TAO/ELIS の UNIX への移植

天海良治
NTT 光ネットワークシステム研究所

TAO/ELIS は，いまから 10 年ほど前に作られた記号処理専用ワークステーションである．Lisp マシン ELIS とその上で動作するマルチパラダイム言語 TAO で構成されている．ELIS はマイクロプログラム制御のタグアーキテクチャマシンで，TAO のインタプリタは直接 ELIS のマイクロプログラムで実装されている．今回，このマイクロプログラムを C 言語に変換することで，TAO を UNIX 上に移植した．パーソナルコンピュータのオペレーティングシステム FreeBSD で稼働している．Pentium II 300MHz の計算機の上で，オリジナルの TAO/ELIS の 2.5 倍以上の性能が得られた．

TAO/ELIS on UNIX

Yoshiji Amagai
NTT Optical Network Systems Laboratories

The AI workstation TAO/ELIS was developed ten years before. ELIS is a Lisp machine and TAO is a multi-paradigm programming language. ELIS's architectural features are tagged memory, micro programmed control and a set of dedicated registers for list processing. The TAO interpreter is fully microcoded. I have ported TAO/ELIS to the UNIX operating system by converting microprogram into the C language. The ELIS on UNIX executes TAO programs 2.5 times faster than the original hardware.

1. はじめに

TAO/ELIS は，メガセル級タグアーキテクチャの Lisp マシン ELIS[1][2] に，Lisp，論理型，オブジェクト指向を融合したマルチパラダイム言語 TAO[3][4] を実装した記号処理専用ワークステーションである．マルチユーザ，マルチプロセスの実行環境を備え，利用者に export されているものだけで 1500 を越える関数を含んだ大型言語である．ELIS は 1978 年から開発がはじまり，1982 年に試作機の完成をみた．その後，CPU の VLSI 化などを計り，1987 年に製品として世にでた．このときのマシンは現在でも我々の部所では現役で稼働しており，実時間記号処理カーネル TAO/SILENT[5] の開発環境としても欠かせないシステムである．

ELIS はマイクロプログラム制御方式の計算機であり，TAO の核部分は WCS (Writable Control Storage, WCS) おに収めるマイクロコードで直接記述されている．今回，このマイクロコードを C 言語に変換することで，TAO を UNIX 上に移植した．まず，2 章で ELIS ハードウェアについて説明する．3 章でマイクロコードを C に変換する方法，4 章で移植にあたっての変更点について述べる．5 章では性能について述べる．

2. Lisp マシン ELIS

2.1 ELIS の構成

ELIS のブロック図を図 1 に示す．

ELIS の内部バスは 32 ビットである．ALU は 32 ビットまたは 24 ビットのデータに対して算術論理演算を行なう．TAO は上位 8 ビットをデータタグとして使用する．24 ビットモードでは，タグ部分を保存またはクリアしつつポインタ部分の操作が可能となる．YBR は ALU 結果を保持する内部レジスタである．MGR はメモリ多目的レジスタであり，メモリデータレジスタ，メモリアドレスレジスタ，汎用レジスタのいずれの用途にも使用できる．メモリと MGR 間は 64 ビットのバス幅をもち，Lisp のセルデータが一度に読み書きできる．

SDC はカウンタレジスタである．PSW (Processor Status Word) レジスタのビット設定により，mod 8, mod 16, mod 32 のカウンタとして利用できる．SDC による分岐条件は，31 から 0 になったときに分岐，または SDC の下位 3 ビットが 7 のときに分岐，の 2 つがあり，PSW の設定により選択できる．これらは 3 つのカウンタごとに設定できる．さらに，MGR 内の 32 ビット，16 ビット，8 ビットのデータを間接アクセスするポインタレジスタの機能ももつ．

キャッシュ，アドレス変換，仮想記憶の機能はない．

ELIS の仕様をまとめる

- 8 ビットタグ + 24 ビットポインタのタグアーキテクチャ
- 主記憶容量最大 128M バイト (16M セル)
- 制御記憶 64K 語 (1 語 64 ビット)
- 専用ハードウェアスタック 32K 語 (1 語 32 ビット)
- スタックポインタ 15bit 3 本
- 汎用レジスタ 32bit 32 本
- メモリ多目的レジスタ (MGR) 64bit 4 本
- ソースデスティネーションカウンタ (SDC) 5bit 3 本
- クロックサイクル 60nsec
- マイクロ 1 ステップの実行 180nsec

