論　文

UDC 681.325.67.012

# パイプラインマージソータの構成

正員　喜連川　優[†]　　准員　伏見　信也[†]　　非会員　桑原　和宏[†*]
正員　田中　英彦[††]　　正員　元岡　達[††]

# An Organization of Pipeline Merge Sorter

Masaru KITSUREGAWA[†], *Regular Member*, Shinya FUSHIMI[†], *Associate Member*, Kazuhiro KUWABARA[†*], *Nonmember*, Hidehiko TANAKA[††] *and* Tohru MOTO-OKA[††], *Regular Members*

あらまし　ソーティングは計算機システムにおいて多用される基本的な操作であり，近年のハードウェア技術の進歩を反映しソートを高速に処理するハードウェアへの関心が高まりつつある．ソートの利用は多岐にわたるが，データベース処理等では二次記憶とホストとのデータの受け渡しが頻繁に生じており，その転送時間を利用し，データ流に追従した$O(N)$処理時間のソータを構成することによってシステム性能は大きく向上するものと考えられる．本論文では$O(N)$ソートアルゴリズムとしてパイプラインマージソートを取り上げ，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検討を行なうと共に，そのハードウェア化について考察する．又，マイクロプログラム制御方式によるプロセッサを試作し，動作を確認したので，その構成及び性能について報告する．

## 1. まえがき

ソーティングは計算機システムにおいて頻繁に利用される基本的な操作である．その利用は多岐にわたるが，データベース等大容量のデータを対象とする場合には一般にソート開始時点で主記憶上にデータが揃っていることは期待できず，二次記憶上のデータを一旦主記憶に転送する必要がある．この時間は大きなオーバヘッドになると予想され，転送時間を利用してソートすることが考えられる．これまでに報告されているデータ流に沿った$O(N)$ソートには，プロセッサを$N$台用いるものとして，(1)パイプラインバブルソート[(1)]，(2)リバウンドソート[(2)]，(3)並列計数ソート[(3)]等が，又$\log N$台用いるものとして，(4)パイプラインマージソート[(4)]，(5)パイプラインヒープソート[(5)]がある．$N$台のプロセッサを用いるソータでは，大量のデータを取り扱う際データ数分のプロセッサが必要となり，そのコストは無視できない．レコード長，レコード数等に対する柔軟な制御を実現するには各プロセッサのロジックは一層増加すると考えられ，$\log N$台構成のソータの方が有利である．又，(4)はソータの容量を単位としたパイプライン処理が可能であり(5)に比べてソータの連続的使用に適している．(4)は，その実装，ハードウェア化に関して充分な検討がなされていない．

本稿では，パイプラインマージソートを取り上げ，外部記憶からデータが連続的にとり出される環境下でその流れに沿ってソートを実行するハードウェアソータの構成法を提案する．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検討を行うと共に，そのハードウェア化について考察する．更に，マイクロプログラム制御によるプロセッサを試作，動作を確認し，3 MByte／sec の処理速度を得たので，その構成及び性能について述べる．

我々はデータ流に沿った処理を実現する高速関係代数マシンGRACE[(7)]の開発を行っている．結合や射影等，処理負荷の重い関係代数演算も関係表のソートにより$O(N)$で実現可能である．本ソータはGRACEにおける基本要素として開発したものである[(6),(8)]．

## 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

† 東京大学大学院工学系研究科，東京都
Graduate School of Engineering, The University of Tokyo, Tokyo, 113 Japan
†† 東京大学工学部電気工学科，東京都
Faculty of Engineering, The University of Tokyo, Tokyo, 113 Japan
* 現在，東京大学生産技術研究所
論文番号：昭58－論146［D－40］

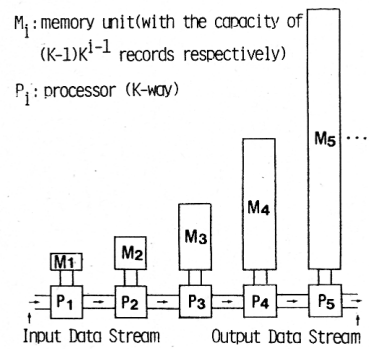


図1　パイプラインマージソータの構成
Fig.1－Global architecture of pipeline merge sorter.

