スタック高速化のための管理アルゴリズムとその解析†

長谷川 誠†† 重井芳治†††

頻繁にアクセスされるスタック先頭部を高速記憶上に割り付けた Top of Stack register 方式スタックに関して，効率的な管理アルゴリズムを提案する．さらに，このアルゴリズムに対する性能解析法を示す．この結果，本アルゴリズムを用い適切な構成方法をとることにより，実効スタック・アクセス時間をほぼレジスタ固有の速度にまで高速化できることがわかった．このときに主記憶との間の転送帯域幅が重要な役割を果たす．

1. まえがき

スタック概念は重要なパラダイムであるにもかかわらず，それを積極的に支援する計算機アーキテクチュアは多いといえない．誕生以来，スタック概念は高級言語計算機の思想と関連して発展をとげてきた[1]．計算の中間結果や環境を保持するための機構として，スタックが大変優れているためである．スタック計算機を仮想ターゲット機械としたコンパイラはすでに一般化しており，スタックは基本的なパラダイムとして完全に定着したといえよう．そしてさらに，ソフトウェアの明解化に役立ち並列処理の自然な表現として期待されている関数型言語の実現に当たってもスタックの果たす役割は大きい[2]．

現在の計算機システムで広く使われている汎用レジスタは，プログラムに対して主記憶の高速小容量な延長部分としての役割を果たしている．しかし，有限容量であって性質の異なる記憶装置が存在するということが言語処理系におけるコード生成の過程を複雑で遅いものにしている．さらに，このことが手続き間にわたるレジスタ割付けの最適化をむずかしくする．これがプログラムのモジュール化に対する障害の一つである．スタックにおいてはこのことは問題とならない．

スタックを全面的に取り入れた計算機はまだけっして多くはないが，LSI 化の進行に伴いスタックを用いた高性能の高級言語計算機を実現する試みがなされている[3]．そして，最新の 32 ビット VLSI プロセッサである iAPX-432[4]・HP-9000[5] はともにスタック概念を基礎としたアーキテクチュアを採用している．このことは，スタック機構というものが，高度のソフトウェア概念の実現と大規模 VLSI チップの系統的設計という要望の双方によく応えるうることを示している．

スタック・アーキテクチュアの弱点はその速度性能にあるといわれている[6]．最も頻繁に参照されるであろうスタック先頭部分を CPU 中の高速記憶上に配置するならば，その実効的な参照速度は大きく改善される．このような方式をここでは TOS (Top Of Stack) レジスタ方式と呼ぶことにする．この方式のハードウェア上への実現法とシミュレーションに基づく評価に関しては飯塚[7]，Blake[8] による研究がある．純スタックのみならずフレームスタックまでをも考慮に入れて検討を加えたものとして筆者の報告[9] がある．

適切な TOS レジスタ管理アルゴリズムを用いるならばスタックの動作速度をさらに高速化できるはずである．そのための，より一般化したアルゴリズムを提示するのが本論文の第一の目的である（2 章）．従来の TOS 管理法は本アルゴリズムの特殊な場合として記述できる．さらに，性能解析のために本管理法をランダム・ウォークとしてモデル化する．3 章では，実効スタック・アクセス時間に対する解析的な解を得るのをおもなねらいとして，ランダム・ウォーク・モデルの解析法を示す．解析的な解を得ることにより系の挙動を明確にすることができる．現時点において妥当と考えられる諸元を用いた数値例によると，適切な構成の下ではほぼ TOS レジスタ固有のアクセス時間にまでも高速化できる．最後にスタック動作速度の支配的要因の一つであるデータ転送帯域幅についてその改善策を示す（4 章）．本論文では演算スタックあるいはリターンスタックのように，プッシュ／ポップ量が一時にはたかだか ±1 である純スタック方式について扱っている．

† On a Stack Management Algorithm and Its Analysis by MAKOTO HASEGAWA (Computer Science Department, College of Engineering, Shizuoka University) and YOSHIHARU SHIGEI (Department of Information Engineering, Faculty of Engineering, Tohoku University).

