(15)

EC82－32

パイプラインマージソータの構成

喜連川　優・伏見信也
桑原和宏・田中英彦
元岡　達
（東　大）

1982年6月25日

社団法人 電子通信学会

EC82－32

パイプラインマージソータの構成

THE ORGANIZATION OF PIPELINE MERGE SORTER

喜連川 優　　伏見 信也　　桑原 和宏　　田中 英彦　　元岡 達
M. Kitsuregawa　S. Fushimi　K. Kuwabara　H. Tanaka　T. Moto-oka

東京大学　　工学部
Faculty of Engineering, University of Tokyo

§1. 始めに

データの高速ソーティングは、種々の分野で必要とされる基本的な操作である。 特に関係データベースに於ける関係代数演算にはJoin，Projection 等処理負荷がO（N^2）のものがあり、関係データベース処理の高速化の為にはこれら処理負荷の重い演算を効率良く実行する事が不可欠である。一方で、リレーションが予めソートされていればこれら処理負荷の重い演算もO（N）時間で処理が可能であり、従ってソートをO（N）化する事が可能であればこれら演算の処理時間は全体としてO（N）となる。 我々は既にデータベースマシンGRACEの基本要素としてデータストリームに沿った処理を可能とするハードウェアO（N）ソータを開発してきた［1－5］。 O（N）時間でソートを行なうハードウェアソータの提案は他にも少くないが、本ソータは実際の使用環境に於いても柔軟に対処でき、様々な分野に有効に利用できると考えられる。

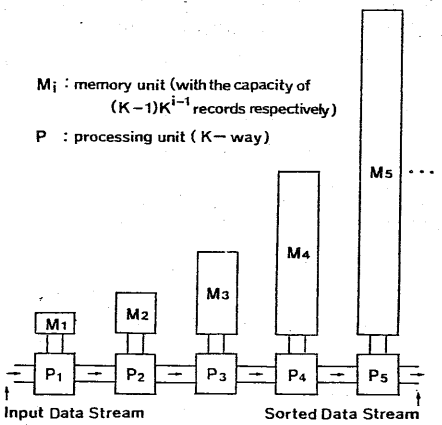

Fig.2.1 Global Architecture Of Stream Driven Sorter

§2. 基本構成

2．1 ソートアルゴリズム

ソートアルゴリズムはパイプライン化されたマージソートを基調としている。 今、N（$=K^n$）ケのレコードのソートを行なうものとすると、K－way merge を行なうプロセッサを n（$=\log_K N$）台用意し、これらを1次元状に結合する。（図2．1） 第i 番のプロセッサは、$K^{i-1}\cdot(K-1)$ レコード分のメモリを持つ。 Nヶのレコードはシリアルに第1番目のプロセッサに入力され、第i 番目のプロセッサは第 i－1番目のプロセッサから送られてくるK^{i-1}ヶのレコードからなるソートされたストリングをK本マージしてK^i レコードから成る一本のストリングを生成し、第 i＋1番目のプロセッサへ送出する。 ここで、プロセッサはマージすべきK本のストリングの内K－1本をメモリにロードした後、K本目のストリングの最初のレコードが到着した時点でマージ処理を開始することができる。 パイプラインのレベルはレコードレベルから、バイトレベル、ビットレベルまで落とす事が可能である。

2．2 性能諸元

入力ストリームの最初のレコードが到着した時点から出力ストリームの最後のレコードが送出されるまでの全ソート時間は、

$$2N+\log_K N-1 \sim O(N)$$

とあらわされる。 この内、2Nの項は一旦全てのレコードをソータに入れ、その後再び外に取り出す為の時間であり、実質的なオーバーヘッド時間は$\log_K N-1$となりきわめて短い。

又、ソータ全体に必要とされるメモリの総和はNレコード分の容量となる。

2．3　プロセッサ内の処理

プロセッサは、前述のようにK本の入力ストリームを1本にマージして出力することを繰り返す。 この動作は、次のように3つのPhaseに分けることができる。 即ちi 番目のプロセッサP iは以下のPhaseを状態遷移する。

Phase0： K－1本のストリングをメモリ内にロードする。ロード終了後、Phase1へ。

Phase1： メモリ内のK－1本のストリングと入力されてくるK本目のストリングをマージして出力する。K本目のストリングの入力が終了するとPhase2へ。

Phase2： メモリ内に残存するストリングのマージ操作を行ないつつ、次段マージ用のK－1本のストリングの入力を行なう。 K－1本のストリングのロードが終了すると（同時に残存ストリングのマージも終了し）Phase1へ。

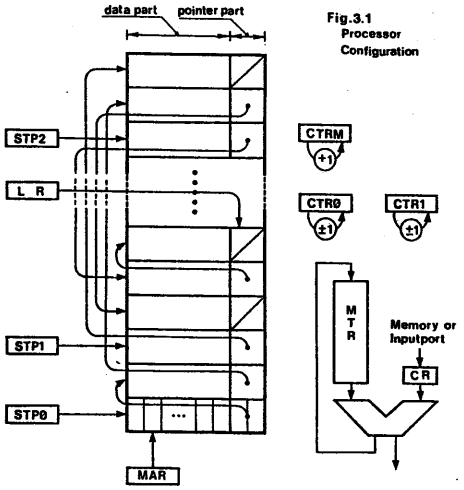

Fig.3.1 Processor Configuration

2．4　メモリの管理

第i 番目のプロセッサは、K－1本のストリングを入力した後、K本目のストリングの1レコードを入力しては比較を行ない、1レコード出力していく為有効なメモリ容量は常に一定である。 一方で入力ストリングは各々ソートされており、出力レコードの占めていた領域に入力レコードをそのままロードしていくのではソート順は保てない。 そこで何らかの方法でこの順序を維持する必要がある。 これに関しては既にいくつかの方法を提案し、優劣を検討した［4］。

§3．　レジスタトランスファレベルアルゴリズム

3．1　概要

ソータの試作に先立ち、レジスタトランスファレベルのより詳細なアルゴリズムの記述を行なう必要がある。 ここではway 数を2とし、バイトレベルの結合、ポインタによるメモリ管理方式を採用した。 即ち前段のプロセッサから送られてきたストリングはそれぞれメモリ中でポインタを用いて linked list として管理される。ポインタ長は2バイトとした。 各レコードはキー部、非キー部を合せたデータ部とポインタ部からなるフォーマットでメモリに格納される。（図3．1）

3．2　バイトレベルパイプライン

バイトレベルで完全パイプラインを実現することは、処理レート、即ち各プロセッサが1バイトの比較を行なうのに必要な時間を十分な余裕を持って与えれば常に可能である。 従って問題は完全パイプラインが可能な最短の処理時間を求め、これを実現することである。 一般に1バイトの比較操作を行なう程度のハードウェアは高速であり、各プロセッサの比較処理に於てはメモリアクセスがネックとなる。 従って1バイト処理に対するメモリへのアクセス回数がソータの処理レートを定めることになる。§2での議論から明らかな様に、一般にPhase1，2の処理中はマージすべき2本のストリングがメモリ中に存在し、この状態に於けるメモリアクセス回数が最も多い。 即ちこれらストリングから各々1バイトのデータを取り出して比較器へ入力する為に2回、更に入力されてくるレコードをメモリに書き込む為に1回のメモリアクセスが必要となる。 従ってこの場合は高々1バイト／3クロック（1クロック＝1メモリアクセス時間）の処理レートしか得られない。

