データストリーム処理方式のデータベースコンピュータ

田中 譲
(北海道大学 工学部)

概 要

DBマシンの開発目標が，単にメインプロセッサの負荷の軽減を図るだけでなく，従来のデータベース処理においては基本演算とは見做されていなかったジョインやディヴィジョンのような高度な集合演算を高速に実行するマシンの開発へと移りつつある。本稿では各々 $\log n$ 個程度の比較器を持ち，パイプライン処理によりサーチとソートを高速に処理するモジュールを提案し，これを用いたストリーム処理方式のデータフローデータベースコンピュータの概容を述べる。

1. 序論

ファイルプロセッサを拡張し，ファイルのアクセスのみならず，データベース処理の一部を肩代りして，メインプロセッサの負荷を軽減することを目指したバックエンドプロセッサの研究は，70年代になって始められた。XDMS[1]等がこの範疇に入り，言わば第1世代のデータベースマシンと言うべきものである。その後，バックエンドプロセッサの研究は，70年代初めに新しく提案された関係データベースモデルの簡明さを取り込み，RAP[2]，CASSM[3~4]，RARES[5]等のデータベースマシンの提案を生んだ。これらのマシンは関係モデルの表形式をアーキテクチュアに反映し，多数の回転メモリーデバイスと，数トラック毎に1個のプロセッサを持つ多重マイクロプロセッサの構成になっている。この型のDBマシンを多重トラックプロセッサ型と呼ぶことにする。これは言わば第2世代のDBマシンである。その後，CCDや磁気バブルデバイスの進歩により，回転メモリーとして，トラック毎にヘッドのある特殊なディスク装置のかわりに，これらの電子ディスクを用いた多重トラックプロセッサの提案がいくつかなされた[6~7]。これらは第2.5世代のDBマシンと言える。

一方，関係モデルの簡明さと普遍性は，知的データベース管理システムや知識データベースの研究に代表されるような高次の知的処理に関する基礎研究を著しく発展させた。これは，関係モデルが集合論に基づいており，従来のデータベースモデルでは基本的演算とは考えられなかった新しい型の集合演算を基本的演算として定義していることに負うところが大きい。特にジョインやディヴィジョンのように2つの関係に跨がる交差的演算は，データベースの処理に関する理論の発展に本質的な役割を果している。しかし，交差的演算を，ポインターやインデックスを駆使して，ソフトウェアで妥当な時間内に処理することは不可能である。

このような事情から，DBマシンの開発目的が，単にメインプロセッサの負荷を軽減することだけでなく，ノイマン型と呼ばれる現在のメインプロセッサでは妥当な時間内に処理することのできない集合演算を高速に処理するシステムの開発へと変遷しつつある。以前に，著者等は多重トラックプロセッサのエミュレータの開発[8~9]と解析的モデルの検討[9]により，この型のDBマシンは第2の開発目的に添うものではないとの結論を得た。

交差的集合演算等の集合演算の高速化には，ビットマップ，ハッシング関数，

サーチ／ソート等の技法が用いられ得る。既にこうした考えに基づく基本的な構成デバイスの提案がいくつか生まれている。この動向を、肩代りを開発目的としていた第1，第2世代のDBマシンから、集合演算の高速化を目指す第3世代のDBマシンの開発への推移の兆しと見ることができる。前述の種々の技法の内，ビットマップ処理では，動的にビットマップを作成することによるオーバーヘッドと，交差的演算に要するビットマップが巨大になることに問題点がある。ハッシングは，交差的演算の前処理に有効であるが，交差的演算そのものを高速化するものではない。[10~11] 本稿で述べるシステムはサーチ／ソート技法を用いている。