ブロック図の他のレジスタについては参考文献 [1][2] を参照してほしい．

2.2 ELIS のマイクロ命令

ELIS は CPU 外部におかれた WCS のマイクロ命令で制御される．マイクロ命令は 64 ビット幅の水平型である．命令形式を図 2 に示す．Type I はメモリ制御，Type II は SDC 制御，Type III は 32 ビットの即値発生と補助制御を行なう．ALU 制御は各 Type に共通で，3 アドレスで演算を行なう．メモリ制御，SDC 制御はそれぞれ ALU 演算と並行して実行される．ただし，メモリ読み込みを実行したとき，MGR にデータが確定するまでには 3 マイクロ命令実行分の時間が必要である．確定する前にデータデスティネーションとして指定した MGR にアクセスすると確定するまで自動的に実行に待ちがはいる．それ以外の命令は，メモリ読み込みと並行して実行できる．他の待ちのはいる条件もハードウェアが検出して，必要なら自動的に待ちがはいる．

Type I,II の Sequence Control 欄は，命令実行順序を制御する．14 ビットの次命令アドレス欄と分岐種別指定欄からなっていて，アドレス欄と指定した分岐ソースとの論理和をとって次の実行命令を決める．例えば，

(br y1-0 24)

は，分岐ソースとして直前の ALU の演算結果 ybr のビット 1-0 を指定している (br は branch のニーモニック)．この値とアドレス欄 24 の論理和を取って次のアドレスとする．つまり演算結果による 24, 25, 26, 27 の 4 方向の分岐となる．分岐の種類は豊富である．条件コードによる分岐，タグ部分のビットによる分岐，演算結果による分岐，SDC 状態による分岐などがある．タグビットによる分岐では，TAO のタグで 64 方向に分岐して，データタイプを直ちに判断するといったことが可能である．ただし，条件が成立すれば分岐，しなければ次の命令，という形式はない．さらに，アドレス欄は論理和で修飾されるので，マイクロ命令のアドレス割り付けの制約は厳しくなる．すなわち，多方向分岐命令のターゲットアドレスは分岐の大きさより大きな 2 のべきのアドレスに整列しなければならない．

例えば，直前の演算結果が 0 かどうか，といった分岐もターゲットは偶数番地で，ゼロでないとき，ゼロだったとき，の 2 命令が連続する必要がある．また TypeIII 形式の命令は Sequence Control 欄がないので，必ず次の番地を実行する．このような制約を満たしたアドレス割り付けを行なうため，プログラムソーステキストで連続している命令列が連続したアドレスに割り付けられるとは限らない．ソースの順に実行するため，無条件 goto が多用される．ELIS にはキャッシュがないので，WCS のすべての番地のアクセスは同じ時間で行なえる．よって，連続実行する命令のアドレスが分散していても実行速度には影響ない．

分岐には，通常の分岐と，マイクロアドレスを大きさ 4 のアドレススタックに積んで分岐するサブルーチン分岐がある．アドレススタックは CPU 内にある大きさ 4 であるから，ごく小さなマイクロサブルーチンを呼び出すのに使用する．これ以外は，外付のスタックにアドレスをプッシュするコードを記述し，戻りは ybr をソースとした 65536 方向分岐で行なう．

スタックオーバフロー，外部プロセッサからのメッセージ到着といった条件は，分岐条件の 1 つ (happen, hap) として実現されている．つまり，マイクロプログラムではこの分岐条件含んだ命令を記述して，例外条件や割込み処理を行なう必要がある．ハードウェアで自動的に割込みハンドラルーチンへ分岐するといった機能は備わっていない．

2.3 ELIS とフロントエンドプロセッサ

ELIS の走行は，フロントエンドプロセッサ (FEP) の

制御のもとに行なわれる．ELIS の WCS へのロード，実行開始 / 停止といった走行制御と TAO の入出力のためのディスク制御，ネットワークインタフェース制御，端末制御を FEP が行なう．ELIS と FEP の間は，両方からアクセス可能なレジスタと，ELIS の主記憶領域と FEP メモリとの間の DMA で通信する．試作機では FEP として PDP-11 が使われた．製品版では MC68010 システムが採用された．

