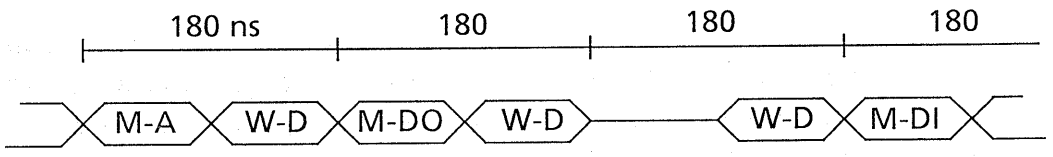
4. LSIの設計

4.1 チップの設計

ELISは、制御記憶(4Mbit)、スタック用メモリ(1Mbit)といった高速大容量のメモリを必要とする。これらをすべてチップ内に組み込むことは現在のLSI技術ではまだ困難なため、外部に市販のSRAMを用いて構成することとした。このとき問題となるのが、外部におかれたSRAMチップとの間のデータのスループットである。ELISチップのマイクロサイクルは180 nsec.　であるので、高速のSRAMを使用することを前提とした。まず制御記憶、スタック、メインメモリへのアクセスパスは独立に設ける必要がある。チップのI/Oピンは必然的に多数必要となるが、パッケージの制約条件から、ピンを多重化している。メモリデータ幅とマイクロ命令の幅は、両者とも64ビットと同一であることから、これらを多重化した。さらにメモリアドレスもこれに乗せ、ピンの削減に努めた。このタイミング関係を図4に示す。毎サイクルの後半でマイクロ命令を取込む。前半は、メモリデータの入出力とメモリアドレスの出力に割り当てている。スタックついては、これとは別に32ビットのデータピンと15ビットのアドレスピンを設けている。これにより、

表1. チップ諸元

内部演算幅	32ビット
外部データ幅	64ビット (主記憶 / 制御記憶) 32ビット(スタック) 16ビット(フロントエンド)
アドレス空間	2Gワード(64ビット / ワード)
制御方式	マイクロプログラム
クロック周波数	16.67 MHz
マシンサイクル	180 nsec.
プロセス技術	2 μm CMOS 2層Al配線
トランジスタ数	79,438
チップサイズ	15 mm × 15 mm
消費電力	1.0 W
電源	+5 V
パッケージ	208ピン PGA



M-A：メモリアドレス
M-DO：メモリデータ出力
M-DI：メモリデータ入力
W-D：マイクロ命令データ

図4. データ入出力タイミング

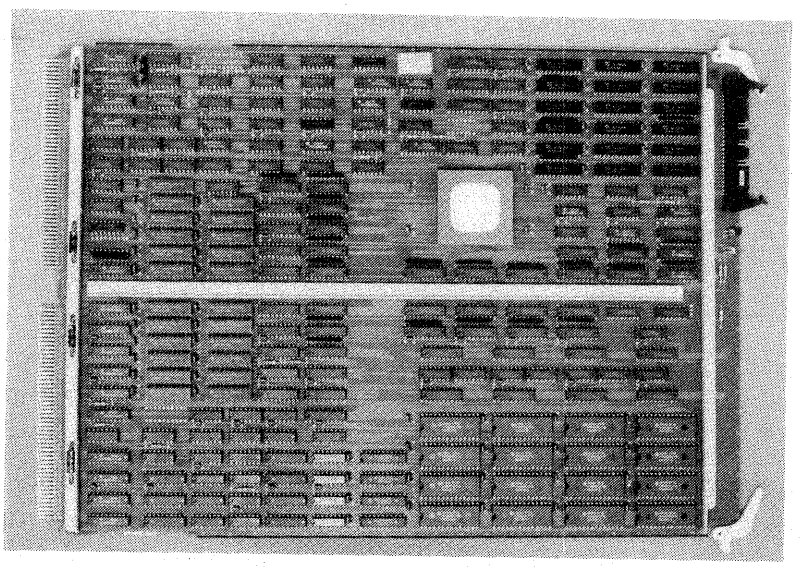


図5. CPU基板

マイクロ命令コードを含めて、チップのトータルデータスループットは、約80MByte/sec.となった。チップの諸元を表1に示す。チップ内部にはクロック発生機、シーケンサ、データバス制御回路のすべてを取込んでいるので、チップ外部にSRAMチップ、それにメモリアクセスのための若干の周辺制御回路とバストランシーバIC付けることにより1ボードのLISP専用CPU(図5; サイズ:30cmX42cm) を組み上げることができる。