サーチ／ソートの高速並列処理アルゴリズムは既にいくつか知られているが[12~13]，これらをDBマシンの基本モジュールとして採用するには次のような問題点がある。第1に，コストと開発可能性の観点から，メモリーを除く論理回路の複雑さが，処理の対象となるデータ数に比例するようなアーキテクチュアは採用すべきでない。第2にデータベースの扱う膨大な量のデータが処理部に常駐すると仮定すると膨大な数の処理デバイスを必要とするので，処理部と記憶部は別れていて，両者の間，および処理部を構成するモジュール間でデータ転送がなされると仮定すべきである。したがって，データの処理部常駐を仮定したアーキテクチュアや，転送に用いられるチャンネル容量を過大に仮定したアーキテクチュアは現実的でない。一方，従来より，DBマシンの隘路はデータの転送に起因することが常識として認識されている。上述の第2点は，この認識と一見矛盾している。本稿で述べるシステムは，転送時間と処理時間を完全に重畳することにより，この矛盾を解決している。すなわち，著者等は，対象データはデータのストリームとして各種モジュール間を転送されるものと考え，データストリームの転送とサーチ／ソート等のデータストリームの処理とをパイプライン的に重畳させ，2つの過程を同時に進行させることのできるアーキテクチュアの開発を目指している。本稿では，著者等の開発目標となっているデータストリーム処理方式のデータベースコンピュータの基本素子であるサーチエンジンとソートエンジンの2つのモジュールのアーキテクチュアを明らかにし，データストリーム型データベースコンピュータ（以下DSDBCと略す）の概容を示す。[14]

2. サーチエンジン（SEE：Search Engine）

2.1. サーチエンジンの機能

DSDBCでのキー探索処理はSEEで行う。SEEは値の順に整列したn個の被探索キーからなる順序集合中で，別に与えられるm個の任意に並んだ探索キーを順々に探索し，値の等しい被探索キーが見つかった時はその被探索キーの順番を出力し，見つからなかった時はその旨を出力するモジュールである（図2.1）。

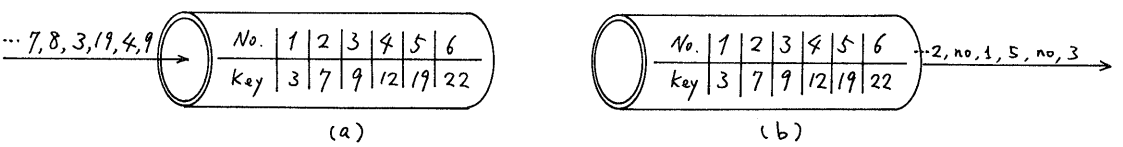


図2.1. SEEによるキー探索の一括処理.

SEEは，この処理過程における3つの過程，すなわち，探索キィーの他のモジュールからの転送，探索処理，探索結果の他のモジュールへの転送をパイプライン処理により重畳させ，探索キィーがデータストリームとしてSEEというパイプを通過する間に探索結果のデータストリームに変換することができる。

2.2. 左充填2分木とパイプライン探索

SEEの用いるデータ構造は図2.2に示すように，通常の2分木を左に詰めた形になっており，以下ではこれを左充填2分木と呼ぶ。小文字は整列した被探索キィーが2分木のノードに格納される順序を示している。

