解　説

スタック・マシンとソフトウェア*

井　上　謙　蔵**

1. まえがき

ALGOL 60 のプログラムの機械語への翻訳や，目的プログラムの実行にスタックが有効な役割を果すことが示されてから，KDF 9 や Burroughs の計算機など，スタック機構を持つものがあらわれ，しばらくの間は特異な計算機としての立場を誇示していたが，最近スタック機構を持つ多数のミニコンピュータが出現した．これはソフトウェアに対してスタックの及ぼす大きな影響が認識されたためであるが，IBM のアーキテクチャの強大な市場的拘束をうける大，中形機と異なって，独創性を発揮し易いミニコンに対して，初めてなしうることであろう[1]．

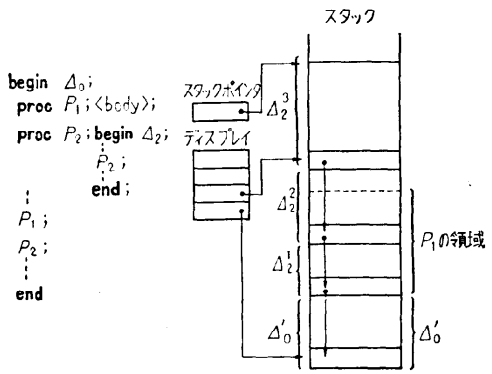
実際に，現在の高級プログラム用言語はますますスタック処理を必要とするようになっているし，オペレーティング・システムも，スタックを取入れることで，その構造を簡素化できる．

本解説では，スタックが活躍するソフトウェア上の場所と範囲，実際のスタック・マシンの特徴を示し，スタックのソフトウェア構成への影響を論じよう．

以下では特に断らない限り，高級プログラム用言語，オペレーティング・システム，ALGOL 60 等の代りに，言語，OS，アルゴル等と言うことにする．

2. プログラム用言語とスタック

アルゴルがあらわれてから，変数名やラベルの局所的宣言を可能にするブロック構造が言語の普遍的な基本特徴と考えられるようになった．その理由は，局所的にだけ使用される変数の名前は，全体的共通的な変数と無関係に自由に命名でき，かつ使用する手元で宣言できることである．また実行上は，並列的な2ブロックの局所的変数はそれぞれ生きている時点が重複しないから，同じ記憶領域を割付けることができ，従って記憶領域の節約を図ることができる（図-1）．



注）$\Delta_2^1, \Delta_2^2, \Delta_2^3$ はそれぞれ P_2 の 1,2,3 回目の呼び出しに対する変数領域で Δ_0' は Δ_0 に対応する．

図-1　手続き P_2 の3回の回帰呼び出中のスタック領域

最近大形のソフトウェア作りとの関連で，プログラミングの構造化が叫ばれているが，その方法は，プログラムの手順を細分化していく際に，各部分プログラムの内部構造が，互いに干渉の少ないように分解することである．ここに言う部分プログラムをブロックに対応させれば，ブロック間の干渉は，それが実行される順序と共通変数であるから，不必要に共通変数を使用しないで，できるだけ局所的変数で用を足すように心掛ければよい．ブロックの役割の再認識である．

言語がブロックの構造を持つ場合のデータの割付けには，スタックが最も効果的であることは良く知られている．アルゴルのように手続きの回帰的呼出しや配列の大きさの動的変更が生ずる場合でも，データの有効範囲がブロック構造によって決定されるときは，スタックを用いて効率のよいデータの動的割付け管理を行い得る．むしろこのような場合に，スタックはその有効性を発揮すると言ってよい．

グラフィクスや人工知能の分野の発達で，リスト的

---

* Stack Machine and Computer Software by Kenzo INOUE (Department of Information Science, Tokyo Institute of Technology)

** 東京工業大学理学部情報科学科

なデータ処理を行う言語が多数作られたが，これらの言語ではその性格上手続きの回帰的呼出しが一層重要な機能となるので，スタックの重要度は高くなる．実際にハードウェア・スタックを付することによってLISP プログラムの処理時間を 30% 以上短縮した場合の例が報告されている[2]．