そこで、比較器入力の一方にレコード長を持つレジスタ（MTR）を割り当て、一方のストリングの先頭レコードを常に保持させることとした。 比較入力のもう一方には1バイトのラッチを割り当て（CR）、入力レコードを1バイトずつ与える（図3．1）。 これにより、1本の入力ストリングは常にレジスタ空間に存在するように仮想化されたことになり、メモリへの比較データ読み出しの際のアクセス競合は回避される。 他方、メモリへの書き込み／読み出しのアクセス競合を回避することは原理的に無理であり、ここでは1バイト／2クロックの処理レートを採用した。 具体的に

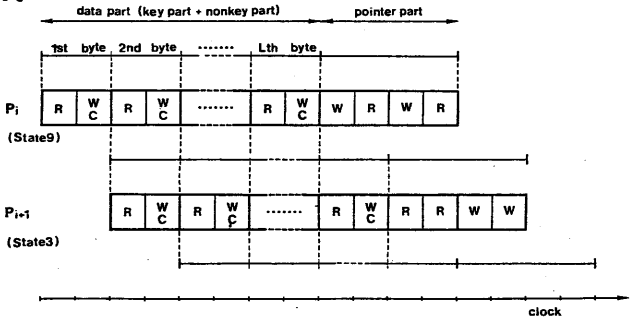

Fig.3.2 Pipeline Flow Example

は始めのクロックをメモリ読み出し（ read clock ）に、後のクロックをメモリ書き込み（ write clock ）に割り当て、read clockではメモリ読み出し、及び比較の準備操作を行ないwrite clock でメモリ書き込みと比較を並行して行なうこととした（図3．2）。

3．3 比較器

以下、本稿を通じて昇順ソートを仮定する。　前節で述べたように、比較器への入力は1レコード長を持つレジスタMTRと1バイトのレジスタCRから成る。 比較の結果、大きい方のレコードは次段のプロセッサに送り出され、小さい方のレコードは常にMTRに帰還する。この際、MTRはシフトレジスタとして機能する。 比較の結果は、以下のように定義された2ビットのフラグ rf に与えられる。

rf ＝ 00： 大小関係はまだ定まっていない。
01： CRに入力されたレコード>MTRに入力されたレコード
10： CRに入力されたレコード<MTRに入力されたレコード
11： don´t care

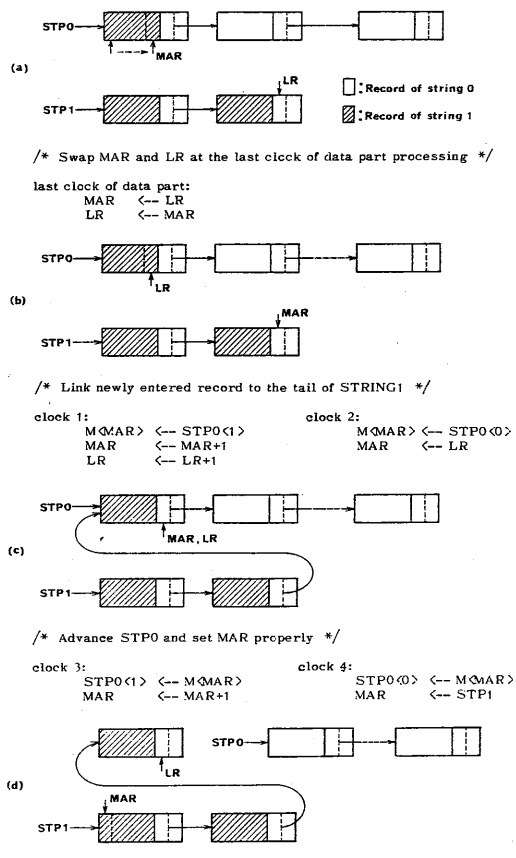

Fig.3.3 Pointer Manipulation

一方、 rf はレジスタを識別するだけであり、マージすべき2本の論理的ストリング（ string 0 及び 1）とこれら物理的レジスタの対応関係を管理する必要がある。2－way の場合、これは1ビットのフラッグ（ s01）を用いた簡単な論理操作で実現できる。 即ち、

s01 ＝ 0： string0 →CR、
string1 →MTR
1： string0 →MTR、
string1 →CR

とすれば、

s01 ← ～（ s01⊕ rf ［0］）
（ rf ［0］は rf の下位ビット）

によって s01は矛盾なく更新される。 以上の構成によって、通常のマージ操作だけでなくMTR内のレコードの追い出し（1－way マージ）等も容易に行なうことができる。

3．4 ポインタ操作とメモリ管理

メモリ内でストリングを linked list として管理する為には、リストの最後尾へのレコードの追加、及び最前部からのレコードの削除の機能が必要である。 この為、実装にあたっては次のようなレジスタ構成を考えた。

STP i（i ＝ 0,1,2）： string i のメモリ中の先頭アドレスを保持する。
LR ： 最も最近にメモリにロードされたレコード（Phase iでは、string iのリストの最後尾のレコードとなる。）のポインタ部の第1バイトを指す。
MAR ： メモリアドレスレジスタ

ポインタ操作に与えられる時間はポインタ2バイト分、即ち4クロックであり、この時間内でリストの削除、追加処理、レジスタ値の更新等を行なう必要がある。 ポインタ操作の典型を図3．3に示す。 図はPhase 1中のスナップショットで、 s01＝0即ち、string 0の先頭レコードをメモリから1バイトずつCRに取り込み、入力されてくるstring 1のレコードをメモリ中に生じた空きバイトに格納している状態を示す。 MARがポインタ部の1バイト手前までくると、MARとLRがスワップし（MARとLRの間に値を互いに交換するスワップパスを仮定する）、例えば、 rf ＝10（即ちMTR＞CR）のときは図3．3（a ）～（d ）のように処理が進む。 図3．3（d ）に示されるように4クロック終了した時点で、各レジスタは矛盾のない値を保持し、リスト操作が適切に行なわれたことがわかる。

3．5 状態遷移

§2で述べた3つのPhaseは更に細く9つの stateに分けられる。 プロセッサはレコード間の大小比較の結果に従って図3．4に示すような状態遷移を繰り返す。　以下、各 stateについて説明する。

<Phase 0>

string 0 のメモリへのロードを行なう。 ロード終了と共にPhase1へ。

[state 0]： まず string 0 の先頭レコードをMTRにロードする。ストリング長が1の場合はこれで string 0 のロードが終了したことになり、 Phase1.state2へ。 そうでなければ state1へ。

[state 1]： string 0 の残りのレコードにリンク操作を施しつつメモリ内にロードする。 最終レコードまでこの状態でロードを続け、Phase1.state2へ。

<Phase 1>

string 1 の入力がPhase1に対応する。 このPhaseで string 1と string 0 のマージが行なわれるが、 string 1 のレコードをメモリにロードするか否かで状態は大きく2つに別れる。 Phase1での比較の結果、一方のストリングが他方のストリングよりも完全に大きい時はPhase2.state7へ、そうでない時は同 state5へ遷移する。

[state 2]： 入力されてくる string 1 の先頭レコードと、MTRにある string 0 のレコードの比較を行なう。 string 1 のレコードが大きければ、この状態のまま、次に入力されてくる string 1 のレコードとMTRの string 0 との比較を続ける。 この stateで string 1 の最終レコードの比較を行なった場合、その比較に於ても string 1 のレコードよりも大きければ、 string 1 は string 0 よりも完全に大きかったことになり、Phase2.state7へ、最終レコードのみ string 0 のレコードが大きかった場合にはPhase2.state5へ、最終レコード以前のレコード比較で string 0 のレコードが大きかった場合には state3へ各々遷移する。 またストリング長が1の場合は、この stateで1度比較を終了するとストリングとしてどちらか一方が完全に大きかったことになり、無条件にPhase2.state7へ遷移する。