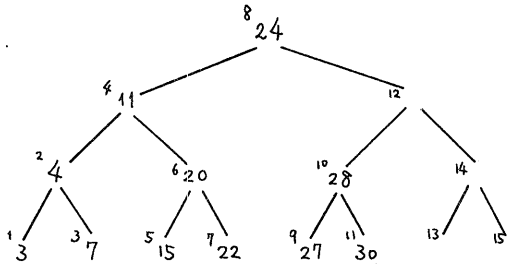


図2.2 左充填2分木の例

SEEによるパイプライン探索の概念図を図2.2の例を用いて図2.3に示す。

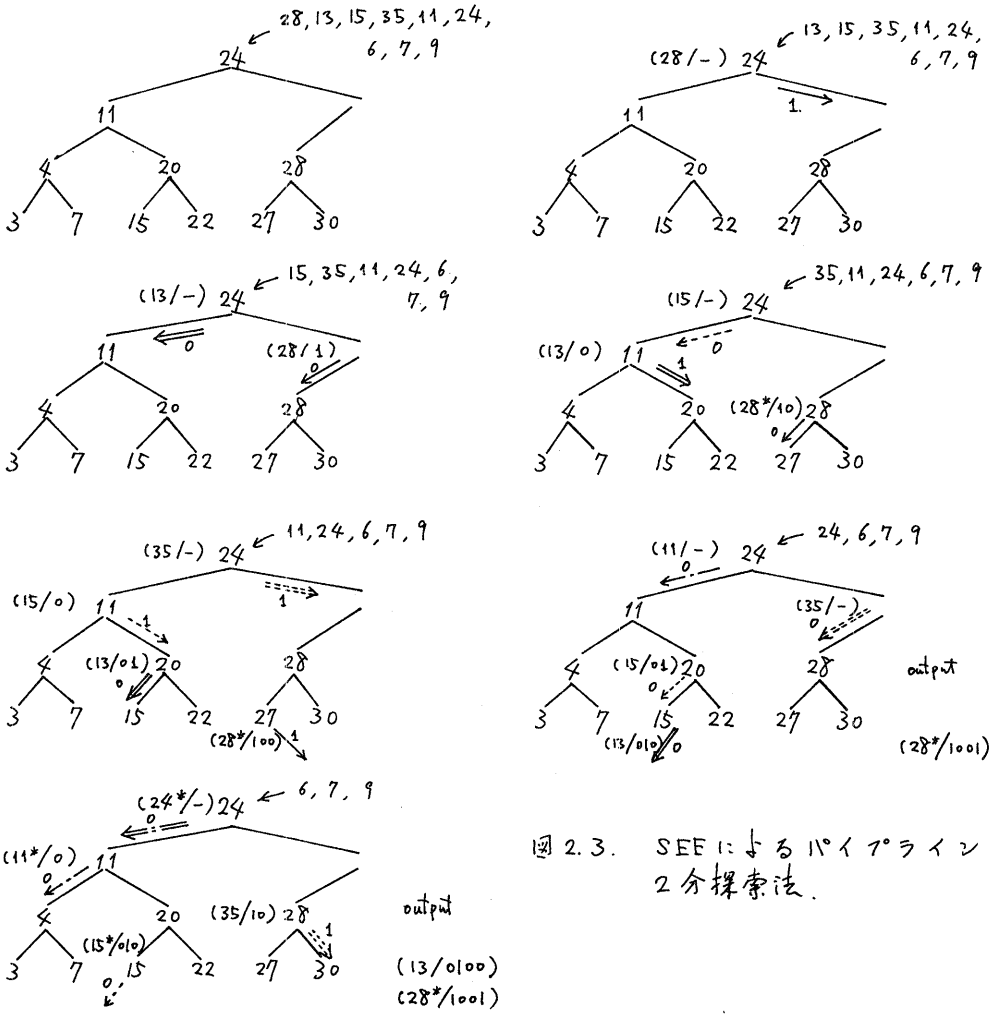
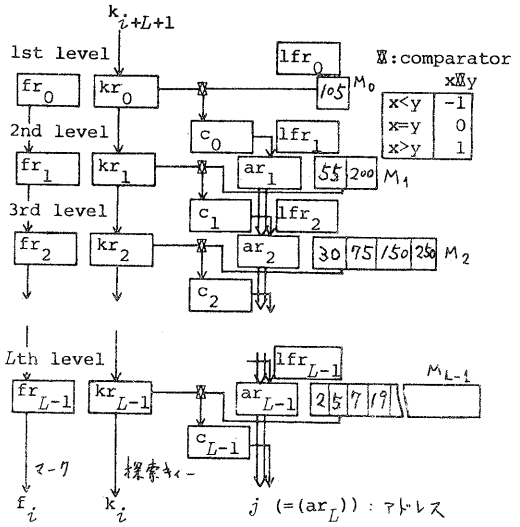