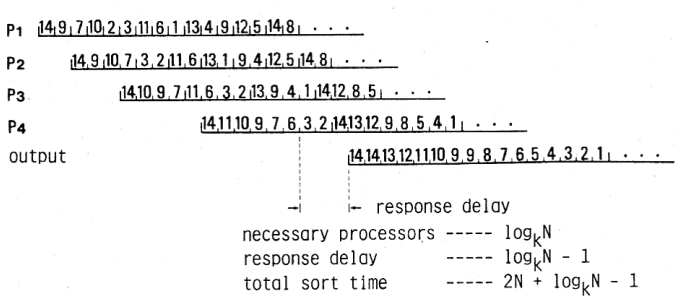


図2　パイプラインマージソータにおける処理の様子
Fig.2－Sorting process overview of pipeline merge sorter.

ソートを基調としている．今，$N(=K^n)$ 個のレコードのソートを行うものとすると，$K$-way merge を行うプロセッサを $n(=\log_K N)$ 台用意し，1次元状に結合する（図1）．第 $i$ 番のプロセッサは，$K^{i-1}\times(K-1)$ レコード分のメモリを持つ．$N$個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 $K^{i-1}$ 個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 $K^i$ レコードからな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．図2に $K=2$ の場合の処理の様子を示す．図から明らかなように，各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，$K$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できる．図2ではパイプラインのセグメントはレコードレベルとしてあるが，これはバイトレベル，ビットレベルまで落とすことができる．

### 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，

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

である．この内，$2N$の項は全レコードを一旦ソータに入れ，その後再び取り出すための時間であり，実質的なオーバーヘッド時間は $\log_K N-1$ となり極めて短い．

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，$K$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．即ち各プロセッサは以下の Phase を状態遷移する．

Phase 0：　$K-1$ 本のストリングをメモリ内にロードする．ロード終了後，Phase 1 へ．

Phase 1：　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後 Phase 2 へ．

Phase 2：　メモリ内に残存する $K$本のストリングのマージ操作を行いつつ，次の $K-1$ 本のストリングの入力を行う．$K-1$ 本のストリングのロードが終了すると（同時に残存するストリングのマージも終了し）Phase 1 へ．

### 2.4　メモリの管理

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い，1レコード出力していくため有効なメモリ容量は常に一定である．一方，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ゆ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．この問題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[6]．

## 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う必要がある．ここではマージ数を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の後にポインタ部を付加したフォーマットでメモリに格納される．本ソ

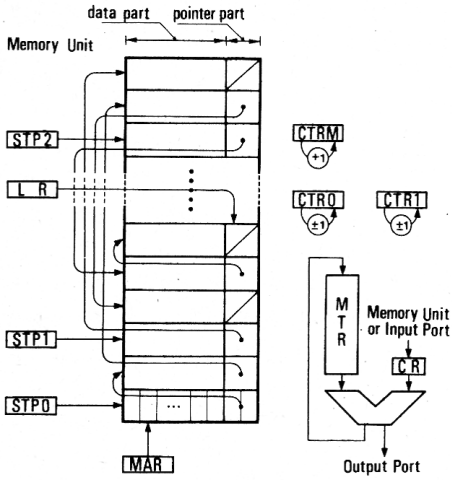


図3 プロセッサの構成
Fig.3-Configuration of a processor.

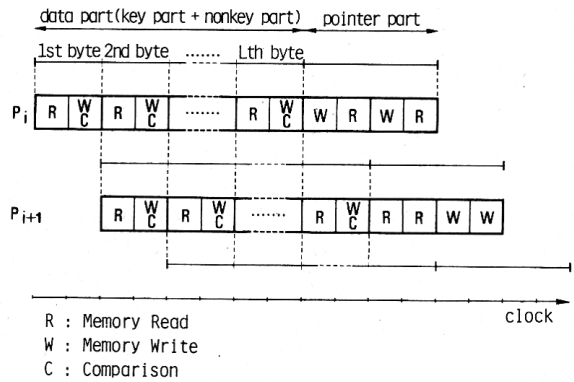


図4 パイプライン処理の様子
Fig.4-Pipeline processing overview.