但しこの場合には，スタックでは処理しきれないデータ構造，すなわちリスト（ポインタで要素を結合したデータ構造を，ここではリストと呼ぶことにする）が現れる．PL/I や ALGOL 68 等でもポインタ（後者ではリファレンス）を用いてリスト処理を行うことができる．その名前の有効範囲がブロックに限られるときは，ある程度スタックの制御を受けるが，個々のデータの割付け領域の発生消滅は，必らずしもスタックに都合のよい，last-in, first-out の規則に従わない．

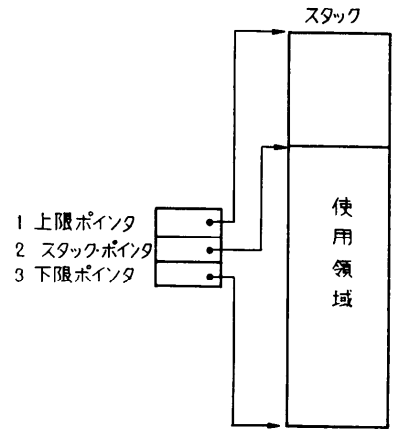
次に言語処理系を考えよう．IBM 704 の上に作られた最初の FORTRAN コンパイラでも，すでに式の処理にスタックを使用していた[3]．アルゴルは言語を句構造としたため，その構文解析がスタックを用いてコンパクトに実行された．文脈自由文法を用いた言語の定義は，構文を単純にかつ厳密に定義すると共に構文と意味との関係を単純化したので，その後の言語の定義法はこの影響を強く受けるようになった．コンパイラの中で構文解析を受持つプログラム部分は，大きさにして全体の 10% にも満たないであろう．しかしその実行時間は，全走行時間の 30% から 40% に及ぶのが普通である．従ってスタック処理を効率的に行うことは，コンパイラの走行時間の観点から重要となる．

処理系の場合には，それで処理するプログラム中に宣言された名前と，その属性を保存するための表が活躍する．その要素の構造はいわゆる構造体であるが，言語がブロック構造を持つときは，表への要素の記入，削除，引用はすべてスタックと同じ制御を受ける．

目的プログラムの生成のさいに，意味情報のスタックが必要となる．これは原プログラムの文脈依存の情報を伝播するためのもので，誤りからの回復処理を効果的に行うためにも必要となる．

ここ数年来，目的プログラムを最適化するための技術が発達した[4]．その方法はループの内側の命令列を，演算時間の観点から，効率のよいもので置換する．このためにはプログラムの制御の流れをあらわす有向グラフを作らねばならない．これはリストである．

以上で，言語処理系及びそれで処理されるプログラムでは，スタックとリストが重要な役割を担っていることを示した．スタック処理で最も頻繁に使用されるのは棚あげと棚おろしである．これらの処理にはスタックの頭を指すポインタの内容の変更と，その変更がスタックに与えた記憶領域の限界を突破しないかどうかの検査が必要である（図-2）．棚あげ，棚おろしの動作は，ハードウェア上の簡単な手当で，それを必要とする命令と並列的に実行できる種類のものである．



注）棚上げのさい，ポインタ 1,2，棚下しのさい，ポインタ 2,3 が比較される．

図-2 スタックの上下限

3. OS とスタック

OS で最も頻繁に使用されるデータ構造はリストである．外部からのプログラムの投入，資源の割付待ち，時間の配分待ち，他のスタック待ち等によってジョブやタスクの待ち行列が形成されるが，これらの待行列のデータ構造は単純リストである．

