≪論　文≫

ミニコン付加のプッシュダウン・スタック機能*

大館祥三** 山下堅治** 菱沼千明**

Abstract

This paper presents a new architecture of push-down stack which is implemented by putting some hardware to a minicomputer HITAC 10 interface. The major advantage of this stack is that it is not a last-in-first-out buffer, Which is useful for recursive programming, backtracking, coroutines, and multiprocessing. In order to make it possible, the top four cells are constructed with accessible registers. The lower portion of the stack is layed in the successive location of main memory. Both overflow and underflow of data from the specified portion of main memory are detected automatically as interrupts, so that it is possible to make the software routine independently which can save or unsave some blocks of the stack in the memory to/from the secondary memory. The method for saving and unsaving is also described.

1. はじめに

記号処理，たとえば高級言語の構文解析やリスト処理，あるいはある種の数値解析などは手続きの定義の中にその定義自身の参照を含んでいる．このような手続きは自分自身をサブルーチンとして呼び出すいわゆる再帰的手続き (recursive procedure) としてよく知られている[1]．再帰的手続きの実行には，サブルーチンの各回の呼び出しに対してそれぞれ別個にパラメータと戻り先番地のための記憶場所が用意されなければならない．プッシュダウン・スタックは最後に入れたものを最初に取り出す (Last-In-First-Out) という原則にもとづいて働く記憶のメカニズムであり，このような記憶場所として適している．

本来，プッシュダウン・スタックの思想はB5000[2][3]などの大型機に採用され成功したものであるが，ミニコンのようにレジスタ数と命令の種類が限られ，アドレッシングも窮屈である計算機では特に有効であると思われる．このような傾向はたとえば，PDP-11[4]，Micro-data 3200[5] などにみられる．

本論文では，ミニコン HITAC-10 にプッシュダウン・スタックの機能を付加したシステムについて述べる．これは，インターフェイスにスタック・プロセッサと呼ばれる装置を付加し，スタックの先頭4語をハードウェアとしたほかは，DMA (Direct Memory Access) によって主記憶の一部をメモリー・スタックとしたものである．

以下，2章ではスタック・プロセッサの概要を，3章および4章でその機能，動作について詳述する．さらに，5章では記憶容量の少ないミニコンにおいては特に有効であるスタックの待避回復が本システムで容易に行なえることを示し，6章はシステム全体の構成および競合対策について述べる．

2. スタック・プロセッサの概要

プッシュダウン・スタックを実現するのに，LIFO のバッファではスタックの先頭部分だけにしかアクセスできないので，実際に使用するのには不十分であると考えられている[6]．その理由は，プログラムによっては，複雑な階層構造を有し，以前にスタック内に格納されているデータの直接的な参照を必要とするため，LIFO では能率が悪くなるからである．たとえば，LISP のような再帰的プログラムでは，スタックにすでに格納されているパラメータや戻り先番地を直接参照したり，変更したりすることにより実行の能率を高めることができる．これは再帰的呼び出しがひんぱんな処理系ではきわめて有効である．このように，スタックの先頭に近いいくつかのデータを自由にアク

* Push-down Stack Architecture to a Minicomputer Interface, by Shouzou OHDATE, Kenji YAMASHITA and Chiaki HISHINUMA(Department of Electrical Engineering, Faculty of Engineering, Keio University)

** 慶応義塾大学工学部電気工学科

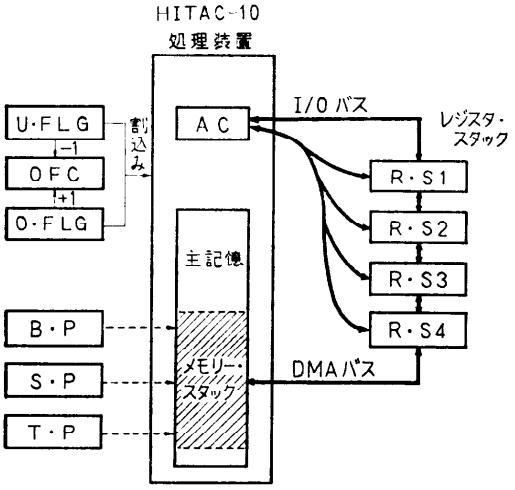

Fig. 1 push-down Stack Configuration with stack prosessor

セスできれば，柔軟性のあるプログラムを組めるばかりでなく，無駄なスタックの消費をおさえることもできる．

ここで述べるスタック・プロセッサでは，このような動作を可能にするため，スタックの先頭4語はアキュムレータとの間で直接データの交換が可能なレジスタ（レジスタ・スタック）で構成されている．そして，この続きのスタックには主記憶のある連続した一部分（メモリー・スタック）を利用している．