ータにおけるプロセッサとメモリからなる各段の構成の概略は図3に示される．以下説明を加える．

### 3.2 バイトレベルパイプライン

パイプラインの1セグメントを1バイトの処理とするバイトレベルのパイプラインを採用した．一般に1バイトの比較操作を行う程度のハードウェアは高速であり，各プロセッサにおける1バイトの処理は，比較操作よりもむしろ操作対象データの取り込みに伴うメモリアクセスが支配的となる．従って1バイトの処理に要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如く，一般にPhase 1，2の処理ではマージすべき2本のストリングがメモリ中に存在し，この状態でのメモリアクセス回数が最も多い．即ち二つの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供給するために2回，更に入力レコードをメモリに書き込むために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．しかし，マージ操作ではいずれか一方のストリングの先頭レコードは出力されるが，他方のそれは再び次のマージに利用されるため，このレコードをプロセッサ内レジスタに保持しておくことにより，メモリアクセスを1回減らすことが可能である．ここでは，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，入力ストリングのメモリへの書き込みとマージ用ストリングの読み出しの2回のメモリアクセスによる1バイト／2クロックの処理方式を採用した．具体的には初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い，write clock でメモリ書き込みと比較を並行して行うこととした（図4）．尚，ポインタ部の操作は種々の場合が存在し，この限りではない．

### 3.3 比較器

以下，本稿を通じて降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からなる．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内のレコード
10： CRに入力されたレコード＜MTR内のレコード

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： string 0→CR，string 1→MTR
1： string 0→MTR，string 1→CR
とすれば，

$s01 \leftarrow \overline{s01 \oplus rf[0]}$

によってs01は矛盾なく更新される（rf〔0〕はrfの下位ビット）．

### 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として管理するためには，リストの最後尾へのレコードの追加，及び先頭からのレコードの削除の機能が必要である．このため，実装に当って以下の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： 直前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バイトを指し，リンクの生成に用いる．

◇ RLC： レコード長管理用レジスタ．

◇ 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長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う必要がある．ポインタ操作の一例を図5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（両者の間に値を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5(a)～(d)のように処理が進む．図5(d)に示される如く4クロック経過した時点で，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われたことがわかる．尚，レコード部とポインタ部の操作の切替えはRLCを用いて行われる．

### 3.5 状態遷移

2.3で述べた3つのPhaseの遷移は各ストリングの入力完了と共に生じる．この状態遷移のストリング長管理のためにCTRMなるカウンタを用意した．又，マージ操作では一方のストリングが尽きるとその後は他方のストリングを単に出力することになる．この制御のため，当該ストリングの出力レコード数を保持するカウンタCTR $i$（$i$＝0，1）を設けた．

ストリング単位の状態遷移図における3つのPhaseは，レコード単位の遷移に展開すると更に細く9つのstateに分けられ，プロセッサはレコード入力と共に図6に示すような状態遷移を繰り返す．以下，各stateについて概略を示す．

〈Phase 0〉 最初のストリングに対してstate 0で先頭のレコードをMTRに，state 1 でそれ以後のレコードをメモリに格納する．

〈Phase 1〉 本Phaseではメモリ内のstring 0と入力されるstring 1 のマージを行う．state 2においてstring 1がstring 0よりも大きい間string 1のレコードは入力ポートから直接比較器に取り込まれる．string 0 のレコードがstring 1 のそれより初めて大きくなった時点で入力レコードはメモリ内に格納されることになり，state 3ではこのレコードに対し閉ループが生成されるようにポインタを設定する．state 4では2つのストリングのレコードをメモ

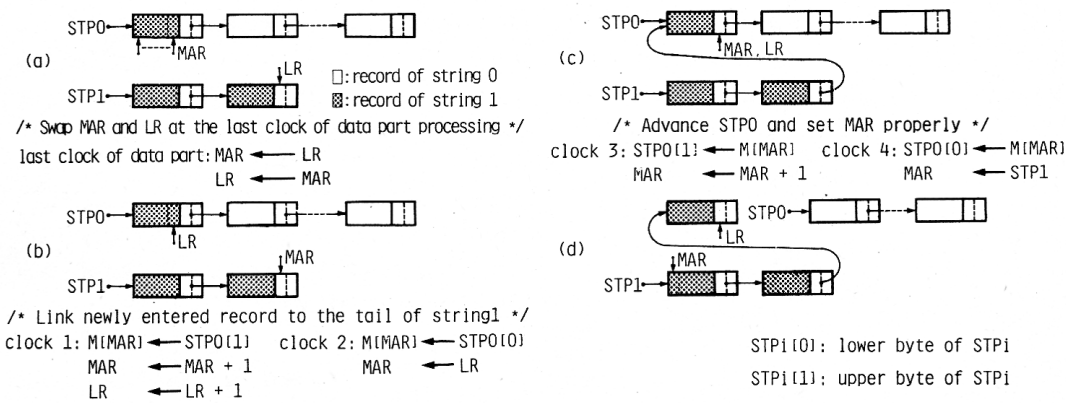


図5 ポインタ操作の例
Fig.5-An example of pointer manipulation.