†† 静岡大学工業短期大学部情報工学科
††† 東北大学工学部情報工学科

2. TOS 管理アルゴリズム

2.1 k-復帰 TOS 管理アルゴリズム

TOS レジスタは CPU 内の s 語のレジスタから構成され，スタックの上部を収容している．スタックの残りの部分は主記憶中に形成されている（**図1**）．スタックの先頭の現在アクセス中の語を指すポインタを TOS ポインタと呼ぶ．スタックは上へ向かって伸びていくものとする．TOS レジスタを使い切ってしまったときあるいは空になったとき，TOS レジスタの底部を通して主記憶中のメモリ・スタックとの間でデータ語をやりとりする．このとき TOS レジスタ全体をシフトレジスタとして押し下げたり押し上げたりすることができるものとする．TOS レジスタが空になった後さらにデータ語を取り出そうとしてポップすることをオーバポップ，使い切った後さらにプッシュしようとすることをオーバプッシュと呼ぶ．また，オーバポップおよびオーバプッシュすることをひとまとめにしてオーバフローという言葉を使う．

効率的な TOS レジスタ管理法として k-復帰アルゴリズムを提案する．このアルゴリズムは，オーバプッシュあるいはオーバポップされようとして TOS レジスタだけではそれ以降の操作を続けられなくなったときに起動される．説明の都合上，まずここでは TOS レジスタ全体がシフトレジスタのような形で実現されていることを前提としておく．同一アルゴリズムが，CPU 内にとった固定高速記憶とポインタによってスタック先頭部を実現した場合にもそのまま適用できる．

まず TOS レジスタがオーバプッシュされようとした場合について示すと（**図2**），

1) TOS レジスタ全体を k 記憶単位分押し下げて TOS レジスタの上部に大きさ k 語の空間を作る．このとき，TOS レジスタの底からあふれ出したデータ語を次々と主記憶へ転送する．

2) TOS ポインタをオーバプッシュの位置から k だけ戻して（$s+2-k$ に位置付け）その位置にオーバプッシュを引き起こしたデータ語を書き込む．

3) この位置を新たな初期状態として演算動作を再開する．

したがって TOS レジスタから主記憶へ k 語が書き込まれる．同様にして，オーバポップ時には TOS ポインタを k だけ戻して（k に位置付けて），この k 語の区間を主記憶から読み出したデータ語で満たす．ここでは，演算評価部とメモリ転送部は並行動作することがないものとしている．アルゴリズムの形式的な記述を**図3**に示す．このアルゴリズムは現実のスタック計算機 HP-9000, HP-3000[8] で使用されている TOS 管理法をそのなかに含んでいる．

2.2 ランダム・ウォーク・モデル

さて本論文においてはこれ以降は境界型の位置表現をとるものとする．すなわち，スタックの先頭の現在参照中の語と次にアクセスされる語との境目を遷移位置として扱うことにする．通常スタックポインタはな

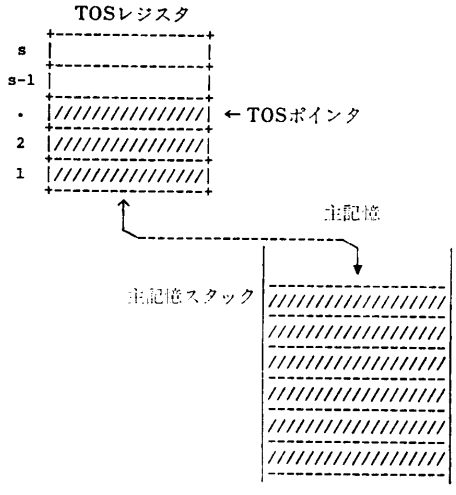

図 1 TOS レジスタ方式スタックの構成
Fig. 1 Configuration of TOS register stack.

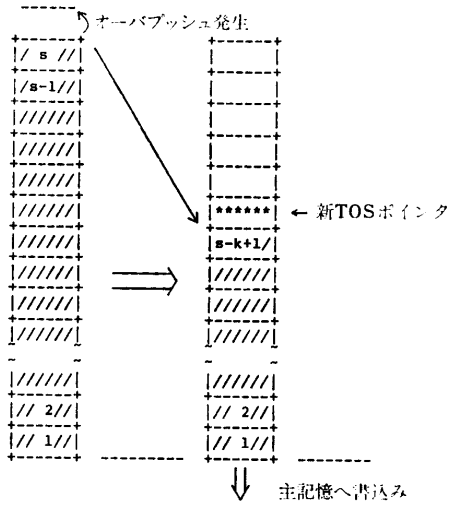

図 2 オーバプッシュ時の処理例
Fig. 2 Example of cut-back-k operation on over-push.