[state 3]： state3での比較開始時点では string 1 の先頭レコードがMTRにあり、これとメモリ内にある string 0 の先頭レコードとが比較される。 この際、 string 0 の先頭レコードが格納されていた領域には入力されてくる string 1 のレコードがロードされて行く。 このように当 stateで string 1 のレコードが始めてメモリにロードされるが、当 stateでの比較の結果、 string 1 のレコードが大きかった場合この1レコードで閉ループを形成していなければ、次のマージ終了後ポインタの値が定まらなくなる。 当 stateではこの為の閉ループを生成する。 state終了時点で string 1 のロードが終了していなければ state4へ。 この state終了と共に string 1 の全レコードの入力が終了した時、 string 0 の方が string 1 に比べて完全に大きい場合（これはストリング長が2の場合に限る）Phase2.state7へ、そうでなければPhase2.state5へ。

[state 4]： この状態に入ると、以降、 string 1

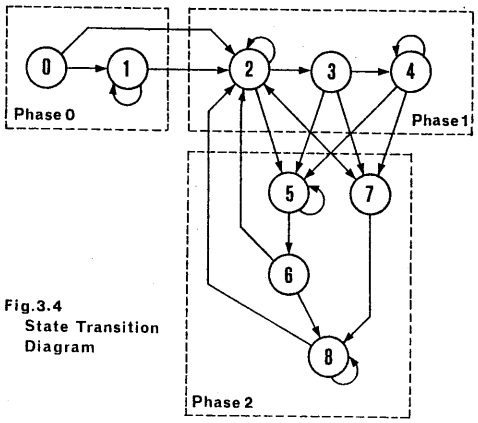

Fig.3.4
State Transition
Diagram

の入力が終了するまでこの状態でマージを続ける。 string 1 の入力終了時に、 string 0 が string 1 に比べて完全に大きかった時はPhase2.state7へ、そうでなければ同 state5へ。

<Phase 2>

string 2（次のマージの string 0 ）が入力される間（或いはメモリ内の string 0 ,1 のマージ処理が終了するまで）Phase2の状態をとる。 このPhaseでは、まず string 0 又は1のどちらか一方のストリングが送出されてしまうまでマージを繰り返す（ state5）。 この間、入力されてくる string 2のレコードはメモリ内に格納しておく。 その後は残りのストリングを追い出すだけであるから、一方が送出された直後にstring 2の先頭レコードをMTRにロードする（ state6）。 state7はPhase1で既にこの状態が達成されていた場合（一方が他方より完全に大きい場合）である。残ったストリングは state8で送出され、その後Phase1.state2に戻って処理が繰り返される。

[state 5]： Phase1での比較の結果、一方が他方よりも完全に大きくない場合、この stateに遷移してくる。ここではメモリ内に残存する string 0 及び1のレコードを、一方が送出されてしまうまでマージを繰り返す。 この間に入力されてくるstring 2のレコードはメモリ内にロードされる。 一方のストリングが消費されると state6へ。

[state 6]： この状態に遷移した直後は残されたストリングの先頭レコードがMTRに格納されている。 そこで、このレコードを次段プロセッサに追い出し、MTRにはstring 2の先頭レコードをメモリから移す。 string 2の先頭レコードが格納されていたメモリ領域には、この時点で入力されてくるstring 2のレコードが入る。 この state終了時に、 string 0 ,1 共に全てのレコードが消費されれば、string 2を string 0 としてPhase1.state2へ。 そうでなければ state8へ。

[state 7]： Phase1での比較の結果、一方のストリングが他方のストリングよりも完全に大きかった場合に

この状態へ遷移する。 この時、残されたストリングの先頭レコードはMTRにある。 このレコードを追い出して、入力されてくるstring 2の先頭レコードをMTRにロードする。 ストリング長1の場合は、これで string 0 , 1 のマージが完了したことになる故、string 2を string 0 としてPhase1.state2へ。 そうでなければ state8へ。

[state 8] : この stateに遷移した時点では string 0 又は1の一方のみがメモリ内に残っており、MTRにはstring 2の先頭レコードが入っている。 当 stateでは、メモリ中に残存するストリングを追い出し、入力されてくるstring 2のレコードを格納する。 残存ストリングの追い出しが終了するとstring 2を string 0 としてPhase1.state2へ。

以上の状態遷移、及びレジスタトランスファレベルでの制御の正当性は、Pascal で書かれたシミュレーションプログラムによって検証した。

3．6 ハードウェアリソース

以下、プロセッサが持つレジスタ、フラグについて簡単にまとめる。

◇ MAR (Memory Address Register) [16ビット] メモリアドレシングに用いる。アクセス単位はバイト。

◇ LR (Link Register) [16ビット] ポインタのリンク用レジスタ

◇ STP i (i-th String Top Pointer) [16ビット] メモリ内に於るstring iの先頭レコードのアドレスを保持する。

◇ CTRM (Master Coun Te R) Phase間遷移の為に入力ストリングのレコード数をカウントする。

◇ CTR i (Coun Te R for i-th string) メモリ内に存在するstring iのレコード数をカウントする。

◇ MTR (Merge Top Register) 一方のストリングの先頭レコードを保持する比較器の入力レジスタ。

◇ CR (Comparison Register) [1バイト] 比較器への入力データを保持する。

◇ RLC (Record Length Counter) state遷移の為にバイト単位でレコード長をカウントする。

◇ rf (Result Flag) [2ビット] CRとMTRの比較結果を示す。

◇ s01 (String flag) [1ビット] 次段プロセッサに出力されたストリングの番号を保持する。

§4． 試作プロセッサの構成

4．1 プロセッサの入出力

図2．1に示されるソータのプロセッサを一台試作した。その入出力は図4．1に示される如く、前段からのデータバスと次段へのそれ、及び当該段のメモリバンクに対するアドレスバス、データバス、そして数本の制御線から成る。 データの幅は1バイト、アドレスは16ビットとしている。 各プロセッサにはクロック及びリセットが共通に供給されている。

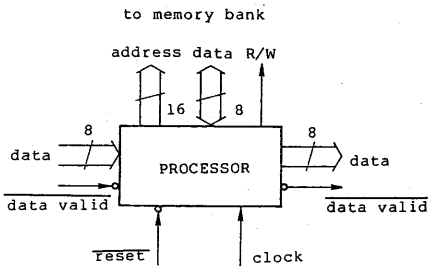

FIG.4.1 INPUT & OUTPUT LINES OF A PROCESSOR TO MEMORY BANK

4．2 プロセッサの内部構成

§3で既に述べた如く、プロセッサ内のハードウェアリソースとして、MAR、LR、STPi 、CTRM、CTRi 、MTR、CR、RLC、 rf 、s01 が必要であった。 比較器の一方の入力としてMTRをシフトレジスタと考えていたが、ここでは、RAMとカウンタによる実装を採用する事とし、MTR (レコードの先頭1バイト分を保持するレジスタ) 、MTM (先頭バイト以外のレコード部分を保持するメモリ) 、TOP (MTMのアドレス保持レジスタ) なるリソースを用いる事にした。 又、各種パラメータを初期化する為以下のレジスタを設けた。 ITOP (Initialize Register for TOP ：キー長保持レジスタ) 、IRLC (I