これらの待行列の処理は，ジョブ管理，タスク管理等のプログラムの仕事であるが，それは入出力機器，コンソール・タイプ，端末，時計のあふれ，等々に従う割込みや，タスクからの呼出しによって起動される．このような割込み処理は，スタックを用いれば，通常の手続き呼出しと全く同様に処置されるので，OS の構造が概念的に単純化される．すなわち割込みが生ずれば，レジスタ類や復帰に必要な情報を割込まれたプログラムのスタック上に積上げて，割込み処理プログラムに入ればよい．割込処理プログラムは，心要ならば同じスタックを継続して使用することになる．

会話型システムのプログラムは再入可能形に作られ

る．この場合もスタックを用いる方式は，他の方式より方法が単純となる．例えばあるタスクが中断され，それと同じ手続きを使用する他のタスクへ制御が移る際，再開のための情報は前者のスタックに積まれるだけでよい．この方法は同じ手続きを使用するタスク間のみならず，一般の並列処理に成立することである．

OS を構成するプログラムは，アセンブラで作成するのが伝統的な手法であるが，OS の巨大化に伴なう作成上，保守上の問題から，ここ数年来言語を用いようとする実験的作業が行われてきた．実際には IBM の PL/S などに見られるように，当面は中級言語が作業用言語として採用される趨勢にあるが，いずれにしても OS の規模やその作成工数を考えれば，その言語にはブロック構造を与え，プログラムの構成を階層化，ブロック化しやすいようにしてやらねばならないし，実際にもそうなっている．

このような言語の目的プログラムの実行にはスタックが用いられるから，さきの割込処理や並列処理の場合は，OS の他の部分と全く同じ処理手法に従い，全く同じプログラム構造を持つことになる．

4. スタック・マシン

はじめに注意したように，最近のミニコンピュータの中にはスタック機能を持つものが多い．例えば，MODCOMP IV, ECLIPSE, PDP 11, HP 3000 等である．

スタック・マシンの元祖は，KDF 9 と Burroughs の B 5500 である．これらはアルゴル向きに作られた計算機で，ミニコンピュータに比べ遙かに高度のスタック機構を持つし，命令系は逆ポーランド記法の処理のためのものである．

ここでは単純と複雑の代表の，PDP 11/20 と B 6700 のスタック機構について一瞥しよう．

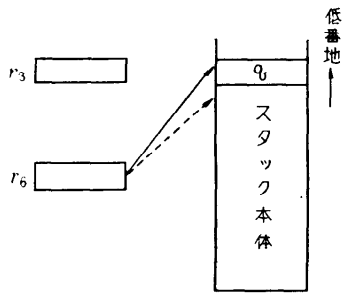
4.1 PDP 11/20[6]

この計算機の命令は大体 2 アドレス方式で，アドレス部にはレジスタ名を直接演算レジスタとして，またはある番地を指すポインタとして与えることができる．ポインタとして使用した場合には，更に次の指定ができる．

（1） 使用の後，ポインタの内容を 1 増加，

（2） ポインタの内容を 1 減じてから使用．

これらの機能は，低い番地のほうをスタックの頭とすると，それぞれ棚おろし，棚あげに対応する．従って第二オペランドに指定されたポインタでスタックの本体を制御すれば，第一オペランドのレジスタをスタックの最先頭要素として，1 命令で演算と共に棚あげ，棚おろしが同時に処理される（**図-3**）．もちろん，スタックの溢れも検出される．



命令 $A, r_3, (r_6)+$ によって q がレジスタ r_3 に加算され r_6 の内容 1 は増加．

図-3 PDP 11/20 のスタック機構

第三の機能としてサブルーチン呼出しのさい，戻り番地がスタックに棚あげされる．これは手続きの回帰的呼出しの手段を与える．また割込発生にさいしては，ステータス・ワードが棚あげされるから，割込処理も通常のサブルーチン呼出しと同じ方法でなされる．

この計算機の場合は，これだけである．スタックもその下限がハードウェア上 256 番地に定まっている．しかし ECLISPSE 等では最内側の手続きに対応するスタック領域（フレーム）のベースを別のポインタに確保し，パラメータや変数の引用，領域のブロックでの確保，解放の便宜を図っている[7]（**図-4**）．