ELIS と FEP は独立したプロセッサである．この間の最も下位レベルの通信は，共有レジスタを介したハンドシェークで，ビジーウェイトで通信確認を行なう．

3. マイクロコード変換

この章では，ELIS のマイクロコードを C に変換する手法について述べる．なお，UNIX に移植した TAO/ELIS を C で記述した ELIS の意味で CELIS と呼ぶ．

3.1 変換の方針

TAO はメモリ管理，プロセス管理，インタプリタ，S 式の入出力，多くの関数が手書きのマイクロコードで記述されている．ELIS にはマイクロコードの実行時ロードの機能がないので，TAO の実行中は，マイクロコードの変更はない．よって，WCS のマイクロ命令語を逐次解釈実行していくエミュレーション方式ではなく，マイクロ語を C 言語に変換して WCS 全体を 1 つの UNIX 実行ファイルとしてしまう方式をとることができる．こうすれば，命令語のフェッチ，命令デコードを C への変換の段階ですべて完了することができる．マイクロ語の実行についても，32 ビット整数演算は C の式そのものに変換されるので，CPU の本来のスピードで実行される．

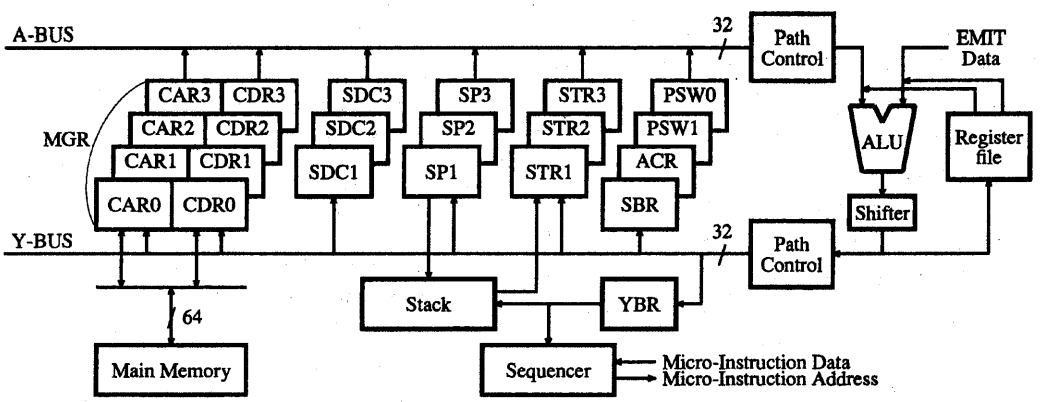

図 1. ELIS のブロック図

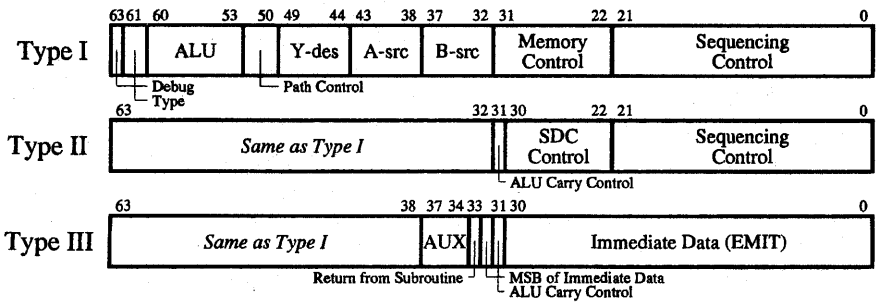

図 2. 命令形式

また，分岐先のアドレスもサブルーチンリターン以外は静的に決まっているので，フロー解析をして冗長な条件コードの計算やレジスタへの代入を省くことも容易である．Cに変換することで，プラットフォームを選ばないで移植できる可能性もある．

C変換には，WCSイメージとともに，マップファイルから作成したマイクロ配列ファイルを使用した． この情報を使って，WCSの語をマイクロソースの順に変換する．マイクロ語のアドレス順ではない．こうすることで，マイクロ語アドレス割り付けで飛び飛びになったアドレスがC上で連続する場合が増え，そのぶんCのgoto文の生成を抑制できる． さらにType I, IIの命令で次命令アドレス欄も含めて同じ命令語が連続しているときには，その語の全体の生成を抑制する (削減量はマイクロ命令約3400語分)．