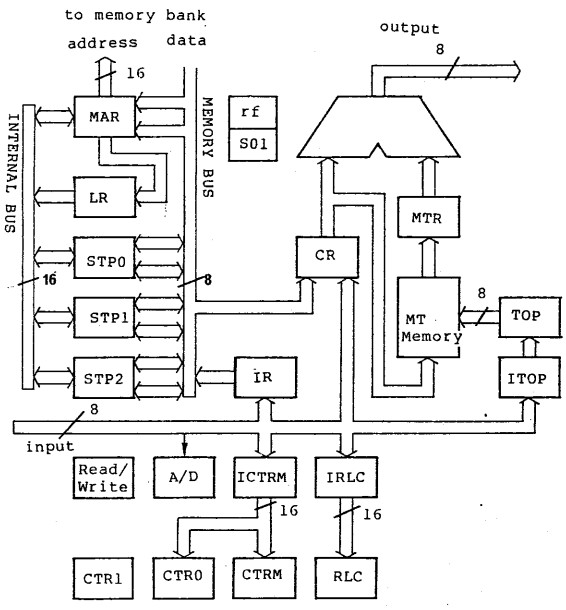

FIG. 4.2 INTERNAL STRUCTURE OF A PROCESSOR

nitialize Register for RLC ：レコード長保持レジスタ）、ICTRM（Initialize Resister for CTRM ：ストリング長保持レジスタ）、その他、IR（入力データー時保持レジスタ 8bits）、A／D（昇順降順ソートを指定する為のフラグ 1bit ）、R／W（read cycle 、write cycle を示すフラグ 1bit ）を設けた。プロセッサの内部構成を図4．2に示す。

4．3 プロセッサの制御方式

O（log N）台プロセッサ構成によるハードウェアソータでは、O（N）台プロセッサのソータに比べてプロセッサ部分の占めるオーバヘッドは大きく減少出来る。 従って、逆に、各プロセッサに高い柔軟性を持たせる事が可能となる。我々は§6で述べるような種々の機能拡張について検討を行なっている。 従って、試作に当たってもこれらの成果を受け入れ易い柔軟な構成とする為、制御方式としてマイクロプログラム方式を採用する事とした。

一般にマイクロプログラム方式では主記憶と制御記憶を区別し、後者のアクセスタイムは前者のそれに比べてかなり速い場合が多い。 本ソータでは、将来充分な集積度が得られた場合、メモリとプロセッサを同一チップとすることが考えられる。 よって、試作に当っては制御記憶とデータ用記憶にはアクセスタイムの等しいメモリを用い、水平プログラム方式による制御を行なう事にした。 データ1バイトの処理は2つのクロック（2回のデータメモリアクセス）で構成される為、当該処理は2つのマイクロ命令で実現する必要がある。 又、マイクロ命令の取り出しと実行はパイプライン化されている。

4．4 マイクロプログラムの順序制御

マイクロ命令の順序制御を実現する場合、特別に分岐命令を持たせる事も可能である。 ここでは1バイトを2つのマイクロ命令で実行する必要があり、バイトレベルの完全なパイプラインを維持する為、命令内に次アドレスのフィールドを設ける事にした。 又、図3．4に示される状態遷移からもわかるように多重分岐をおこなう事が多い。 これに関してはブランチ制御用のフィールドを設け、これによって指定したステータスビットにより次アドレスを修飾することとした（図4．3）。 ここでは下位の数ビットを修飾しており、多重分岐に伴うアドレス境界は意識する必要がある。

又、マイクロ命令のフェッチ、実行のパイプライン化の為、分岐条件は分岐の1クロック前に確定していなければない。本ソータではレコード本体に続く2バイトタイムのポインタ処理に条件分岐が集中し、タイミング的に不適切な場合が生ずる。 そこで、これらについてはマイクロ命令自体を条件によって直接修飾することにしている。 これによって当該条件は実行の直前に確定していればよいことになる。 これによりマイクロ命令のデコード回路はより複雑になるが、分岐数は少くなり、制御記憶容量の減少が期待出来る。

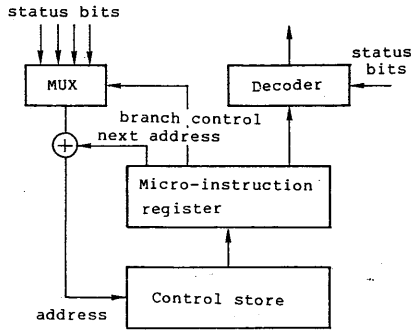

FIG. 4.3 SEQUENCE CONTROL OF MICRO-INSTRUCTION

4．5 プロセッサ間結合とタイミング

プロセッサは、8bit のデータ線と data valid なる制御線によって結合される。 本ソータはバイトレベルのパイプラインを実現しており、1バイトのデータを入力すると共に、1バイトのデータを次プロセッサへ出力する。 ポインタの処理はレコードに続く2バイトタイムに行なわれる。 図4．4に処理のタイミングを示す。

又、制御線 data valid を high にすることによってプロセッサを一時的に停止させることが出来る。 この機能により、入力データストリームが一時的に中断してもパイプラインを乱さずに柔軟に対処出来る。 data valid は2クロック遅れて次段のプロセッサへ伝搬される。

4．6 初期化

ソーティングに先立ち、各プロセッサに対しレコード長・ストリング長・昇順／降順等のパラメータの設定を行なう必要がある。 この初期設定は、リセットをかけた後、ソートするデータに先だって各定数を入力データストリームとして加える事にした。

各プロセッサは送られてきたデータを所定のレジスタに格納すると共に、CRを介して次段へ送り出す。 ここで、ストリング長は他の定数と異なりプロセッサ何段めに位置するかによって変化し、一段毎に2倍して次段へ送出している。

処理の開始は全プロセッサが初期化されるまで待つ必要はなく、各プロセッサは、その初期化が完了した時点で直ちにソートを開始出来る。

§5． 試作システム

5．1 全体構成

プロセッサ一台だけでは動作確認を行なうことは出来ず、何らかの動作環境を整える必要がある。 又、ハードウェア、ファームウェアのデバッグを容易にする為の支援環境を設けることが望ましい。 今回の試作ではパーソナルコンピータPC8001をサービスプロセッサとして用い、I／Oユニット（PC8012）を介してソートプロセッサを結合する構成を採った。 （図5．1）