4.2 B 6700[8]

この場合は，ジョブごとに独自のスタックを設けることができ，スタックの頭は 2 個の演算レジスタ A，

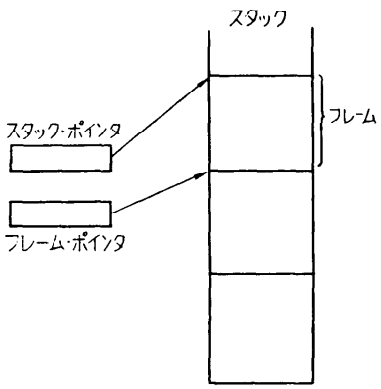


図-4 フレーム・ポインタを持つスタック

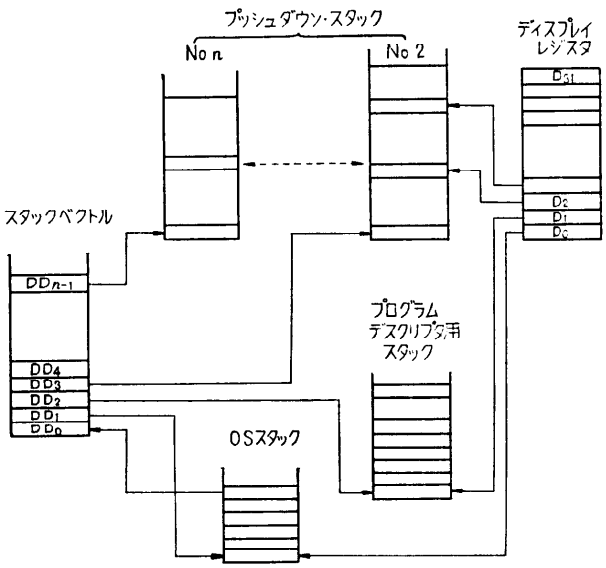


図-5　B 6700 のスタック構成[8]

B となっている（長精度に対してそれぞれ X, Y の拡張レジスタを持つ）．プログラムの不変部分（手続きのコードと定数）のデスクリプタ（マッピング）はひとつのスタックに入り，変数部分は別のプッシュダウン・スタックを構成する．後者はこの再入可能プログラムの同時的使用ジョブの数だけ設定できる．前者をスタックというのは一寸当らないが，プログラム上の取扱いは後者と同じである（**図-5**）．

ベクトル DD の第 0, 1, 2, … 要素が，それぞれ OS のスタック（OS の各プログラムや定数のデスクリプタ，その他），上記プログラム不変部分スタック，ジョブ 1 のプッシュダウン・スタック，…等のベースを指す．

32 個のディスプレイ・レジスタ $D_0, D_1, \cdots$ が，変数部分スタックの現在生きている 32 レベルまでのブロック領域のベースを指す．これは 1 レベルだけの場合のフレーム・ポインタの拡張で，入れ子をなす変数領域の最も効率的な表現方式である[9]．

ディスプレイを活用して，ブロック領域のベースが手続きの呼出しに応じて変更されるときの，更新復旧等のための履歴の保存が簡便に行われる．

この計算機の最大の特徴は，上記複雑なスタック機構に加えて，逆ポーランド記法処理用の命令系である．このため，例えば加算命令は番地部を持たないから，命令の長さは 8 ビットでよい．この命令によって，A, B レジスタの加算が行われ，結果は B へおかれて，A レジスタは空と印づけられる．しかしあらかじめ両レジスタに被演算子の値を置かねばならないので，通常の計算機に比べやや冗長の感があるが，レジスタ中にデータをもたらす値とりの命令は 16 ビット長にすぎないからプログラムは長くなるわけではない．それればかりか値とりの相手が単純変数でなく，関数であれば，その呼出しがスタックの機能を活用して，自動的に行われるので，この部分に関しては，通常の計算機より遙かに簡単なプログラムとなる．