同じ命令をまとめることを追及すると，ビットパターンでもって命令をハッシュして同じ命令を捜し，それを1回しか変換しないといったことが考えられる．これは，実現してみたが，命令の順序が不連続になること，削減できる命令がそれほどないことから不採用とした．

多方向分岐はGNUプロジェクトのコンパイラgccの間接goto構文を使用して実現した．間接gotoはCのラベルのアドレスを得る演算子 && とそのアドレスへジャンプするgoto *式で記述できる．実際には，間接goto用の配列を用意し，WCS命令の数だけラベルアドレスを初期値として定義している．ラベルはマイクロアドレスを16進数に変換した識別子を使用した．

```
const static void *brwcs[] = {
&&wcs00000, &&wcs00001, &&wcs00002, &&wcs00003,
&&wcs00004, &&wcs00005, &&wcs00006, &&wcs00007,
... };
```

例えば (br y7-0 32768) はこの配列を参照する次のコードに変換する．

```
goto *brwcs[32768|(ybus&0xff)];
```

変数ybusは直前の演算結果を保持する変数である．ELISのybrを意味する．Cのラベルの有効範囲は宣言された関数内であるので，WCS全体をCの1つの関数に変換する．

ハードウェア機能のうち，条件コードの設定，SDCのアップデート，スタックオーバフローのチェック，乗除算など，ソフトウェアでは大きな処理になってしまう機能は別に関数として用意し，その関数の関数呼び出しのコードに変換する．

ELISの主記憶，スタック，レジスタ等はCの配列や整数変数とする．

変換はパソコン用UNIXの1つであるFreeBSD2.2.5-RELEASE上で行なった．最終的にELISのマイクロコードはPentiumの命令列に変換される．

3.2 変換例

変換の例をあげる．マイクロプログラムはS式で記述する．次のはソースではなく，マップファイルである．最初の数が割り付けられたアドレス，次が次命令アドレス欄である．

```
(2087 1216 ((and <sp>+ gmc r3) (jsr tagh5-0 delinv)))
(2088 2028 ((mov <sp>) (br tagcadbl (m0 m1))))
(2028 0 (!m0 (mov gcmark <sp>) srl return))
```

この3語のマイクロ命令は次のCソースに変換される．

```
wcs00827:/* 2087 */
    oybs=ybus;
    ybus=r3=stack[sp1]&0x7fffffff; ...(1)
    sp1++;
    cond32(); ...(2)
    mstack[--msp]=0x828; ...(3)
    if (oybs&TAG5)
      goto *brwcs[0x4c0|((oybs>>24)&0x3f)];
    goto wcs004c0; ...(4)
wcs00828:/* 2088 */
    oybs=ybus;
    ybus=stack[sp1];
    /*cancel CC set*/ ...(5)
    if (oybs&0x20000000) goto wcs007ed;
    /* else goto wcs007ec; */
wcs007ec:/* 2028 */
    oybs=ybus;
    psw0=(psw0&~PSW_F)|PSW_N;
    ybus=stack[sp1]=0x40000000; ...(6)
    psw0&=~PSW_Z; ...(7)
    goto *brwcs[oybs&0xffff]; ...(8)
```

(1) gmcはR26の別名である． TAOでは0x7fffffffを常に保持する定数レジスタとしてだけ使用するので，CELISでは展開する
(2) 条件コード計算の関数の呼び出し
(3) マイクロサブルーチン分岐の戻り番地をプッシュ
(4) tagh5-0はタグの第5ビットが0なら修飾をせずにアドレスを使い，第5ビットが1ならtag5-0とアドレス欄の論理和へ32方向分岐するという変則的な33方向分岐指定である．TAOではタグの第5ビットでアトムかそうでないかを分けているので，このような命令が用意されている
(5) フロー解析で条件コード計算を削除した
(6) gcmarkはR25だが，TAOでは0x80000000の定数レジスタとして使用している． srlは1ビットの右論理シフト指定である．変換時に定数計算をして結