データの転送は，プログラムI/Oバスを用いてアキュームレータとレジスタ・スタックの間で行なわれ，レジスタ・スタックからオーバーフローしたデータは，DMAによってメモリー・スタック上に書き込まれる．メモリー・スタックの領域を定めるために2つのアドレス・レジスタ（ボトム・ポインタ，トップ・ポインタ）が備わっており，この領域を超えて動作しようとしたときには，スタックのオーバーフローまたはアンダーフローとして自動的に処理装置に割込みが生ずる（**Fig. 1**）．

3. 処理装置とのインターフェイス

スタック・プロセッサの構成と処理装置とのインターフェイスを**Fig. 2**に示す．これらは次のレジスタ群で構成されている．

（1） レジスタ・スタック （R·S1, R·S2, R·S3 R·S4）

スタックの先頭4語に相当する16ビットのレジスタの組であり，シフト・レジスタで構成されている．

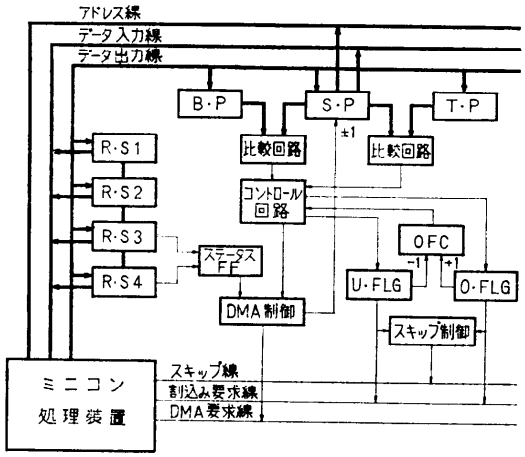

Fig. 2 Hardware confignration of stack processor

（2） スタック・ポインタ （S·P）

メモリー・スタックの先頭番地を指す15ビットのカウント・レジスタであり，処理装置のアドレス線につながる．

（3） ボトム・ポインタ （B·P）

トップ・ポインタ （T·P）

それぞれ，メモリー・スタックとなる主記憶の下限，上限番地を保持するレジスタ．

（4） オーバーフロー・フラグ （O·FLG）

アンダーフロー・フラグ （U·FLG）

それぞれ，スタックのオーバーフロー，アンダーフローを示すフラグで処理装置の割込み要求線につながる．

（5） オーバーフロー・カウンタ （OFC）

スタックのオーバーフローによって+1，アンダーフローによって−1される4ビットのカウンタ．

（6） ステータス・フリップフロップ

R·S3，R·S4が空か否かを表示するフリップフロップであり，DMA要求を制御する．

スタック・プロセッサには，これらのレジスタ群のほかにいくつかの制御部がある．そのうち，コントロール回路は，スタックのオーバーフローやアンダーフローを検出し，O·FLG. U·FLGをオンにすると同時にOFCを1増加させたり，減少させたりする．また，DMA制御部は処理装置へのDMA要求を制御すると同時にS·Pを1増加させたり，減少させたり

する．

つぎに，スタックの基本動作であるプッシュダウン，ポップアップの実行によって各部がどのような動作をするかを詳しく説明する．

まず，プッシュダウンでは，レジスタ・スタック内のデータは深い方にシフトされ，アキュムレータの内容が R·S1 に転送される．このとき，R·S3 にデータが格納されていれば S·P の内容が1増加すると同時に DMA 要求が出され，R·S4 の内容が S·P の指す番地に書き込まれる．ただし，R·S3 が空のときは DMA 転送は行なわれず，レジスタ・スタックだけが働く．したがって，スタックが先頭3語の範囲で使われている間は S·P の増加と主記憶へのアクセスは行なわれない．