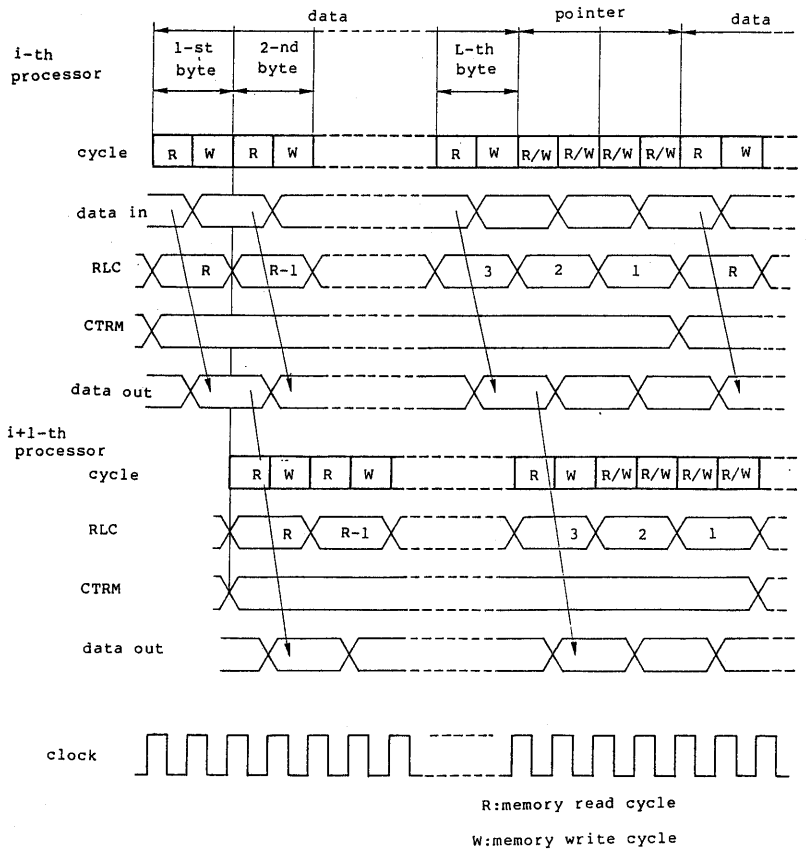

FIG. 4.4 TIMING CHART

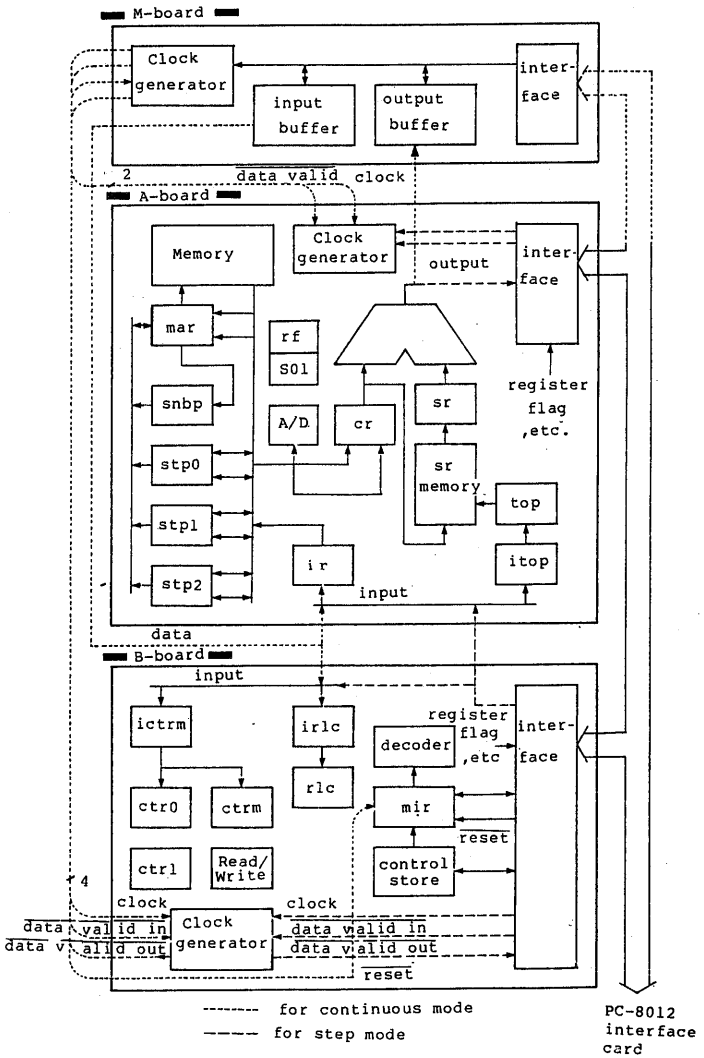

FIG. 5.2 ORGANIZATION OF THE PILOT SYSTEM

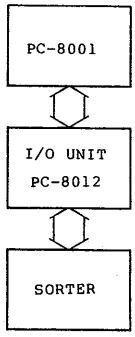

FIG.5.1 OVERVIEW OF
THE PILOT SYSTEM

```
 MAR=0003        CR=02        CYCLE=WRITE  1
  LR=0002       MTR=02          RLC=0001
STP0=0001                      CTRM=0001
STP1=0000      CTR0=0000         TOP=01
STP2=0000      CTR1=0000         MIR=20 15 00
 s01=   1                            CC 01 C0
  rf=  01        IN=05        N.Add=42
               OUT=02         STATE=1

MEMORY = 05 01 00 01 FF FF FF 00     0


 MAR=0000        CR=02        CYCLE= READ   1
  LR=0002       MTR=02          RLC=0004
STP0=0000                      CTRM=0002
STP1=0000      CTR0=0002         TOP=04
STP2=0000      CTR1=0000         MIR=30 08 00
 s01=   1                            01 00 03
  rf=  01        IN=05        N.Add=16
               OUT=02         STATE=2

MEMORY = 05 01 00 01 FF FF FF 00     0
                                     CC=27
```

Fig 5.3 State Information in Step Mode

5．2　ハードウェア実装

プロセッサの試作に於いてはＬＳＩ化への見通しを良くする為、汎用ＴＴＬ及びCMOSメモリを用いることとし、多機能のＬＳＩは使用しなかった。 プロセッサはメモリを含めて２枚の基板に分けて実装した（A，B基板　図5．2）。基板毎にサービスプロセッサとのインターフェイス回路が設けてある。 更に、一枚（M基板）は入出力データストリームのバッファとして用いている。

5．3　デバッグ支援

本ソータでは3．4で述べた如く、比較的複雑な処理を伴う。 ソータを駆動するに当たり、最低

- 制御記憶の読み書き
- 入力データの供給
- 出力データの監視

程度の機能は必要となる。 本ソータでは、よりデバッグ機能を強化する意味で、上記機能に加え各種レジスタ，カウンタ，フラグ，及びメモリの値を読み出せるようにした。 又、マイクロインストラクションレジスタには書き込みも可能なハードウェア構成を採用した。 これによってデバック時間を大きく短縮できた。 又、ソータの動作モードとしては、これらの機能を利用して２つのモードを設けた。

（1）ステップモード

本モードでは、クロックをサービスプロセッサからソフトウェアにより与え、１クロック毎にプロセッサ内の各種リソースの値の変化を追うことが可能である。 入力データはサービスプロセッサから１バイトづつ供給することができる。

図5．3にソートのトレースを行なっている際の画面の一例を示す。 プロセッサ内のほとんど全てのリソースを監視することができる。 トレース用のプログラムはサービスプロセッサ上にBasic言語を用いて書かれている。

（2）連続モード

クロックをM基板上に別に設け、高速動作を行なうことを目的とする。 入力データを入力バッファメモリ上に用意した後、ソータを起動する。 出力は同じくM基板の出力バッファに書込まれソートが完了した後サービスプロセッサが結果を確める。

尚、パイプラインソータはプロセッサを複数台パイプライン結合して構成されるが、試作は一台しか行なわなかったため長いデータストリームを一度にソートすることは出来ない。そこで、各段を順次シミュレートする（毎回プロセッサのイニシャライズが必要）。 すなわち、$\lceil \log_2 N \rceil$ 回データストリームを回すことによってNヶのレコードのソートを実現することとした。