んらかの実体を指すものとして扱われている（たとえば，現在参照中の語，あるいはそれよりも一つ大きい番地等）．これに対して，語の境界とランダム・ウォークの遷移位置とを対応させることにより，スタックへのプッシュ／ポップはまったく対称的な動作であるとして統一的な取扱いが可能となる．この方法をとったとき，通常の意味での TOS レジスタ中の位置 $j-1$ に対して遷移位置 j が対応することになる（図4）．したがって，TOS 記憶の大きさ s に対して，取りうる遷移位置は $[0, a]$ である（ここで $a=s+2$）．オーバポップが位置0への遷移，オーバプッシュが位置 a への遷移として表現される．

一時には一語のプッシュ／ポップしかされない純スタック方式を対象として，これを離散時間系におけるランダム・ウォークとしてモデル化する．単位区間は固定長命令サイクル時間にとり，各命令動作におけるスタック操作部分のみについて注目する．いま，ある命令サイクルにおいて，スタックがプッシュ・ダウンされる確率を p，ポップ・アップされる確率を q，変化しない（無動作）確率を r とする（$p+q+r=0$, $0 \leqq p, q, r < 1$）．実際のスタック操作が行われたときにのみ TOS レジスタ・アクセス時間 T_s が付加されるものとする．TOS レジスタの大きさは s 記憶単位分用意されているものとする．スタックの特性から次のことがいえる．すなわち，いったんプッシュされた語は必ずポップされてプログラム終了時には開始時と同じ位置がスタックの先頭となっている．したがって $p=q$，すなわち単位サイクル時間中にプッシュされる確率とポップされる確率は等しい．

3. 純スタック方式の解析

ランダム・ウォークのモデルをもとにして k-復帰アルゴリズムにおける実効スタック・アクセス時間を求める．

3.1 あふれ確率

位置 k から出発してランダム・ウォークを続けた後に TOS レジスタの下側にあふれる確率（TOS レジスタが空になった後さらにオーバポップされる確率）を Q_k，ランダム・ウォークを続けた後上側にあふれる確率（TOS レジスタが全部満たされた後さらにオーバプッシュされる確率）を P_k とする．最初の単位命令サイクル時間経過後には，スタックの先頭位置は $k-1$, k, $k+1$ のいずれかであり，したがって $1<k<a-1$ の範囲に対して，

```
var Tos_Register: array [1..S] of integer;
    sp: integer;     {Top-of-stack pointer}

procedure push(Tos_Data:integer);
begin
  if sp<S then begin                {normal operation}
       sp:=sp+1;
       Tos_Register[sp]:=Tos_Data end
  else begin                        {for overpush}
       sp:=(S+1)-K+1;
       SAVE_STACK;
       Tos_Register[sp]:=Tos_Data end;
end;

procedure pop(var Tos_Data:integer);
begin
  if sp>0 then begin                {normal operation}
       Tos_Data:=Tos_Register[sp];
       sp:=sp-1 end
  else begin                        {for overpop}
       sp:=K;
       RESTORE_STACK;
       Tos_Data:=Tos_Register[sp] end;
end;
```

図 3　k-復帰 TOS 管理アルゴリズム
Fig. 3　Cut-Back-k TOS management algorithm.