4.2 レイアウト設計

LSIを新たに設計する場合、設計期間の短縮化は大きな課題である。専用のプロセッサであるからといって論理を不規則にすると、設計に多大の時間が必要となる。設計工数の点からは、規則的な設計が望ましい。 ELISを最初に試作する時点では、素子としてTTLを選択、使用したが、しかしそのアーキテクチャは、バス構成からわかるように、将来のLSI化を見越し、リスト処理に本質的な操作を単純な回路の繰り返しで実現している。例えば、4本のMGR、3組のSPR-STR対、3本のSDCは全て繰り返し構造を持っている。さらにALUそれにシーケンサの一部にも繰り返し構造がある。これらを合計すると、繰り返し部の全体に占める割合は、面積比で50%以上になる。

ELISのLSI設計は、スタンダードセル方式を用いた階層化設計により、進められた。全体の論理回路を繰り返しとなる単位に分割し、これらの論理設計およびレイアウト設計を進め、完成した単位を繰り返して配置することにより全体のレイアウトを求めた。この結果、レイアウトは約半年で終了できた。チップ写真を図6に示す。

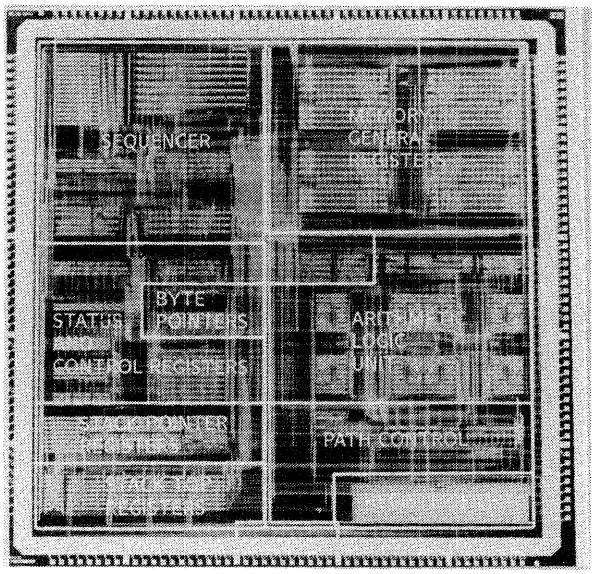


図6. ELIS-VLSIチップ

表2. ベンチマークプログラム実行CPU時間

ベンチマークプログラム	ELIS (TAO)		商用専用機-A (ZetaLisp)		商用専用機-B (Interlisp-D)		Dec-2060 (MacLisp)	
	I	C	I	C	I	C	I	C
Tarai-5	17.1	4.18	608	3.24	1856	26.9	126	4.39
Tarai-6	628	153	22366	119	67357	988	5241	157
Srev-5	.028	.010	1.07	.025	3.98	.084	.236	.008
Srev-6	.140	.041	4.34	.055	16.10	.335	.951	.032
TPU-3	.878	.041	17.8	2.77	75.7	9.05	6.10	1.22
TPU-6	4.48	2.26	98.9	11.7	421	53.8	31.5	4.89
TPU-9	.550	.232	12.2	1.43	53.9	4.62	3.60	.549

(単位: 秒)

I : インタプリタにより実行
C : コンパイル後、実行

5. 性能

ELIS-VLSIを搭載したワークステーション上でのベンチマークプログラムによる性能測定結果を表2に示す。表2は、プログラムを実行するために要したCPU時間を示している。測定では、CPU時間を計算するLISPの関数を利用した。