5．4　処理速度

パイプラインソータの処理速度は主にメモリのアクセスタイムで決定される。 これはパイプラインの各セグメントが２回のメモリアクセスで構成されていることによる。 今回の試作では１５０nsecのCMOSメモリを使用しており、６MHz での動作を確認した。 即ち、ほぼ3Mbyte／sec のデータ流に追随することが出来る。 現在のディスクの転送レートは0．5～3Mbyte／sec 程度であり、本ソータによりディスクからのデータ転送に重畳してソートを完了することが可能となる。

§6　機能の拡張

6．1　概説

ハードウェアソータは、現在までに様々なものが提案されており、本ソータと同程度の処理時間を実現しているものも少なくない。 しかし、一方でハードウェア化による種々の制約が発生し、実際の使用環境に於ては大きな問題となっている。 ここではレコード長、ストリーム長等のパラメータ変化に対する柔軟性、及び種々の性能向上の可能性について考え、これらの拡張された機能を実現する為の制御方式について述べる。

6．2　K－way への拡張

試作されたソータは２－way であるが、これは容易にＫ－way に拡張することができる。　この場合も２－way の場合と同様、メモリ管理方式としてポインタ方式を採用することにする。　従って、各ストリングの先頭レコードのアドレスを保持するレジスタ（ＳＴＰ ｉ）は、Phase２に於けるマージ操作から明らかな様に、当該マージ中のＫ本のストリングに対してＫ個（ＳＴＰ ｉ）、また同様に入力されてくる次段マージ用のＫ－１本のストリングに対してＫ－１個（ＳＴＰ ｉ´）、計２Ｋ－１個必要となる。　またストリング長を管理するカウンタもこれに応じてＫ個（ＣＴＲ ｉ）、Ｋ－１個（ＣＴＲ ｉ´）、計２Ｋ－１個必要となる。　一方、メモリのアクセス競合を回避する為には、２－way の場合と同様、メモリからの直接入力は１入力に制限しなければならない。従って、２－way の場合からの自然な拡張として、ＭＴＲをＫ－１個（ＭＴＲ ｉ）、ＣＲ１個を比較器入力とする構成が考えられる。　しかし、この場合比較の結果、あるＭＴＲ ｉにあったレコードが次段に送り出されると、この時のＣＲ入力レコードはこのＭＴＲ ｉに帰還することになり、一般に論理的なストリング番号と物理的なＭＴＲ番号の関係はランダムとなる。　従ってこの対応関係を保持する表が必要となる。２－way の場合は、この表の大きさは１ビットで済み、管理の為の操作も比較的簡単なものであったが、Ｋ－way の場合は表も大きくなり、また表管理の操作も複雑化する。　又、この表へのアクセス時間も無視できず、ＬＳＩ化には不利である。

そこで、ＣＲをＭＴＲ ｉで置き換え、比較器へのＫ個の入力全てに１レコード長を与え、論理的にストリング番号と物理的ＭＴＲ番号を固定的に対応させる構成が有利と考えられる。　また、Ｋ－way では、Phase２に於てマージ数がＫまたはＫ－１から順に１つずつ減少し、１－way マージが終了した後にPhase１へ戻ることになり、一般にＫ以下の任意の way 数でマージが行なえることが望ましい。　２－way の場合は、マージ数は２から１に変化するだけであり、これは３．４で述べたstate ８によって処理されている。

Ｋ－way の場合は、　このようにway 数の減少に対してstate を直線的に展開することは好ましくない。　従って、state は実効マージ数と独立に動作し、各ＭＴＲ ｉにフラグを与えハードウェアレベルでマージ数の変化に対処することとする。

6．3　レコード長変化に対する制御

§３で述べたように、本ソータはキー部、非キー部を分離せず、レコードを直接ソートすることを前提としており、比較器入力のレジスタＭＴＲも全レコード長を割り当てた設計となっている。　しかし、本来非キー部は比較操作には関係なく、これをキー部と分離して格納することができれば、非キー部に対してはほとんど大きさの制限が解消されることになる。　今ＭＴＲ ｉのレジスタ長をＬ（バイト）、ソートすべきレコード長をＸ（バイト）と仮定し、また、便宜上、レコードのキー部を含む先頭のＬバイトを改めてキー部と呼ぶことにすると、上記の議論から図６．１のようなメモリ／ＭＴＲ ｉの割り当てが考えられる。

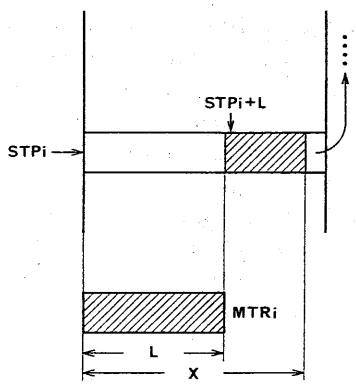

Fig.6.1　Memory/Register Allocation For Long Record

このようなレコード配置を行なった場合、キー部はＭＴＲ ｉに存在しており、メモリへのアクセス回数が増加することはなく、基本的には１バイト／２クロックの処理レートを維持できる。　しかし、一方で、各ストリングの先頭レコードのキー部と非キー部がＭＴＲ ｉとメモリに分離されてしまい、string ０ ,１ のいずれの領域が解放されるかはキー部の比較が終了するまで判明しない為、入力されてくるレコードに対してその時点で連続した１レコード分の空間を確保することが困難となる。　そこで、予めメモリ内に１レコード分の write buffer 領域を用意し、書き込み専用のアドレスレジスタＭＡＲＷを用いて入力レコードを常にこの領域に書き込むこととし、一方、メモリからの読み出しには読み出し専用のアドレスレジスタＭＡＲＲを用いる構成が考えられる。　この構成では、入力レコードに対して常に１レコードの領域が保証され、上記の問題は解決される。　比較の結果、次段プロセッサに出力されていったレコードがメモリ中に占めていた領域が、次のマージ用の write buffer を提供し、プロセッサのマージ動作中、常にこの buffer 領域は確保されることになる。

一般にＸ＜Ｌの場合の制御はＸ＝Ｌの制御とほとんど同一の制御で十分である。　一方、上記の拡張制御を行なうならば、（キー長）＜Ｌである限り、Ｘ＞Ｌの場合もソートが可能となり、レコード長の変動に対してもパイプラインを乱すことなくソートを実行することが可能となる。

6．4　Length　Tuning

Ｘ≠Ｌの場合、一般にプロセッサのメモリには空き領域が生じる。　以下この空き領域の有効利用について考える。

ｎ台のＫ－way プロセッサを結合したソータのパワー（ソート出来るレコードの最大個数）はK^nである。　従って、

X＜Lの場合は空き領域にレコードを詰め込むことはパワーを越えたレコードのソートを行なうことを意味し、プロセッサの台数を増さない限り不可能である。 一方、X＞Lの場合にはソータのパワーに余力があり、空き領域の利用はソート出来るレコード数の増加を意味する。 length tuningとはメモリの利用効率を高めることによってより多くのレコードをソートする為の制御機構である。

X＞Lの場合の基本的な制御方式は、K－1個のレコードが収容出来るプロセッサP_i まで$i-1$個のプロセッサをスキップし、P_i を改めて初段のプロセッサと考えてソートを行なう手法である。 この場合、結果的にX＜Lの場合に帰着したことになり、メモリの使用効率ηはX／K^{i-1}Lとなる。一方、少なくとも1個のレコードがメモリにロードできればプロセッサは2－way でマージ可能であり、一般にway 数を落とすことによって当該段を有効に利用することが出来る。