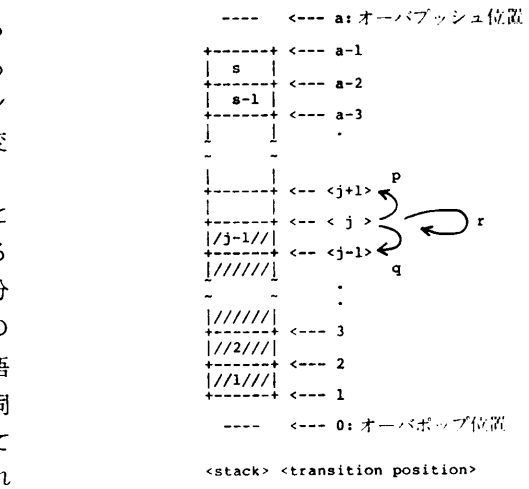

図 4　TOS レジスタの遷移位置表現
Fig. 4　Stack transition diagram: relation of TOS-pointer and random walk transition position.

$$Q_k = pQ_{k+1} + qQ_{k-1} + rQ_k \tag{3.1}$$

を得る．整理して次式となる．

$$(1-r)Q_k = pQ_{k+1} + qQ_{k-1} \tag{3.2}$$

$k=1$ に対しては最初の単位サイクル時間にオーバポップされるかもしれないので，このとき $Q_1 = pQ_2 + q + rQ_1$．同様に $k=a-1$ に対しては最初の単位時間にオーバプッシュされることがありうる．したがって $Q_{a-1} = qQ_{a-2} + rQ_{a-1}$ である．このことから，方程式を統一するために次のように定義する．

$$Q_0 = 1, \quad Q_a = 0. \tag{3.3}$$

こうすると，オーバポップ確率 Q_k は $k=1, 2, 3, \cdots, a-1$ に対して(3.2)を満足する．方程式(3.2)は差分

方程式であり，(3.3)はその Q_k に関する境界条件を表しているわけである．ここでは特殊解の方法によって Q_k の解の形を求める．

スタックの性質から $p=q$ がいえた．このとき差分方程式(3.2)は二つの特殊解をもつ[10]．したがって任意の定数 c_1, c_2 に対して，数列

$$Q_k=c_1+c_2\cdot k \tag{3.4}$$

は(3.2)の一つの形式的な解を表している．境界条件(3.4)を満足させるためには，$c_1=1, c_1+c_2\cdot a=0$ と置かねばならない．よって，

$$Q_k=1-k/a \tag{3.5}$$

となる．

同様の議論によって，今度はオーバプッシュする確率を求めると，

$$P_k=k/a \tag{3.6}$$

である．この双方の和を求めると，

$$P_k+Q_k=1 \tag{3.7}$$

となり，このことは時間の経過とともにいつかは必ずTOS レジスタのあふれることを（確率1で）意味している．

3.2 継続時間

スタック操作が TOS レジスタ内だけで行える区間数を継続時間と呼ぶことにしよう．ここではその期待値 D_k を求める．

もし最初の単位命令サイクル時間でスタックに1語プッシュされたならば，その後はあたかも最初の状態が $k+1$ であったかのようにしてこの系は推移する．したがって，かりに最初のスタック操作がプッシュであったとすると，その条件付継続時間の期待値は $(D_{k+1}+1)$ である．操作がポップであれば $(D_{k-1}+1)$，無操作の場合は (D_k+1) となる．このことから，継続時間の平均値 D_k は次の差分方程式

$$\begin{aligned} D_k&=p(D_{k+1}+1)+q(D_{k-1}+1)+r(D_k+1)\\ &=pD_{k+1}+qD_{k-1}+rD_k+1 \end{aligned} \tag{3.8}$$

さらに整理して，

$$(1-r)D_k=pD_{k+1}+qD_{k-1}+1 \tag{3.9}$$

で記述され，その境界条件は，

$$D_0=0,\quad D_a=0 \tag{3.10}$$

であることがわかる．

定数項1を含んでいるので差分方程式(3.9)は非同次型になっている．ここでスタックの性質から $p=q$ がわかっているので，いまの場合 $D_k=-k^2$ が(3.9)の一つの解となっている．そこで，(3.9)のすべての解は $D_k=-k^2+c_1+c_2k$ の形式をとる．境界条件(3.10)によって，$c_1+c_2=0, -a^2+c_1+c_2a=0$ でなければならない．これを c_1, c_2 について解けば求める解 D_k が得られて，

$$\begin{aligned} D_k&=k(a-k)/(1-r)\\ &=k(s+2-k)/(1-r) \end{aligned} \tag{3.11}$$