S·P の内容と T·P の内容が等しい状態でプッシュダウンの命令が実行されると，オーバーフローとなり上述の動作は行なわれず，OFC を1増加させるとともに，O·FLG をオンにして，処理装置に割込みを要求する．**Fig. 3** に，これらのプッシュダウンの動作をフローチャートにして示す．

つぎに，ポップアップの動作を説明する．R·S4 が空でない，すなわちメモリー・スタック内にもデータが格納されている時には，R·S1 の内容がアキュムレータに転送され，レジスタ・スタック内のデータは浅い方にシフトされる．そして同時に DMA 要求が出され，S·P の内容が1減少し，S·P の指す番地の内容が R·S4 に DMA 転送される．ただし，R·S4 が空の場合には DMA 転送は行なわれず，レジスタ・スタックだけが働く．これは，プッシュ動作と同じようにスタックが先頭4語，すなわちレジスタ・スタックの範囲で使われているときは，S·P の減少と主記憶へのアクセスは行なわれない．

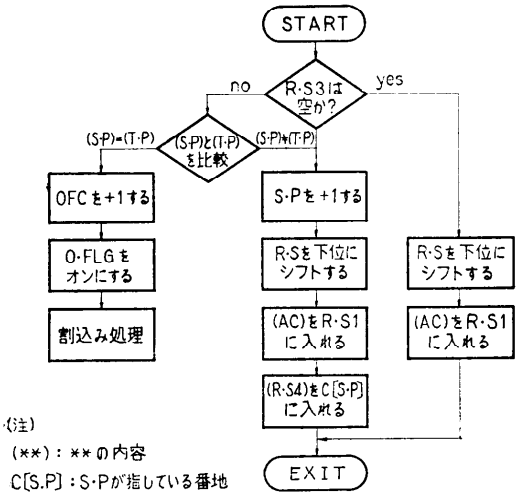

Fig. 3 Flow chart of push Instruction

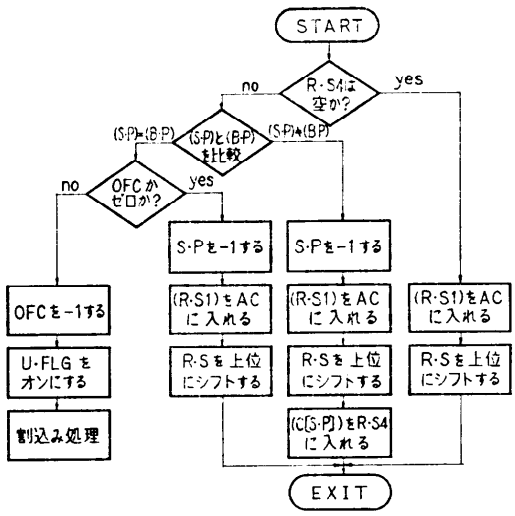

Fig. 4 Flow chart of pop Instruction

S·P と B·P が等しい状態でポップアップの命令が実行されたとき，もし OFC が1以上ならばアンダーフローとなり，処理装置に割込みを要求する．また，もし OFC がゼロならば，アンダーフローとはならず，レジスタ・スタックに残っているデータがポップアップされる．これらのポップ動作をフローチャートにしたものが **Fig. 4** である．

割込み処理については5章で説明する．

4. スタック制御命令

スタック・プロセッサは通常の入出力装置と同じように IOC 命令を用いて制御される．これに必要な IOC 命令とその機能はつぎのとおりである．

(1) Set Stack Pointer and Clear (SSPC)

AC の内容を S·P にセットするとともに，割込みフラグ (O·FLG, U·FLG) をクリアーする．

(2) Set Stack Bottom Pointer (SSBP)
Set Stack Top Pointer (SSTP)

AC の内容をそれぞれ B·P, T·P にセットする．また，SSBP 命令によってスタック・プロセッサのイニシャライズが行なわれる．

(3) Push Down (PUSH)

Pop Up (POP)

これらの命令によって前章で述べたプッシュダウン，ポップアップの動作を行なう．

(4) Put (PUT 1, PUT 2, PUT 3, PUT 4)