さてX＝Lの場合、i 番目のプロセッサP_i は長さ$K^{i-1}L$（バイト）のストリングをK本マージして長さ$K^i L$のストリングを次段のプロセッサP_{i+1}に送出する。 そこで$M_i = K^i L$の大きさの記憶をP_i の論理空間と呼び、一方、P_iが実際に持つメモリ容量$M_i' = K^{i-1}\cdot(K-1)L$の記憶を$P_i$ の物理空間と呼ぶことにする。 ここでX＞Lの時、P_{i+1}の論理空間に出来るだけレコードを詰め込む、即ちP_{i+1}の論理空間の使用効率η_{i+1}を

$$\eta_{i+1} = [M_{i+1}/X]\ X/M_{i+1}$$

に保つことをi 次のlength tuningと呼ぶ。 i 次のtuningによりP_{i+1}以降のプロセッサの論理空間の使用効率は全てη_{i+1}に保たれ、ソータ全体の論理空間の使用効率はほぼこの値に等しくなる。 η_i の平均値はK＝8、i ＝2でも98％程度が得られ［4］、比較的低次のtuningでも大きな効果が得られる。

i 次のtuningに於てM_{i+1}にレコードを詰め込む操作は、P_iがP_{i+1}にK本のストリングを送出することによって行なわれる。 この際、最初のK－1本のストリングはP_{i+1}の物理空間M_{i+1}'に格納されねばならない。 ここで次の事実が成立する。

「任意の正実数α、任意の自然数Kに対し、

$$[K\alpha] = l_1[\alpha] + l_2([\alpha]+1)$$
$$(l_1+l_2=K,\ l_1,\ l_2 \geqq 0)$$

なる整数l_1、l_2が定まる。」

これにより、$M_{i+1}' = K^i\cdot(K-1)L$に対し、

$$[M_{i+1}'/X]$$
$$= [K^i\cdot(K-1)L/X]$$
$$= l_1[K^i L/X] + l_2([K^i L/X]+1)$$
$$= l_1[M_i/X] + l_2([M_i/X]+1)$$
$$(l_1+l_2=K-1,\ l_1,\ l_2 \geqq 0)$$

なるl_1、l_2が定まる。 これはP_i の論理空間がtuningされており、かつP_i がtuning時に詰め込むことの出来るレコード数よりも1多い（$[K^i L/X]+1$）レコードを送出することが可能ならば、$[M_i/X]$の長さのストリングをl_1回、$[M_i/X]+1$のものをl_2回、計K－1回送出することによってP_{i+1}の物理空間M_{i+1}'のtuningが可能であることを示している。 P_i は論理空間を越えた大きさのレコードを処理することが必要となり、補助メモリを付加せねばならないことは明らかである。

次にP_{i+1}の論理空間M_{i+1}をtuningするには、上記のM_{i+1}'のtuningを行なった上で、M_{i+1}'に残された未使用領域$K^i\cdot(K-1)L - [K^i\cdot(K-1)L/X]X$と$P_i$から送られてくるK本目のストリングの持つ空間$K^i L$を会わせた空間をtuningすることによって実現される。 即ち、K本目のストリング長は

$$[(K^i\cdot(K-1)L - [K^i\cdot(K-1)L/X]X + K^i L)/X]$$
$$= [K^{i+1}L/X] - [K^i\cdot(K-1)L/X]$$

となる。 一般に任意の正実数α、βに対して

$$[\alpha+\beta] = [\alpha] + [\beta] + \varepsilon$$
$$(\varepsilon = 0 \text{ or } 1)$$

であるから、

$$\text{上式} = [K^i L/X] + \varepsilon \qquad (\varepsilon = 1 \text{or} 0)$$

を得る。 即ち、K本目のストリングに対しても要求されるストリング長は高々tuning時の送出ストリングよりも1レコード長いだけであることが示された。 従ってP_{i+1}の論理空間のtuningは次式に示されるストリング送出アルゴリズムによって実現される。

$$[M_{i+1}/X]$$
$$= [M_{i+1}'/X] + [M_{i+1}/X] - [M_{i+1}'/X]$$
（K－1本）　（K本目）
$$= l_1[M_i/X] + l_2([M_i/X]+1)$$
$$+ [M_i/X] + \varepsilon$$
$$(l_1+l_2=K-1,\ l_1,\ l_2 \geqq 0,\ \varepsilon = 1 \text{ or } 0)$$

即ち、前段のプロセッサがP_{i+1}の物理空間M_{i+1}'をK－1本のストリングの送出によってtuningした後、K本目のストリングとして$[M_i/X]+\varepsilon$を送出すればP_{i+1}の論理空間のtuningが実現される。 ここで上式を帰納的に展開すれば、

$$[M_{i+1}/X]$$
$$= l_1[M_i/X] + l_2([M_i/X]+1)$$
$$+ [M_i/X] + \varepsilon$$

$$= l_1 (l_1' [M_{i-1}/X] + l_2' ([M_{i-1}/X] + 1) + [M_{i-1}/X] + \varepsilon'') + l_2 ((l_1'' - 1) \cdot [M_{i-1}/X] + (l_2 + 1) \cdot ([M_{i-1}/X] + 1) + [M_{i-1}/X] + \varepsilon'') + (l_1'' - \varepsilon) \cdot [M_{i-1}/X] + (l_2' + \varepsilon) \cdot ([M_{i-1}/X] + 1)$$

$$= \cdots\cdots$$

となり、結局

$$[M_{i+1}/X] = L_1[M_1/X] + L_2([M_1/X] + 1)$$

と書け、初段のプロセッサP_1のみがK以下で動的にマージウェイを変化させることによって任意次数のtuningを行なうことが出来る。 ここでP_1のway数の変化はP_1への入力ストリング長が1（record）から0へ変化したことと等価であり、この場合初段のプロセッサも含めて全てのプロセッサがK−wayで動作し、ストリング長のみが動的に変化するとみなすことも可能である。

又、tuningに参加しているプロセッサに付加するメモリは各々のプロセッサにつき高々1レコード分あればよいことは明らかであるが、実際にはこれ以下の量で十分である。 例えば、1次のtuningを行なった場合、初段のプロセッサP_1に要求される論理空間の最大値は、

$$MAX\lceil [K^2 L/X]/K \rceil X = 2K^2 L/(K+1) \quad (X = K^2 L/(K+1))$$

となる。 この時のP_1のマージwayは2であり、一方のストリングのみP_1の物理空間に格納できればよく、付加すべきメモリは

$$K^2 L/(K+1) - M_1' = L/(K+1)$$

となり、僅かな量で済むことがわかる。

6．5 ストリーム長変動に対する制御

6.5.1 ストリームエンド検出機構

ソータのパワーをM（records），入力ストリーム長をY（records）とする。 Y<Mの場合、入力ストリームにダミーレコードを付加し、Y=Mとして制御を行なう手法が最も単純な解決法であるが、この場合、$i > \lceil \log_K Y \rceil$なるPiは実質的に機能しておらず、意味のない操作を行っていることになる。 これらのPiで、入力ストリームを単に素通しさせることが可能であればソート時間は

$$Y + M + \lceil \log_K M \rceil - 1$$

から

$$Y + K^{\lceil \log_K Y \rceil} + \lceil \log_K M \rceil - 1$$

に改善される。 この機能は、eos（end of stream）及びeor（end of record）の2つのフラグを設けることによって実現出来る。 eorはストリームの最後のレコードに、一方eosは入力ストリームの最後のストリングの先頭レコードに付加される。 これらフラグの領域はレコードのポインタ部2バイトを用いることが出来る。 eosとeorはマージの度に上記の条件を満たすようにbubble upされる。 この時、$i > \lceil \log_K Y \rceil$なるPiに於ては入力ストリームはソートが完了し、1本のストリングとなっている。 この時、eosが先頭レコードに、eorが最終レコードに付加されていることになり、プロセッサはこの間のレコードを素通しすればよいことがわかる（図6．2）。