果を出している．条件コードも計算済みである．

(7) シフト結果によって条件コードのゼロフラグをセットする
(8) 外付ハードウェアスタックを使ったサブルーチンの戻り

このようにして，現時点ではマイクロ 49993 命令が 280851 行の C ソースに変換される．この C ソースをコンパイルしてアセンブラソースを出力し，命令数を数えたところ Pentium の 585310 命令に翻訳されていた．単純平均で，ELIS の 1 マイクロ命令が Pentium の約 11.6 命令に翻訳された．実行時ルーチンを呼び出している命令では，実行時ルーチンの命令数を加えている．例えば，1 つの SDC を 1 増加するルーチンは Pentium で 48 命令である．これを加えた結果では，1 マイクロ命令が最長で Pentium の 407 命令に変換された．ただし，実行時ルーチンには条件判断があるので，407 命令が実行されるわけではない．

C ソースをコンパイルするときコンパイルに対する最適化オプション (-O 等) は指定していない．28 万行の関数をコンパイルするのに最適化を指示すると，gcc はフロー解析ステージで swap 領域の 2G バイトを使い尽くして停止してしまう．最適化をしていないので，効率の良いコードがでているとはいえない．また，この 28 万行のソースを SUN ワークステーション (SUN OS 4.1.3) の gcc でコンパイルしてみたところ，gcc が internal error で停止してしまった．この原因はまだ解析していない．

変換用のプログラムは C で約 2500 行である．

4. TAO システムの移植

WCS 変換以外に TAO の動作に必要な，FEP や実行時ルーチン，移植にあたって変更したマイクロや TAO ソースについて述べる．

4.1 FEP の実現

FEP の実現には，(1) FEP の UNIX プロセスを別に用意して通信する，(2) FEP をスレッドで実現する，(3) 1 つのプロセスで実行する，の 3 つの方法が考えられる．

今回は単純な (3) の方式をとった．また，オリジナルコードの変更量を最小限にするため，下位レベルからの FEP と ELIS 間の通信プロトコルはオリジナルを踏襲した．ELIS から FEP の通信は，もっとも下位レベルのハンドシェークで送ったメッセージを配列に保持し，メッセージを送り終わったところでそのメッセージに対応する処理を実行する．この処理の最後の部分で，ELIS への返事のメッセージをキューにいれ，リターンする．ELIS からみれば，メッセージを送り終わると同時に処理は完了していて，ビジーウェイトによる返事の待ちは発生しない．FEP から ELIS の通信もオリジナルどおりメッセージをキューにいれて ELIS の hap ビットをセットすることで行なう．

FEP の処理で，実行がブロックしてしまうようなシステムコール (例えばネットワークからの read) を実行すると，ELIS 全体の実行が止まってしまう．現在のバージョンでは，待ちがはいる可能性があるのは端末からの読み込みだけである．この部分は非同期読み込みと SIGIO シグナルによる入力完了通知の機能を使って，ブロックするシステムコールを使わないようにした．

別に，TAO のマルチユーザ環境を実現するため，CELIS の端末部分のクライアントプログラムも作成した．ソケットと端末からの入力をそれぞれ相手に送る小さなプログラムである．これを使って 1 つの CELIS プロセスを最大 8 人でアクセスして TAO プログラムを実行することができる．

FEP 部分は C で約 5000 行である．すべて新規に書き下ろした．これを先の 28 万行とリンクして 1 つの実行ファイルができる．

ELIS 及び TAO の移植に要した期間は約 2ヵ月である．

4.2 ELIS マイクロコード，TAO ソースの変更点

OS と関わる入出力部分はすべて FEP で吸収されるので，オリジナルコードの変更量は少ない．

マイクロコードの変更点は，主に ELIS の付属ハードウェアに関する次のコードである．行数で 1000 分の 1 以下であった．

- メモリ実装量獲得ルーティン: バスエラーを起こすまで主記憶を順に読み込んでいるルーチン
- バスエラー等の診断レジスタの読み込み: 主記憶の高位番地にマップされていた
- 100 マイクロ秒タイマの読み込み: UNIX プロセスの利用者 CPU 時間の獲得に変更する
- アイドル時間計測: オリジナルはビジーウェイトでアイドルカウンタを増やしている．pause システムコールの呼び出しと，日付時刻獲得ルーチンに変更した
- 実行時ルーチンの呼び出し指定: 各種の実行時ルーチンを呼び出すコードをマイクロソースに埋め込めるように変更した．TypeIII 命令の AUX 欄のうち使用していないコードを使った