AC の内容をそれぞれ R·S1，R·S2，R·S3 R·S4 に転送する．ほかのレジスタの内容は変わらない．PUT 4 命令が行なわれたときは，S·P の指す番地の内容も書き換えられる．

(5) Get (GET 1, GET 2, GET 3, GET 4)

それぞれ，R·S1，R·S2，R·S3，R·S4 の内容が AC に読み込まれる．

(6) Skip Stack Over Flow (KSOF)

Skip Stack Under Flow (KSUF)

スタックがオーバーフローあるいはアンダーフローならばつぎにおかれた命令をスキップして進行し，そうでなければこの命令は無効命令となる．

(7) Get Stack Pointer (GSP)

S·P の内容が AC に読み込まれる．

以上がスタック・プロセッサのための IOC 命令であるが，これらのうちデータ入力命令は前もってアキュムレータをクリアーしておかなければならない．

5. スタックの待避回復

ミニコンでは一般に記憶容量が小さいため，大きなメモリー・スタックを持たせることは好ましくない．そこで，スタックを出入りするトラヒックの大部分は上位の数位置だけにしか関与しないため，何らかの方法で，現在必要としないスタックの深い部分を一時的に外部記憶* に待避させ，必要になったら回復させればメモリーを有効に使うことができる．すなわち，スタックをいくつかのブロックに分け，最新のブロックだけ主記憶に残してアクセスする．こうすることによって，概念的には無限に大きなスタックを作ることができ，しかもハードウェアと待避回復の方法を工夫すると，このためのオーバーヘッドをほとんど無くすことができる．

本システムでは，スタック・ポイントとトップ・ポインタ，ボトム・ポインタとの比較によって，スタックのオーバーフロー，アンダーフローを金物が自動的に検出し，処理装置に割込みを要求する．これにより，スタックの待避回復は割込み処理によって行なうことが可能となる．この様子をつぎに説明する．

* 本システムでは，H 10 (B) の主記憶を用いる『Fig. 7 (次頁照参)』．

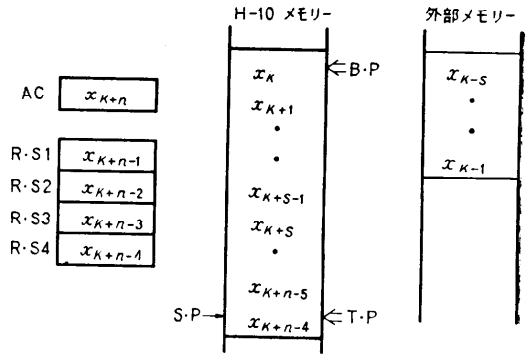

Fig. 5 Stack Condition before overflow of the stack

Fig. 5 はオーバーフローが起こる直前の状態を示したものである．ここで，S·P の内容は T·P の内容に等しくなっている．この位置でプッシュ・ダウンが行なわれるとオーバーフローとなり，スタック・プロセッサは処理装置に割込みを要求する．処理装置は割込みフラグを調べてオーバーフローであることを知ると，スタック待避プログラムへ制御を渡す．待避プログラムはメモリー・スタックの内容を外部記憶に移し，スタック・ポインタを新しくセットし直した後，新ためて PUSH 命令を行なう．このとき，待避されるのはメモリー・スタックの深い部分だけである．このことは重要で，もしすべてを待避させ，つぎに POP 命令が実行されると，アンダーフローとなり待避した部分を再び元に戻さなければならなくなる．さらに，この状態を境にしてプッシュダウン，ポップアップの動作が繰返されると，スタックの待避回復のためのオーバーヘッドが非常に大きくなってしまう．**Fig. 6** はオーバーフローの結果スタックが待避された後の様子を示している．