表-1 ALGOL 目的プログラム (OP) の比較

	アトラス	U 1108	B 5500
命　令　数	1.0	0.41	0.63
ビットでの長さ	1.0	0.31	0.16
実 行 時 間	1.0	0.28	1.8
実行命令数	1.0	0.83	0.96

表-1 にアトラス，U 1108，及び B 5500 に対するアルゴル・プログラムの目的プログラムの大きさや速度の結果を引用する．これはコンパイラの作り方の問題もあるから，傾向をあらわす数字と考えたほうが無難だが，B 5500 の場合に著しくプログラムが小形となることに注目したい[10]．

B 5500 は B 6700 の祖形であるが，逆ポーランド記法処理用命令系で 0 アドレス命令が活用される結果，プログラム長は著しく短くなっている．この点は，その後小形計算機である B 1700 で意識的に採用された[11]．すなわち，中間言語の目的プログラムをマイクロ・プログラムで実行するようになっており，中間言語は原言語によく調和するように，数種類設けられている．その性格はアルゴルや FORTRAN に対しては，B 6700 レベルの機械語程度と考えてよいが，特に最も頻繁に使用される命令が，最も短かくなるように選ばれた．その結果 FORTRAN の目的プログラムでは，IBM 360 に対し 50%，COBOL の場合は同じく 30% の大きさに止まっている．かつ原言語ごとに中間言語を採用した結果，効果的な複合命令が作り易く，そのため比較的処理速度が速い．

5. スタック・マシンのソフトウェアへの影響

すでに必要なことはつくされたので，要点をまとめてみよう．

言語がブロック構造の上に作られるのは，実用上，

論理上常識となっているが，そのような言語の翻訳処理，目的プログラムの実行の両面にわたって，スタックが重要な役割を果たす．これらのプログラムはOSの環境の中で動くのであるから，OSの諸プログラムを最外側のブロックで定義されたものと考えれば，思想的には完全に調和のとれた体系が形造られる．

スタックを用いる命令体系としては逆ポーランド記法処理用のものが適切であり，それがビット数で数えたプログラムの大きさを小型化し，記憶装置の大きな節約が図られる．もっともこの点では，B 6700の命令系はデータを2段階で引用する機会が多いので，時間の上では余り経済的とはいえないであろう．またスタック用命令がアルゴル一辺倒になっているのも応用の点から気にかかる所である．

スタックのほかに重要なデータ構造にリストがある．それゆえハードウェアはスタックのみならず，リストに対しても，その操作を高速に行えるような手段を与えるべきである．

スタックとリストに対して効率のよい引用方法を与える番地づけと命令系が得られるならば，今日の複雑すぎるOSの構造を単純で，統一的な思想の上に再構成することができよう．

参考文献

1) 松崎，田中：日経エレクトロニクス No. 110, 48 (6, 16, 1975).
2) 山下，その他：計算機アーキテクチャ研究会資料 75-10 (1975).
3) Sheridan, P. B.: CACM 2, 2, 9 (Feb. 1959).
4) Schaefer, M.: A Mathematical Theory of Global Program Optimization, Prentice-Hall (1973).
5) 寺島：情報処理 16, No. 5, 492 (June 1975); SIGPLAN Notices 6, No. 9 (Oct. 1971).
6) PDP 11/20, 15, r 20 processor handbook, digital equipment corporation (1971).
7) 松崎：日経エレクトロニクス, No. 104, 33 (3, 24, 1975).
8) Hauck, E. A. and Dent, B. A.: AFIPS Conf. Proc. 32, 245 (1968); Burroughs B 6700 Information Processing Systems Reference Manual (1972).
9) Gries, D.: Compiler Construction for Digital Computers, John Wisley, chap. 8 (1971).
10) Wichman, B. A.: CJ 15, 1, 8 (Feb. 1972)
11) Wilner, W. T.: AFIPS Conf. Proc. 41, 579 (1972).

(昭和50年6月23日受付)
(昭和50年7月15日再受付)