である．

3.3 実効アクセス時間

TOS レジスタと主記憶との間で k 記憶単位の転送を行うのに必要な時間を $t(k)$ とする．1回の主記憶参照では1語しか読み出せない場合には，これは主記憶アクセス時間 T_a に等しい．高度にインタリーブされている場合あるいは MTB 方式[11),12)] の場合には，

$$t(k)=T_a+(k-1)T_c \tag{3.12}$$

となる（**図5**）．ここで T_c は主記憶参照サイクル時間である．用語の定義を明確にしておくと，ここでいうアクセス時間とは主記憶の参照を開始してから最初の語が利用可能となるまでの時間，サイクル時間とはその後相続いて参照されている語が利用可能になるまでの時間間隔であるものとして用いる．すなわち，インタリーブ方式の場合その多重度を m とすると $T_c=T_a/m$ である．

平均的には，TOS 上での操作を D_k 区間続けた後に TOS レジスタはオーバフローする．したがって，オーバフロー率 P_{ovf} は，

$$\begin{aligned} P_{ovf}&=1/D_k\\ &=(1-r)/(s+2-k)k \end{aligned} \tag{3.13}$$

で表される．オーバフローが生じるたびに TOS レジスタと主記憶の間で k 記憶単位のデータを転送する．また，D_k 区間の間には平均 $D_k(1-r)$ 回のスタック操作を行う．したがって，1回のスタック参照に要する時間の平均値，すなわち実効スタック・アクセス時間 T_{eff} は，

$$\begin{aligned} T_{eff}&=\frac{D_k(1-r)T_a+t(k)}{D_k(1-r)}\\ &=\frac{k(s+2-k)+T_a+(k-1)T_c}{k(s+2-k)} \end{aligned} \tag{3.14}$$

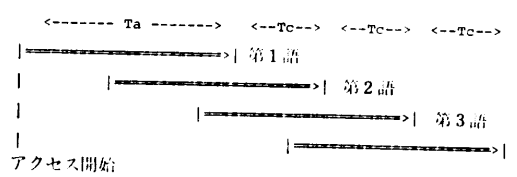

図5 主記憶アクセス時間とサイクル時間の関係
Fig. 5 Memory accessing scheme.

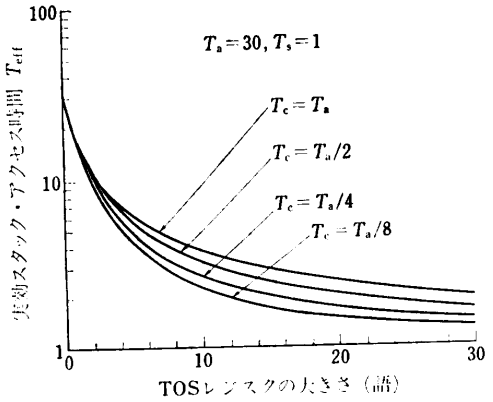

図 6 TOS レジスタの大きさの実効スタック・アクセス時間に与える効果
Fig. 6 Effective stack access time versus TOS register size.

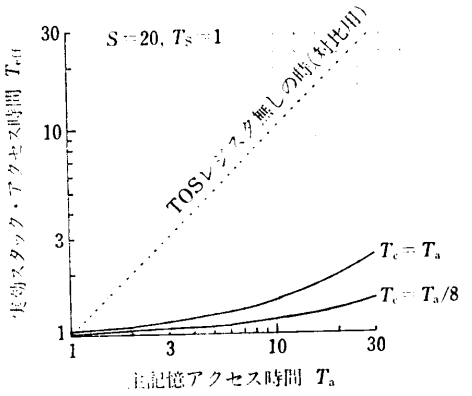

図 7 主記憶アクセス時間の実効スタックアクセス時間への影響
Fig. 7 Effective stack access time versus main memory access time.