図2.3. SEEによるパイプライン2分探索法

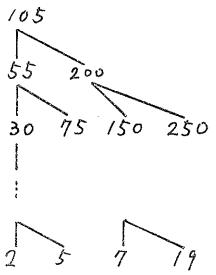
2分探索原理に従い，各探索キィーはルートを起点に下方へと探索が進められるが，この過程で各レベル毎に"0"か"1"の1ビットづつを情報として付加される。探索が左下へ進むときは，それまでに得られた情報の最後尾に"0"が1ビット付加され，右下へ進むときは"1"が1ビット付加される。探索が空ノードに遭遇したときは左下に進むものとし，探索キィーに等しい被探索キィーを持つノードに遭遇したときは探索キィーにマークを付け左下に進む。図ではマークを"*"で表わしている。左先頭2分木において，各レベルでのキィー比較の処理は独立であり，図2.4に示すような，各レベル毎に独立のメモリーと比較器を備えたアーキテクチュアにより，図2.3に示すようなパイプライン探索処理が可能である。[14] 図2.3において，(28*/1001)の1001(=9)は探索キィー"28"が10(=9+1)番目の被探索キィー K_9 と等しいことを表わしており，(13/0100)のように見つからなかった探索キィーに対しては，0100(=4)は $K_{4-1} < 13 < K_4$ であることを表わす。

被探索キィーに値の重複がある場合には最小番号のキィーを探索する。探索法と被探索キィーの先頭法を少し変更することにより，最大番号のキィーを探索するようにもできる。この2つの探索モードを用いることにより，探索キィーに等しい被探索キィーが何番目から何番目までの間に連続しているかを2個のSEEで同時に調べることができる。



fr_l: flag register, kr_l: key register,
lfr_l: loading factor register,
ar_l: address register, M_l: memory

2.4 (a)



2.4 (b)

図2.4. SEEのアーキテクチュアの概略

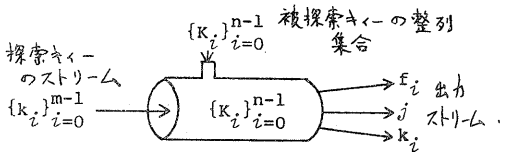


図2.5. SEEのハードウェア・シンボル

図2.4のマークというのは，図2.3中の(ス/8)における"*"をさし，アドレスというのはjをさす。被探索キィーは図のように各レベルのメモリーに格納される。格納順の規則性により，格納は簡単なハードウェアで実現できる。各キィーの適当な位置への格納は，キィーの他デバイスからの転送に遅れることなく処理され，転送過程と整列集合から左先頭2分木への変換過程は完全に重畳される。

図2.5にSEEのハードウェア記号を示しておく。

被探索キィーの各K_iに対して，対応するレコード部R_iが存在し，順番iでなくR_iを求めることが必要な場合には，SEEの出力ストリームのアドレス部iを各々R_iに変換する変換器をSEEの後に置けばよい。これは通常のRAMに他ならない。

SEEはn個の整列した被探索キィーとm個の探索キィーの一括探索を$O(m+\log n)$の時間で処理することができる。試作システムのレベル数と語長は12レベル／2バイトである。SEEはIチップ化に適している。

3. ソートエンジン（SOE: Sort Engine）

3.1. ソートエンジンの機能

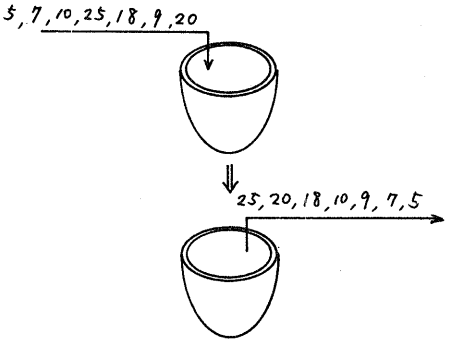


図3.1. SOEによるキィー集合の整列処理.