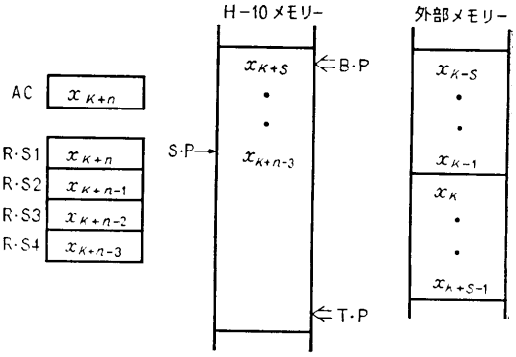

Fig. 6 Stack condition after save of the stack

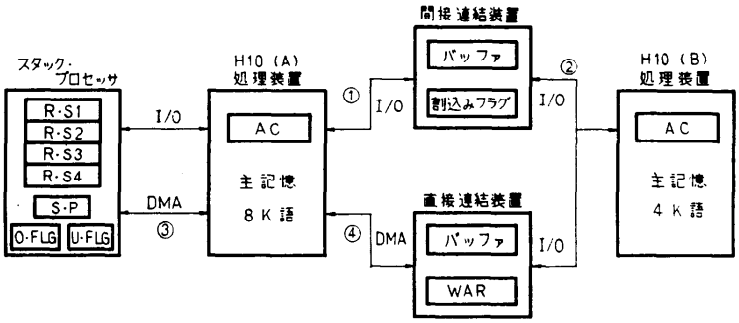

Fig. 7 Organization of Dual-System

スタックのアンダーフローと，それによるスタックの回復はオーバーフローと待避の逆である．このときも，待避の場合と同じように何語か余裕を持たせて回復させる．なお，オーバーフロー・カウンタがゼロのときにはアンダーフローは起こらず，レジスタ・スタックのみがスタックとして働らく．このときは，外部記憶に待避されているブロックは無いはずであり，レジスタ・スタック内にはまだ有効なデータが残っているからである．

6. システムの構成と競合対策

本システムの構成は2台の HITAC 10 (以下 H 10 (A) と H 10 (B) と略記) を並列処理制御機構によって結合したデュアルシステムとなっている (**Fig. 7**)．これまで述べたスタック・プロセッサは H 10 (A) に対して働らくものである．

並列処理制御機構は互いの入出力インターフェイスを介して連結するためのものであり，つぎの2つの装置よりなる．

(1) 間接連結装置

バッファとフラグを備えた簡単な装置であって，両方の処理装置は互いに入出力命令によって，AC とバッファ間のデータ転送，およびフラグのセット，検出を行なう．

(2) 直接連結装置

H 10 (A) の主記憶と H 10 (B) の AC 間で直接データ転送を行なうための装置であり，バッファと主記憶の番地を指示するための WAR(Word Address Register) より成る．WAR は H 10 (A) のアドレス線 (AIN) につながり，WAR が指す番地とバッファ間でデータの転送が行なわれる．これと同時に，H 10 (B) の AC とバッファの間でデータ転送が行なわれるので，H 10 (B) はプログラム制御のもとで H 10 (A) の主記憶に直接アクセスすることができる．

本システムでは，2台の処理装置がそれぞれ独立に動き，しかもインターフェイスは非同期式であるから，競合に対する対策が必要である．HITAC 10 では，プログラム IO によるデータ転送 (低速転送) と DMA による転送 (高速転送) は時分割制御のもとで互いに同時動作が可能であるから，これらの間では競合は起こらない．問題となるのは，Fig. 7 に示すインターフェイスのうち，

(1) ①と②で両方の処理装置が同時にデータ転送命令を実行したとき，

(2) ③と④の DMA 機能が同時に働くとき

である．(1)では，一方の処理装置からデータ転送命令を受けると，命令スキップ機能を利用して次の命令をスキップさせるとともに，相手の処理装置に対して割込み要求を出す．この場合バッファは共有であるから，両方の処理装置からのアクセスを同時に受けると，競合状態に陥る．これを避けるために，以前にどちらか一方からのデータ転送要求を受けてフラグがオンのときは命令のスキップが行なわれず，また両方から全く同時 (±460 n sec 以内) にデータ転送要求があったときは，両方の処理装置において命令のスキップが行なわれない．

(2) の場合は DMA バスを利用して高速データ転送する装置間の競合対策に帰着する．HITAC 10 には拡張機構としてマルチプレクサ・インターフェイス機構が用意され，同期信号によって時分割に同時動作させることができるが，本システムではこれを用いずに，非同期のまま競合を回避し，同時動作を可能にしている．この方式の概略を **Fig. 8**(次頁参照)に示す．