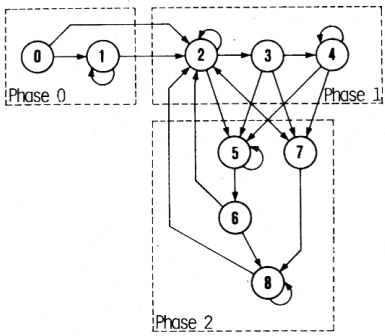


図6 状態遷移図
Fig.6-State transition diagram.

リから取り込みマージすると共に入力ストリングをメモリに格納する．

〈Phase 2〉 本Phaseではメモリ中の2本のストリングstring 0，1のマージを行うと共に次ストリングstring 2 のメモリ内へのロードを行う．state 5ではいずれか一方のストリングが全て出力されるまでマージを行い，比較操作が不要となった時点でstate 6に遷移する．state 6ではメモリ内のstring 2 の先頭レコードをMTRにロードし，本Phase終了後遷移するPhase 1でのstring 0，1のマージに対する準備を行う．state 8では残存するストリングをメモリから直接出力する．出力完了時点でPhase 1へ戻る．state 7は一方のストリングが他方のそれよりも完全に大きい特殊な場合に用いられる．

以上のように状態遷移にはレコードの入力系列，レコード数に依存して種々の場合が存在し，又この処理をポインタ操作部の2バイトタイム4クロック時間に集約しているため，比較的複雑な遷移制御が必要となっている．詳細な説明については(8)を参照されたい．以上の状態遷移はレジスタトランスファレベルの記述を行い，更にPascal を用いてシミュレーションプログラムを作成することによってその正当性を検証した．

### 3.6 ハードウェアリソース

各プロセッサが持つレジスタ，カウンタ，フラグについて簡単にまとめておく．

◇ MAR (Memory Address Register)
メモリアドレッシングに用いる．アクセス単位は1バイト． 〔16ビット〕

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

◇ STP $i$ ($i$-th String Top Pointer)
メモリ内におけるstring $i$ の先頭レコードのアドレスを保持する．($i=0, 1, 2$) 〔16ビット〕

◇ CTRM (Master Counter)
Phase 間遷移のために入力ストリングのレコード数をカウントする．

◇ CTR $i$ (Counter 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．($i=0, 1$)

◇ 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〕
比較器へのもう一方の入力データを保持する．

◇ RLC (Record Length Counter)
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〕
出力レコードのストリング番号を保持する．

## 4. 試作プロセッサの構成

### 4.1 プロセッサの入出力

図1に示されるソータのプロセッサを1台試作した．その入出力は図7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なる．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る．

### 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, STP $i$, CTRM, CTR $i$, MTR, CR, RLC, rf, s01が必要であ

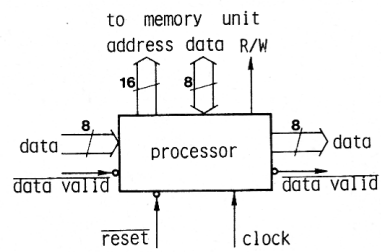


図7 プロセッサの入出力
Fig.7-Input and output lines of a processor.

った．比較器の一方の入力としてMTRをシフトレジスタとして考えていたが，ここではRAMとカウンタによる実装を採用することとし，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ことにした．又，各種パラメータを初期化するため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(Initialize 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（入力データ一時保持レジスタ，1バイト），A／D（昇順降順ソートを指定するためのフラグ，1ビット），R／W（read cycle，write cycleを示すフラグ，1ビット）が加わっている．プロセッサの内部構成を図8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数は僅かでよく，そのロジックの占めるオーバヘッドは大きく減少できる．従って逆に各々のプロセッサに種々の機能を加え，より柔軟なソータを構成することが可能と考えられる．我々は既に種々の機能拡張について検討を行ってきた[8]．従って，試作に当ってもこれらの成果を受け入れ易い柔軟な構成とするため，制御方式としてマイクロプログラム方式を採用することとした．

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．将来高い集積度が得られた場合，本ソータでは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，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，水平マイクロプログラム方式による制御を行うことにした．1バイトのデータの処理は2クロック（2回のメモリアクセス）で構成されるため，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ことも可能であるが，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ため，時間的な制約から命令内に次アドレスのフィールドを設けることにした．又，図6に示される状態遷移からもわかるように多重分岐を行うことが多い．これに関しては分岐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方式を採用した（図9）．ここでは下位の数ビットを修飾しており，マイクロプログラム上で多重分岐に伴うアドレス境界を調整する必要がある．