DSDBCでのソート処理はSOEで行う。SOEは勝手な順序で並んだキィーを，キィーの値の順序に並べかえる（図3.1）。

SOEはキィー集合の他デバイスからの転送入力過程と整列処理の一部を重畳させ，整列処理の残りと整列結果の他デバイスへの転送出力過程とを重畳させ，最後のキィーが入力され終るやいなや，整列結果の最初のキィーの出力を行うことができる。

3.2. ヒープとパイプライン・ヒープソート

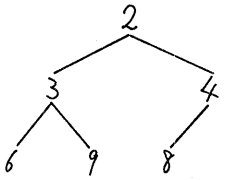


図3.2. ヒープの例.

SOEで用いるデータ構造は，ヒープソートで用いられるヒープである（図3.2）。SOEで用いるアルゴリズムは基本的にはヒープソートであるが，ヒープソートそのものは並列処理やパイプライン処理に不向きである。著者等は，ヒープソートとは異なる方法で，ヒープソートと同じ効果をもたらし，しかもパイプライン処理に適したアルゴリズムを新たに開発した。パイプライン・ヒープソートが通常のヒープソートと同じようにヒープ構成とヒープ出力の2つのフェーズを持つ．順序を変えて，先にヒープ出力を説明する。

図3.3の(0)~(7)に図3.2に示すヒープの出力過程を示す。図3.2のヒープは，レベル数3のSOEでは図3.3の(0)のように格納され，キィーの格納されないノードには"∞"が入る。"∞"はSOEの語長で表現できる最大数を表わし，キィーがこの値をとることはないと仮定する。

ヒープ出力の動作では，奇状態と偶状態が交互に繰り返される。図中，丸で囲ったキィーは，そのキィーが既に親ノードへ転送されており，キィー値はまだそのノードに残ってはいるが，論理的にはそのノードがキィーを持たない"空孔"状態になっていることを示す。奇状態（偶状態）では，ルートから数えて奇（偶）数番目のレベルに1個の空孔が存在し，偶（奇）数番目のレベルでは，空孔の下に位置する1対のキィーの間の比較が行われ，小さい方のキィーが上の空孔へと転送される。最下位レベルに空孔が生じると，次のステップに移る前に"∞"が入