となる，ここで T_s はスタック・レジスタの固有アクセス時間である．実効アクセス時間を最小値とする k を求める．式(3.14)を k について微分したものを0とおいて k について解くと次式となる．

$$k=1-m+\sqrt{(1-m-s-2)(1-m)} \qquad (3.15)$$

ただし，$m=T_a/T_c$ である．

3.4 数 値 例

前節までに得られた結果を元にして数値例を以下に示す．実効スタック・アクセス時間は TOS の大きさの増加とともに急激に減少するが，それ以降の変化は少ない（**図6**）．この飽和領域においてさらに実効アクセス時間を改善するには，主記憶転送バンド幅を拡大することが有効である．TOS レジスタの大きさ (s) を30とし主記憶アクセス時間 $T_a=30T_s$, $T_c=T_a/8$ にとった場合，実効アクセス時間は TOS レジスタの固有アクセス速度の1.2倍程度となる．**図7**は主記憶の動作速度の変化が実効スタック・アクセス時間に与える影響を示している．実効スタック・アクセス時間の増加率は主記憶アクセス時間の増加率の 1/15～1/20 である．

実効スタック・アクセス時間と最適復帰位置の関係を**図8**に示す．一度の主記憶アクセスでは一語しか転送できない場合（$T_c=T_a$）には，実効スタック・アクセス時間を最小にする復帰位置は $k=1$ である．これは HP-3000 等で使用しているパラメータと一致している[8]．メモリ転送バンド幅の増大（T_a に対する T_c の比の増大）にしたがって実効アクセス時間の最小値を与える復帰位置は TOS レジスタの中央部へ移っていき，$T_c=T_a/16$ のときには $k=8$ となる．このとき，復帰位置が多少変化しても実効アクセス時間が大きく変化することはない．オーバフロー率を最小にする復帰位置は式(3.13)により $s/2+1$ である．したがってオーバフロー率を最小化する条件と実効スタック・アクセス時間を最小化する条件とは必ずしも一致していない．

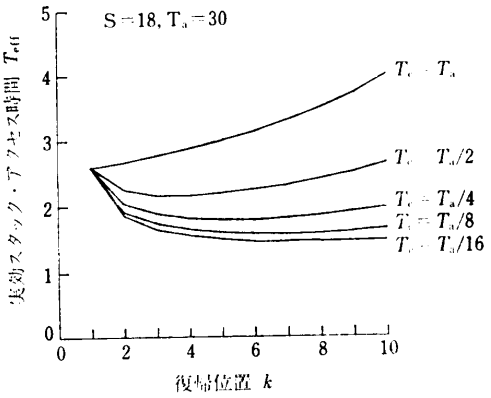

図 8 実効アクセス時間最小化のための最適復帰位置
Fig. 8 Optimum cut-back position for effective stack access time.

4. メモリ・アクセス帯域幅

TOS レジスタと主記憶間の転送帯域幅がスタック実効アクセス時間を大きく左右することがわかった．メモリ帯域幅を拡大するためにはいくつかの方法がある．アクセス経路の幅を物理的に拡張したり高多重度のインタリービングを行ったりすることが代表的なものであろう．しかし，いずれもかなりコストを要するものである．これに対して，効率的な転送方式として Multi-Track Bus (MTB) と呼ばれる方式が存在す

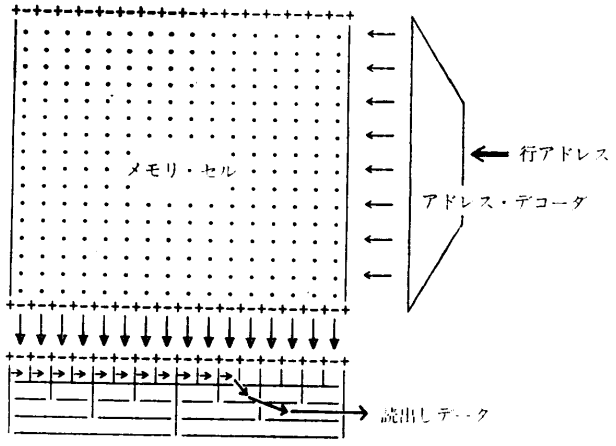

図 9 チップレベル MTB による高転送帯域幅の実現
Fig. 9 Chip level Multi-Track-Bus scheme for high data transfer bandwidth.