又，マイクロ命令のフェッチ，実行のパイプライン化のため，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らない．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ことがある．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

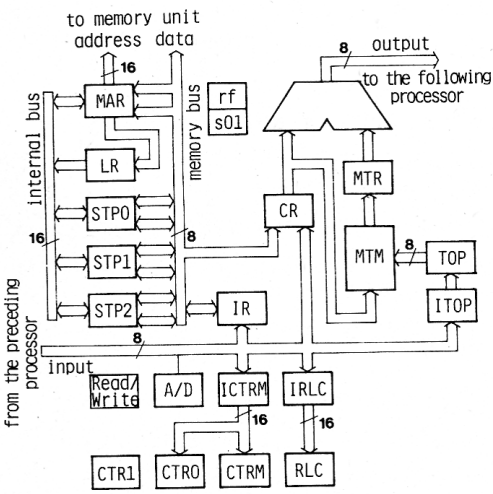


図8 プロセッサの内部構成
Fig.8- Internal structure of a processor.

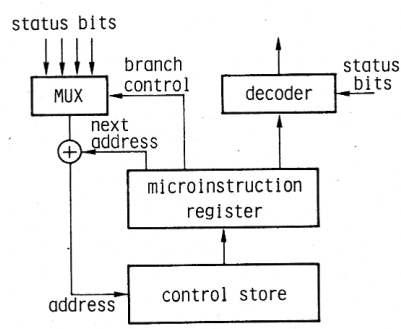


図9 マイクロ命令の順序制御
Fig.9-Sequence control of microinstruction.

条件は実行の直前に確定していればよいことになる．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，制御記憶容量が減少した．

### 4.5 プロセッサ間結合とタイミング

プロセッサは，8 bit のデータ線と $\overline{\text{data valid}}$ 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同時に，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バイトタイムに行われる．図10に処理のタイミングを示す．

制御線 $\overline{\text{data valid}}$ 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できる．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できる．$\overline{\text{data valid}}$ は2クロック遅れて次段へ伝搬される．

### 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立って各定数を入力データストリームとして与えることによってなされる．各プロセッサは送られてきたデータを当該レジスタに格納した後，CR を介して次段のプロセッサへ送り出す．ここで，ストリング長は他の定数と異なりプロセッサの位置する段数によって変化し，一段毎に2倍して次段のプロセッサへ送出される．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，各々その初期化が完了した時点で直ちにソートを開始できる．

## 5. 試作システム

### 5.1 全体構成

プロセッサ1台だけでは動作確認を行うことはできず，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ため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ュータ 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．

### 5.2 ハードウェア実装

試作においてはLSI化への見通しを良くするため，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11）．基板毎にサービスプロセッサとのインターフェイス回

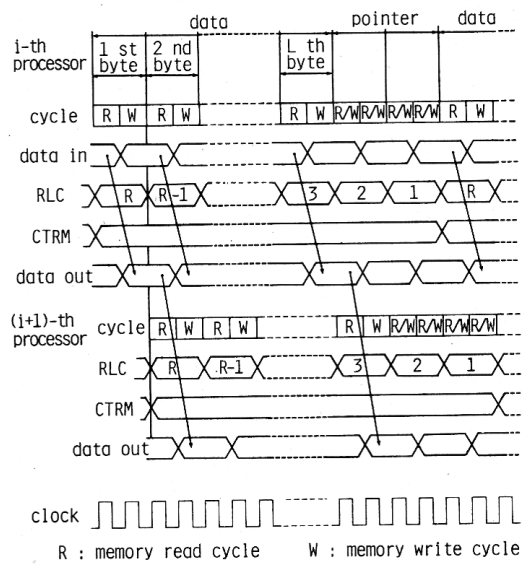


図10 タイミングチャート
Fig.10-Timing chart.