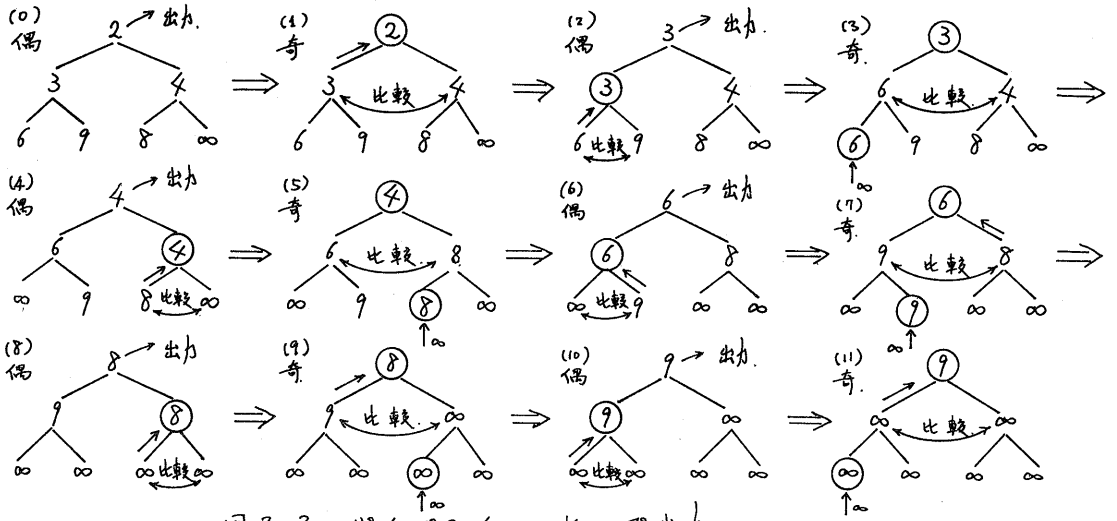


図 3.3 パイプライン・ヒープ出力

れられる。偶状態でルートに位置するキィーが出力されるが，出力を繰り返す内にルートにいつか必ず"∞"が転送されてくる。このとき，ヒープ出力は終了する。終了時には，ヒープの各ノードがすべて"∞"となることに注意されたい。ヒープ出力に要する時間は，キィー数をnとして O(n) である。

ヒープ構成は，ヒープのレベル数に比例するような連想的論理回路によって処理する。図3.4の(a)のようなヒープにキィー"5"を挿入するには，図の次経路上のデータを同時に読みだし，これに"5"を挿入し，再び次経路に同時に書き込めばよい。こうして得られる木はヒープの性質を常に満足する。ヒープ構成が O(n) で実行できる。

図3.5にはSOEのアーキテクチュアの概略を示す。試作中のものは12レベル／2バイトである。SOEがチップ化に適したアーキテクチュアを持っている。

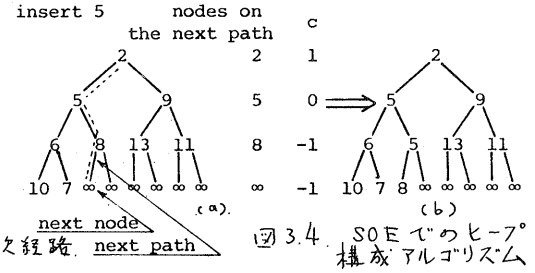
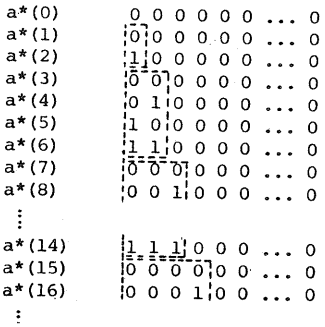


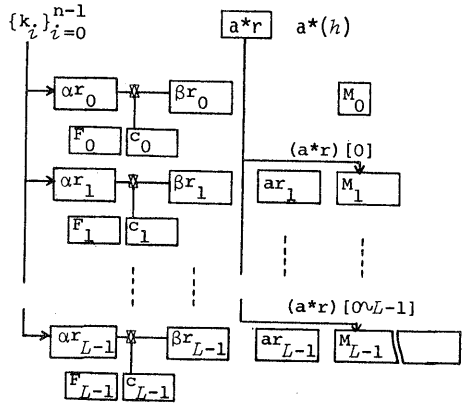
図 3.4 SOEでのヒープ構成アルゴリズム



◀ 3.5(a)

a*r の発生するアドレスのシーケンス

3.5(b) ▶



αr$_l$, βr$_l$: registers, F$_l$:even/odd flag,
M$_l$: memory, ar$_l$: address register,
a*r: address generator.
(a*r)[0∿l] : the leftmost l+1 bits of the register a*r.

図 3.5 SOEのアーキテクチュアの概略

以上の説明では，キィーのみを考えてきたが，キィーとレコードの対をキィーに関して整列しなおすためにはSOEと殆ど同じアーキテクチュアを持つSOEエミュレータ（SOES）でSOE中の対応するキィーの動きをレコード部でシミュレートすればよい。これに必要なSOEからSOESへ送られる信号線は，halt信号と，各レベルの比較結果の出力線で，（レベル数）＋1本である。