TAO 自身で記述された TAO のプロセス制御，ファイルシステムなどの 70000 行のシステムソフトウェアのう

ち移植にあたって変更したのは，端末制御関連で小さなS式を3つ加えただけである．

機能拡張としては，UNIXプロセスとしてのCELISをサスペンドするための関数を加えた．マイクロは5ステップの追加．FEPで実際のサスペンド処理を行なう．

4.3 ハードウェアモデルの変更

移植にあたっては，オリジナルコードの修正を最小限にすることが目標の1つであるが，実行時の性能を確保するのも重要である．そのような例を2つあげておく．

・スタックポインタの範囲

TAOは，ハードウェアスタックの最下位領域と最高位領域を，頻繁にアクセスするデータを保持するための記憶領域として固定的に使用している．よって，スタックポインタが0のときにプッシュして0x7fff番地にデータを書くこと，その逆に高位番地から0番地に循環するプログラムはありえない．そこでスタックポインタの自動増加，自動減少では，結果の値が0から0x7fffにはいっているかどうかのチェックは行なわない．ただし，スタックポインタに値を代入するときには，0x7fffでマスクして代入する．これは上位にタグがついている値をそのまま代入しているコードがあるためである．

・マイクロスタック

マイクロスタックは4段であるが，0のときにプッシュして3に，3のときにポップして0になる構造を期待して書いているコードがかなりの個所あった．エラーの場合など，ポップしないでエラー処理を呼び出してしまうのである．マイクロスタックポインタの操作のたびに0x3でマスクするコードの生成を避けるため，CELISでのマイクロスタックは256段に変更した．そして，unsigned charの変数をマイクロスタックポインタに使うことで循環スタックを実現した．

5. 性能評価

ELISとの速度比較を表1に示す．速度はキャッシュの効きで大きく揺れる．マイクロコードを数語変更するといったことで，taraiは1.55秒から1.75秒まで10%ほど速度が変わる．この数値はあくまでも目安である．

表1. 実行速度 単位 秒

	ELIS	CELIS	比
(tarai 10 5 0)			
interpreted	13.39	4.51	2.9
compiled	4.27	1.55	2.7
accel (1)	23.62	8.27	2.8
asslink (2)	102.82	41.5	2.5

測定項目の(1), (2)は次のとおりである．

(1) 4000行のTAOプログラムをメモリ上でコンパイルする

(2) 2080行のマイクロプログラムのアセンブルとアドレス割り付けを行なう．ディスク入出力を含む．

CELISは，Pentium II 300MHz，キャッシュ512Kバイト，主記憶128Mバイト，FreeBSD2.2.5-RELEASEで実行した．結果として，オリジナルTAO/ELISの2.5倍以上の速度が得られたが，CPUクロックはオリジナルの16.67MHzに対して測定機は300MHzである．実行時のPentium命令実行数をカウントしないと正確な計算はできないが，逆にELISのSDCといった機能が効果を上げていたと考えられる．

6. 今後の予定

まだ実現していないTAO/ELISの機能を実現する．具体的には，TAO/ELISに備わっている，TCP/IPスタックをすべてTAOで記述したネットワークシステムを実現する．また，Pentiumプロセッサの性能評価用カウンタを利用して，実行時の特性を解析し，性能向上を計る予定である．さらに，他のプラットフォームでも動作するよう作業を続ける．

おわりに

水平型マイクロコードを逐次型のCプログラムに変換することで，記号処理ワークステーションTAO/ELISをUNIXに移植した．変換したコードは十分実用的な速度が得られた．またオリジナルのコードの変更点も最小限に抑えられた．

参考文献

[1] 日比野靖，渡辺和文，大里延康: Lisp Machine 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，情報処理学会記号処理研究会，24-3，1983.

[2] 渡辺和文，石川篤，山田康宏，日比野靖: 32ビットAIチップELISのアーキテクチャ情報処理学会計算機アーキテクチャ研究会，69-10，1988

[3] I.Takeuchi, H.Okuno and N.Ohsato: A List Processing Language TAO with Multiple Programming Paradigms, New Generation Computing Vol.4 No.4 Ohmusha and Springer Verlag, 1986

[4] 竹内郁雄: マルチパラダイム言語 TAO, bit Vol. 20, Nos. 1-12, 1988.

[5] 吉田雅治，天海良治，山崎憲一，中村昌志，竹内郁雄，村上健一郎: 記号処理カーネルSILENTのハードウェア構成，情報処理学会計算機アーキテクチャ研究会，114-3，1995.