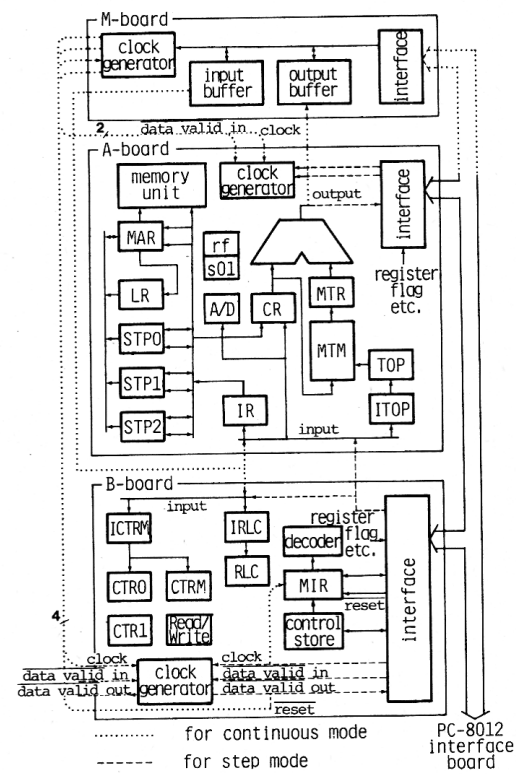


図11 試作システムのハードウェア構成
Fig.11-Hardware organizaton of the pilot system.

路が設けてある．更に，1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

ソータを駆動するに当り，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．本ソータは**3.4**で述べた如く比較的複雑な処理を伴うため，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グ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以下の2つのモードを設けた．

(1) ステップモード　本モードでは入力データ及び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．図12にソートのトレースを行っている際の画面の一例を示す．

(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．

### 5.4 処理速度

本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150 nsecのCMOSメモリを使用しており，6 MHzクロックでの動作を確認した．即ち，ほぼ3 MByte／secのデータ流に追随することができる．現在のディスクの転送レートは0.5～3 MByte／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．

```
 MAR=0003          CR=02          CYCLE=WRITE   1
  LR=0002         MTR=02            RLC=0001
STP0=0001        CTR0=0000         CTRM=0001
STP1=0000        CTR1=0000          TOP=01
STP2=0000          IN=05           MIR=20 15 00
 s01=   1         OUT=02                CC 01 C0
  rf=  01       STATE=1          N.Add=42
 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        CTR0=0002         CTRM=0002
STP1=0000        CTR1=0000          TOP=04
STP2=0000          IN=05           MIR=30 08 00
 s01=   1         OUT=02                01 00 03
  rf=  01       STATE=2          N.Add=16
 MEMORY = 05 01 00 01 FF FF FF 00       0
```

図12　ステップモードにおけるトレース情報の表示
Fig.12－An example of traced information in step mode.

## 6. むすび

パイプライン化されたマージソートを行うプロセッサのレジスタトランスファレベルにおける設計，記述を行い，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，動作を確認すると共に，現在のディスクの最高転送レートに匹敵する3 MByte／sec程度の処理速度を得た．今回の試作では基本的な機能の実装にとどめたが，実際の使用環境においてはレコード長，レコード数，キー長等種々のパラメータの変動に対して柔軟且つ効率の良い処理が必要である．このための制御方式についても検討しており[6]本ソータの拡張と共に稿を改めて論じたい．これらの拡張制御方式はマイクロプログラムの変更により比較的容易に実装可能と考えられる．現在，電電公社武蔵野電気通信研究所の御協力により本ソータのLSI化を進めている．

### 文　献


(1) Kung, H. T. : "The Structure of Parallel Algorithms", Advances in Computers, 19, Academic Press (1980).

(2) Chen, T. C., Lum, V. Y. and Tung, C. : "The Rebound Sorter : An Efficient Sort Engine for Large Files", Proc. 4th VLDB, pp.312-318 (1978).

(3) 安浦，高木："並列計数法による高速ソーティング回路"，信学論 (D), J65-D, 2, pp.179-186 (昭57-02).

(4) Todd, S. : "Algorithm and Hardware for a Merge Sort Using Multiple Processors", IBM J. R & D, 22, 5, pp.509-517 (1978).

(5) Tanaka, Y., Nozaka, Y. and Masuyama, A. : "Pipeline Searching and Sorting Modules as Components of a Data Flow Database Computer", IFIP 80, pp.427-432 (1980).

(6) 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おけるソートモジュール"，信学技報，EC81-15 (1981-06).

(7) 喜連川，鈴木，田中，元岡："Hashとソートによる関係代数マシン"，信学技報，EC81-35 (1981-10).

(8) 喜連川，伏見，桑原，田中，元岡："パイプラインマージソータの構成"，信学技報，EC82-32 (1982-06).


（昭和57年10月6日受付，11月8日再受付）