SOEのハードウェア記号は図3.6に示すものを用いる。

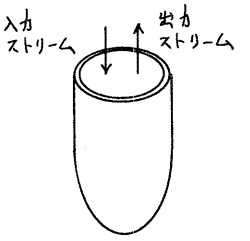


図3.6. SOEのハードウェア・シンボル

4. DSDBCの概容

SOE，SEEは適当な結合により，処理可能なデータ数とデータの語長に関して機能を拡張することが可能である。(4)

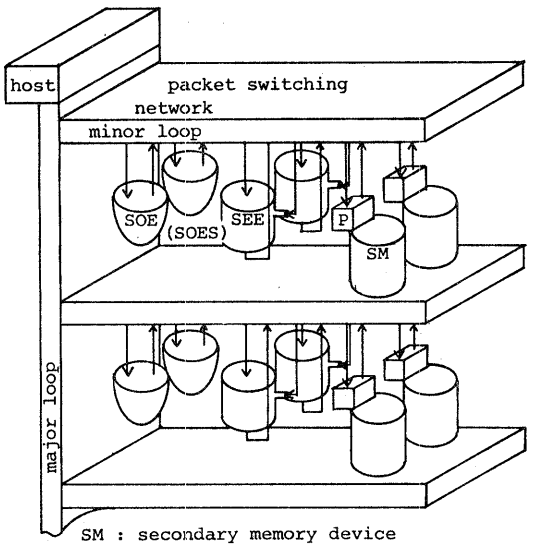
DSDBCは，SOE，SEEの多数のモジュールと，インテリジェント2次記憶装置をコミュニケーションネットワークで結合したDBマシンである。図4.1に現時点でのDSDBCの構想を示す。DSDBCはデータストリームを許すように拡張したデータフロー計算機の1種と考えることができる。

DSDBCのデータフロー言語と，これを用いた関係演算式に対するいくつかのプログラム例は(15)を参照されたい。これらの研究により，交差演算であるジョインやディヴィジョンの所要時間がO(n)になることがわかっている。

例として，全システムの転送レートが5×10^5項目/secであるとき，10^6個の組を持つ2つの関係をジョインし，結果として，やはり10^6程度の組を持つ関係を生成するのには，10秒程度しか要しない。

SOEの採用により，転送ファイルは動的に高速に作成することができ，補助的なファイルは一切必要としない。

このように，DSDBCは，時間と空間の効率において優れたDBマシンと考えられる。



SM : secondary memory device
(e.g. magnetic bubble device)
P : microprocessor (file processor)

図4.1. DSDBCのアーキテクチュアの概略

謝辞．貴重な御助言を頂く東大宇宙航空研究所大須賀節雄助教授と電総研古川康一氏に深謝します。

参考文献．

(1) Canaday et al, CACM Vol.17, pp575-582, 1974
(2) Ozkarahan, E.A., Proc. AFIPS, Vol.44, pp379-387, 1975
(3) Copeland, G.P. et al, Proc. 1st Annual Symp. Computer Architecture, 1973.
(4) Su, S.Y.W. et al, ACM TODS, Vol.3 No.1 pp57-91, 1978.
(5) Lin, C.S. et al, ACM TODS, Vol.1, No.1, 1976.
(6) Schuster et al, IEEE Trans. Computers Vol.C-28, No.6
(7) Ozkarahan et al, Proc. 4th VLDB, pp300-311, 1978
(8) 田中譲他，信学研資 EC76-77, 1976.
(9) 田中譲他，四学会連合大会論文集，1977.
(10) Babb, E., ACM TODS, Vol.4, No.1, pp1-29, 1979
(11) McGregor, D.R. et al, Proc. 2nd VLDB, pp103-116, 1976.
(12) Batcher, K.E., 1968 SJCC, pp307-314.
(13) Nassimi, D. et al, IEEE Trans. Computers, Vol.C-28, No.1
(14) Tanaka, Y. et al, Proc. IFIP '80, 1980 (to appear).
(15) 田中譲他，昭和55年度情報処理学会全国大会論文集 pp.493-494. 1980.