る[11),12)]．この方式によれば，容易に高転送帯域幅を得ることができる．しかも，実現に要するコストは小さい．この方式をメモリ・チップのレベルに適用したMTB メモリ・チップの構成例を図 9 に示す．選択された行アドレス中の全セルについて並列読出しを行い，それらが出力バッファにセットされたならば，階層構造をもちそれ自身がシフトレジスタとしてデータ転送をする Multi-Track Bus を通してブロック単位で高速に直列転送を行うわけである．MTB を構成するシフトレジスタの個々のセルの動作時間が帯域幅を定める．したがって，この方式によれば容易に高メモリ帯域幅を得ることができる．VLSI パッケージのピン数制限も大きな制約とはならない．従来の方法はいずれもメモリ・チップの入出力端子を出た後のみを問題としている．しかしながら，この時点ですでに Sutherland のいうアクセス幅の損失が存在しているわけである[13)]．

5. むすび

本管理アルゴリズムを用いて適切なスタック構成法を選択することにより，実効スタック・アクセス速度がほぼ TOS レジスタ固有の速度 (1.1～1.5 T_r) にまで高速化でき，主記憶の影響をごく小さなものとすることができることがわかった．

そのためには，まず TOS レジスタと主記憶間のデータ転送帯域幅を十分に確保しておくこと，これに合わせて復帰位置 k を選択することが必要である．そのための諸元は 3 章の結果から得ることができる．飽和域における実効スタック・アクセス時間の改善はデータ転送帯域幅の拡大に依存している．計算機の構成要素が VLSI 化していくことを考えると，高転送帯域幅の実現のためにはその特性を生かした新しい方法を考えることができる．チップ・レベル MTB はその一例である．

VLSI チップ上のシフトレジスタとして TOS レジスタを実現するならば，その固有アクセス時間は汎用レジスタよりも高速となしうる可能性がある．そのための構成法，さらに環境フレームの形成・削除をも考慮した取扱いが残された課題であるがこれについては改めて述べたい．

謝辞 スタックという洗練された概念を生み出した Barton にまず感謝したい．佐藤利三郎教授（現東北学院大）をはじめ東北大学情報工学教室の 1977～78 年の生体研究グループ諸氏との討論は有意義なものであった．著者の一人が現在所属する静岡大学の諸兄姉に感謝する．互いに忌たんのない議論をたたかわすことができたのは著者にとって大きな励ましであった．

参考文献

1) Bulman, D. M.: Stack Computers: An Introduction, *Computer*, Vol. 10, No. 5, pp. 18-28 (1977).
2) Berkling, K. J.: Reduction Language for Reduction Machines, Proc. of the 2nd Symp. on Computer Architecture, pp. 133-138 (Jan. 1975).
3) 和田，仲辻，金田，前川：高級言語向きスタックマシン上での Pascal マシンの実現と評価，信学論，D-44, pp. 369-376 (1983).
4) David, L. B. et al.: The Execution Unit for the VLSI 432 General Data Processor, IEEE J. Solid-State Circuits, SC-16, 5, pp. 514-521 (Oct. 1981).
5) Beyers, J. W.: A 32-bit VLSI CPU Chip, IEEE. J. Solid-State Circuits, SC-16, 5, pp. 537-542 (Oct. 1981).
6) Strecker, W. D.: VAX 11/780—A Virtual Address Extension to DEC PDP-11 Family, AFIP Proc. 1978 NCC, pp. 967-980 (1978).
7) 飯塚 肇：適応構造計算機に関する研究，電子技術総合研究所研究報告，767 (1976).
8) Blake, R. B.: Exploring a Stack Architecture,

Computer, Vol. 10, No. 5, pp. 30-39 (1977).
9) 長谷川誠，重井芳治：スタック管理アルゴリズムの性能評価法について，信学技報，EC 83-28 (1983).
10) 高橋武人：差分方程式，培風館，東京 (1961).
11) Hasegawa, M. et al.: Distributed Communicating Media—a Multi-Track Bus—Exploiting Concurrent Data Exchanging, Proc. of the 8th Symp. on Computer Architecture, pp. 367-372 (1981).
12) 長谷川誠：An Interconnection Scheme for Computing Systems, 東北大学学位論文 (1982).
13) Sutherland, I. E. and Mead, C. A.: Microelectronics and Computer Science, *Scientific American*, Vol. 237, No. 9, pp. 210-228 (1977).

(昭和59年6月29日受付)
(昭和59年9月20日採録)