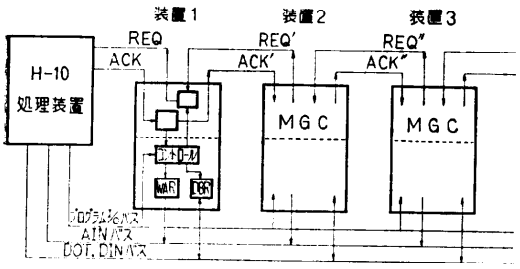

Fig. 8 multiplexer Gate controller

DMA 機能を利用して直接主記憶との間でデータ転送する入出力装置は，その一部に MGC (Multiplexer Gate Controller) を持ち，いもずる式につながる．MGC は優先決定回路 (Priority Controller) とゲート・コントローラから構成されている．優先決定回路は，各入出力装置に優先度をつけるためのものであり，自分の装置と直後につながる装置からくるデータサービス要求 (REQ) を調べ，後の装置が動作中でないときは DMA 要求を出すことができるが，前の装置が動作中であるか，または同時に要求が出されたときには，この要求に対する返事 (ACK) は前の装置の動作が終了した後に返ってくる．ゲート・コントローラは優先決定回路により選ばれた装置の信号ゲートを開く働きをする．すなわち，処理装置に近い装置ほど優先権があることになる．

このように，競合の対策は十分にとられており，わずかな付加装置のもとで完全に回避されている．

7. 特徴

本システムの特徴はつぎのように要約される．

(1) ミニコンの内部に手を加えることなしに，入出力インターフェイスにわずかなハードウェアを付加するだけで，プッシュダウン・スタックの機能が得られている．

(2) スタックの先頭4語は別々にアクセス可能なレジスタであり，これによって柔軟性のあるプログラムが組めるだけでなく，無駄なスタック・アップがおさえられ，メモリーを有効に使うことができる．

(3) スタックのオーバーフロー，アンダーフローはハードウエアが自動的に検出するため，スタックの一部分の待避回復を割込み処理によって行なえる．

(4) 2台の処理装置は互いにバッファを介してデータ転送を行なうため，同期などの制限がなく，相手を一つの入出力装置であるかのように扱える．

(5) 競合対策はじゅうぶんにとられており，わずかな付加装置によって完全に回避されている．

(6) 本システムのために付加されたハードウェアはわずかであり，ミニコンの価格に比べてじゅうぶん低い．

8. むすび

以上，プッシュダウン・スタックの機能，およびスタック機能を備えたミニコンのデュアルシステムの構成と特徴について述べた．本システムは，MINILISP の並列処理を目ざしており，レジスタ・スタックの個数は LISP システムにもとづいているが，これについてはさらに検討中である．

なお，本文で述べたスタックの機能は既存のミニコンの入出力インターフェイスに付加したものであるから，スタックの先頭の2つの要素に関する算術演算や，サブルーチン・ジャンプ，ネスティング・ストア等の機能は無いが，ほかにいくらかの特徴を備えており，再帰的プログラムの実行にはきわめて有効であると考えられる．

謝辞 本システムの作製に際してご検討くだされた本学相磯秀夫教授，中西正和講師に深謝いたします．

参考文献

1) D. W. Barron: Recursive Techniques in Programming, Macdonald Computer Monographs (1968)
2) R. S. Barton: A new approach to the functional design of a digital computer, Proc. of WJCC, May, p. 393-396 (1961)
3) C. B. Carlson: The mechanization of a push-down stack, Proc. of FJCC, p. 243-250 (1963)
4) G. Bell, et al.: A new arcrchitecture for minicomputers the DEC PDP-11, Proc. of SJCC, Vol. 36, p. 657 (1970)
5) R. Burns & D. Savitt: Microprograming, stack architecture ease minicomputer programer's burden, Electronics, Vol. 46, No. 4, p. 95-101 (1973)
6) D. G. Bobrow & B. Wegbreit: A Model and Stack Implementation of Multipie Environments, Comm. of ACM, Vol. 16, No. 10, p. 591-603 (1973)

(昭和49年6月24日受付)