TARAIのベンチマークでは、関数呼び出しの際のスタックの性能が評価され、SREVのベンチマークでは、リスト操作の性能が評価される。TPUプログラム(Theorem Prover by Unit binary resolution; 400行のプログラム)は、実際の応用プログラムとほぼ同一である。ELISのインタプリタによる実行速度は、市販の専用マシンに比べて30倍から100倍速く、大きなプログラムの実行では汎用機DEC-2060がコンパイラを介して実行した速度に匹敵する[9]。ELISがコンパイラを介して実行すると、インタプリタによる実行に比較して、2倍から5倍早くなる。

6. おわりに

現在、ELISは、言語TAOがインプリメントされ[10]、新しいプログラミング環境の構築にむけ、研究開発に使われている。TAOのインプリメントにおいては、ELISのマイクロアーキテクチャを十二分にいかしたコーディングをしており、これがTAOインタプリタの高速化に繋がっている。インタプリタをより高速にするためには、より微細化された素子技術を適用し、LSIチップを高速化することが重要なテーマとなる。ELISのアーキテクチャの特徴は、単純な論理回路と大容量の高速メモリ(制御記憶、スタック)のサポートにある。この高速メモリをそのままチップ上にのせることが出来れば、マシンのサイクルタイムは、大きく短縮化されが、現在では、これは技術的に大変困難である。したがって、チップとLSIとの間のデータスループットをいかにして高めるかが、大きな課題であり、現在検討を進めている。

本報告では、ELISのプロセッサの特徴である、タグアーキテクチャ、ハードウェアスタックのサポート、複数のメモリ汎用レジスタについて述べた。またアーキテクチャの持つ繰り返し構造がLSI設計期間の短縮化に繋がっていることを述べた。

ELISは、マイクロプログラム化インタプリタにより、DEC-2060の上でコンパイラを介して実行した速度に匹敵するインタプリタ性能を得、ソフトウェア開発用マシンとして従来機を凌ぐ性能を得ている。チップのELIS-LSIの開発により、LISPプロセッサをボードに納めることが出来た。

最後に、ELISのLSI化検討段階で、論理設計、レイアウト設計に携わって頂いた武藤信夫主任研究員、島田茂夫主任研究員をはじめとする、旧電子装置研究室の諸氏に深く感謝いたします。

参考文献

[1] R.Greenblatt, T.Knight, J.Holloway, D.Monn: The LISP Machine, MIT AI Lab. Working Paper, 79, 1974.
[2] L.P.Deutsch: Bytelisp and its Alto Implementation, Proc. of 1980 Lisp Conference, p.231, 1980.
[3] 滝、金田、前川: LISPマシンの試作ーアーキテクチャとLISP言語の仕様、情処学会論文誌、20、No.6, p.481, 1979.
[4] P.Bosshart, C.Hewes, M.Chang, K.Chau, et al.: A 553K-Transistor Lisp Processor Chip, IEEE ISSCC'87 DIGEST, p.202, Feb. 1987.
[5] 日比野、渡辺、大里:LISPマシンELISの基本設計、情処記号処理研資12-15、1980.
[6] H.G.Okuno, N.Osato, I. Takeuchi: Firmware Approach to Fast Lisp Interpreter、Proc. of the Twentieth Annual Workshop on Microprogramming (Micro-20), ACM, Dec. 1987.
[7] K.Watanabe, A.Ishikawa, A.Yamada, Y.Hibino: A 32b Lisp Processor, IEEE ISSCC'87 DIGEST, p.200, Feb. 1987.
[8] 日比野、渡辺、大里: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-、情処記号処理研資24-3、1983.
[9] 奥乃: 第3回LISPコンテストおよび第1回Prologコンテスト報告、情処記号処理研資33-4、1985.
[10] I. Takeuchi, H.G.Okuno, N.Osato: A Lisp Processing Language with Multiple Programming Paradigms, New Generation Computing, No.4, p.401, 1986.