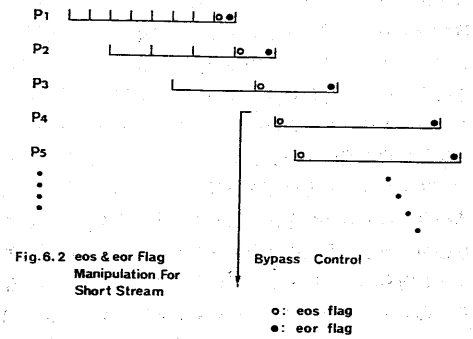

Fig.6.2 eos & eor Flag Manipulation For Short Stream

6.5.2 マルチソータ

Y>Mの場合には、入力ストリームがソータ1台のパワーを越えている為、ソータを複数台制御する工夫が必要である。この場合の基本的な解決法は「Y／M」台のソータを用意し、各ソータに入力ストリームを分散し、ソートを行わせた後に各々の出力ストリームを1本にマージすることである。 各ソータには簡単なマージユニットと、レコードの送出を要求するdemand lineを付加すればよい（図6．3）。 この方法では、ソートの実行につれて各ソータに各々未使用領域が発生して行くことになる。 従って、次のストリームのソ

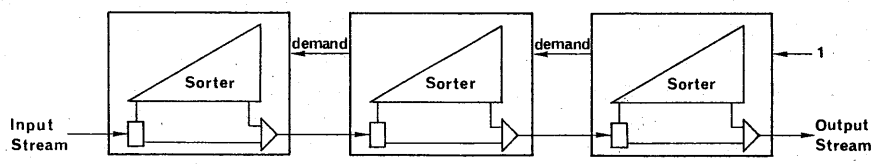

Fig.6.3 Linear-Combined Configuration Of Multiple Sorters

ートを行う場合、各ソータで生成された空領域を利用しつつ、処理を進めて行く必要がある。 レコードの出力があれば必ずいずれかのソータに1レコードの空が生じているはずであり、そのソータに入力レコードを割り付けられるのである。一方、空領域を先頭のソータから順次生成することも可能であり、ここでは次のような手法を採った。 ソータの各プロセッサのメモリにK－1本のストリングをロードした後、先頭のプロセッサを駆動し、以降のプロセッサを前段のプロセッサからの入力ストリームで駆動する手法である。 これは、各プロセッサはK－1本のストリングのロードが出来ればK本目のストリングをその長さによらずマージ実行出来ることに着目すれば明らかである。 この手法では demand line は不要となり、又、未使用領域は先頭のソータから順次生成される為、後述のマルチストリーム機能により、複数本の長い入力ストリームを効率良く処理することも可能である。

6．6 マルチストリーム

ソータを連続して使用する時、N個のレコードから成るストリームのソートを繰り返し行なう場合は問題ない。 しかし一般的には種々の長さのストリームを連続して処理することが考えられる。 又、ストリームとストリームとの間には時間的なギャップが存在することもあり得る。 N個のレコードから成るストリームを扱う場合には、ソータ内には高々2本のストリームしか存在しないが、一般的には2本以上のストリームが存在することになる。 これら複数本のストリームを効率良く処理することがマルチストリーム機能である。

マルチストリームを処理する場合、一台のプロセッサに複数本のストリームが混在することが一般的である。 従って、これらのストリームに対する記憶管理を実現する必要がある。

ソート済みのストリームが多く入力されると、その各々を識別するポインタを設けることは非現実的であり、何らかの工夫が必要となる。 ここではこれらストリームを区別せず1本のストリームとして扱うこととした。 又、これらストリームが時間をおいて不連続的に入力された場合には、1レコード出力1レコード入力の対応がとれない、即ちレコード出力に対して入力レコードが存在しない期間が生じ、この間に生成されたメモリ内の未使用領域の管理も必要となる。又、個々のプロセッサは、レコードの入力と出力が同期していることを前提に動作している一方、入力ストリームがこのようにレコード長の単位で同期していることは一般に期待出来ず、従来の状態遷移だけでは不充分となる。この為の一つの解決法として、プロセッサの動作を入力レコードの処理及びメモリ（未使用領域、フリーリスト）の管理を行なう部分（ m-part ）と比較処理を行なう部分（ c-part ）に分けて制御する方法が考えられる。 m-part はメモリ内のフリーリストから1レコード領域を得て入力レコードを格納し、必要なリンクを施す。 一方、 c-part は比較に必要なレコードをメモリから読み出し、空いた領域をフリーリストに返す。 2つの部分は独立に状態遷移する。 このような手法により複数本のストリームを制御することが出来る。

又、1つのプロセッサ内に複数本のストリームが混在することは、既に各ストリームのソートが完了していることを意味し、これらのストリームを送出する際、1本のストリームに連接してしまうことが許される。 即ち、短いストリームはソート終了と同時に前述の素通し機能によってソータ内を加速し、前方のストリームに次々に追いついて、出力時には1本のストリームとすることが出来る。 この機能は、不連続的に発生するストリームを順次処理する場合有効と考えられる。 例えばデータベースマシンGRACEの環境では、1台のソータ（プロセッシングモジュール）にバケット（入力ストリーム）が不連続的に入力されるが、上記の作用によってこれら不連続的なバケット入力に対しても連続的な出力が可能となり、マルチストリーム機能の効果が発揮される。

§7． おわりに

パイプライン化されたマージソートを行なうプロセッサのレジスタトランスファーレベルに於ける設計、記述を行ない、シミュレーションによってその制御の正当性を確認した。更に2-wayマージプロセッサを実際に試作し、動作を確認した。 レコードの1バイト処理は、3Mbyte／sec 程度のスピードが得られている。 又、本ソータは実際の使用環境に於て種々のパラメータの変動に対して柔軟で効率の良い処理が可能である事を示し、その実現の為の制御方式について検討した。 これらの拡張制御方式はプロセッサのマイクロプログラムを書き換える事によって比較的容易に試作システムに実装する事が出来ると考えられる。 現在、本ソータのＬＳＩ化について検討中である。

<< 参考文献 >>

［1］喜連川，鈴木，田中，元岡
「可変構造多重処理データベースマシンのシステム構成」
情報処理学会第22回全国大会 3L-3 1981

［2］伏見，喜連川，田中，元岡
「GRACEに於けるソーティングユニットの機能拡張」
情報処理学会第24回全国大会 4G-4 1982

［3］桑原，喜連川，伏見，田中，元岡
「GRACEに於けるソーティングユニットの構成」
情報処理学会第24回全国大会 4G-5 1982

［4］喜連川，鈴木，田中，元岡
「可変構造多重処理データベースマシンに於けるソートモジール」
信学技法 EC 81-15, 1981

［5］喜連川，鈴木，田中，元岡
「Hash とSort による関係代数マシン」
信学技法 EC 81-35, 1981

盛光印刷所