マイクロコンピュータ 12－1
（1980. 6. 30）

仮想レコード方式テキストエディタとそのデータエントリ端末への応用について

重松保弘，野上睦夫，安在弘幸
（九州工業大学・情報工学科）

1. まえがき

比較的大容量の外部記憶装置（フロッピーディスクなど）を備えた、安価で処理能力の高いマイクロコンピュータシステムを知能端末（インテリジェントターミナル）としてホスト計算機と結合することにより、経済的で効率のよい分散処理システムの実現が期待される。このような分散処理の1つの形態である出力検索システムについては、すでに報告した[2]。本稿では、マイクロコンピュータをデータエントリ端末として実現した遠隔ジョブ入力（RJE）システムについて述べる。

フロッピーディスクベースのデータエントリ端末としては、キーフロッピー（key-to-floppy）装置が一般的である[1]。キーフロッピーシステムは、従来のパンチカードシステムにおけるデータの記録媒体をカードからディスケットに置換えたものであり、データの蓄積、検査やプログラムファイルの一時退避には便利であるが、ファイルの内容の編集には適していない。すなわち、キーフロッピー装置のディスケットには通常、データが順編成ファイルの形式で格納されているため、ファイルの構成単位であるレコードの削除は容易であるが、挿入は容易でない。また、削除レコードが増加するにしたがって、ディスケット中の未使用領域が増加し、ディスケットの使用効率が低下する。

本研究では、こうしたキーディスケット装置のデータ編集処理上の問題点を解決するため、高度の編集能力をもつデータエントリ端末をマイクロコンピュータを用いて開発した。特に、本システムでは、レコードを記憶単位とする仮想記憶方式（仮想レコード方式と呼ぶことにする）を用いることによって、利用者に媒体を意識させない効率的なテキスト編集システムを実現した。

2. RJE システムの構成

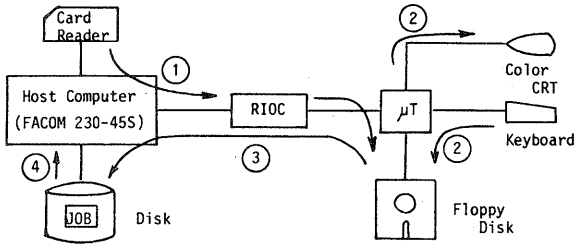
本研究で実現した RJE システムの構成を図1に示す。なお、マイクロコンピュータを用いて開発したデータエントリ端末をμTと略記する。また、μTにおけるデータの処理単位（カード1枚分のデータ）をテキストと呼び、テキストの集合（通常、ジョブ制御文、プログラム，データ等を含む1個のジョブプログラム）をテキストファイルと呼ぶことにする。

以下に、本システムにおけるテキストファイルの生成方法、テキストファイルの編集方法、リモートジョブ入力の方法について述べる。

テキストファイルの生成は、次の方法で行なう。第1は、ホスト計算機上のファイルをμTに転送し、ディスケットに格納する方法（媒体変換）である。現在は、ホスト計算機のカード読取装置から入力したプログラムファイルのみを扱っている（図1の①）が、磁気テープや磁気ディスク等のファイルを扱うことも容易である。第2は、μTのキーボードから直接、テキストを入力する方法（図1の②）である。

テキストファイルの編集は、μTのCRT表示装置（CRTと略記）に表示されたテキストに対して、本システムの編集用コマンドを用いて会話型で行なう（図1の②）。なお、本システムはスクリーンエディタの機能をもつので編集が容易である。

また、リモートジョブ入力は次の手順で行われる。μTからホスト計算機へ転送されたテキストファイル（ジョブプログラム）はまず、ホスト計算機のディスク中に一時ファイルとして蓄積され（図1の③）、ついでホスト計算機にジョブとして入力され、実行される（図1の④）。



RIOC：遠隔入出力制御装置
μT：micro-terminal

図1．マイクロコンピュータを端末とするRJEシステムの構成

3．　コマンドと機能

本システムのコマンドの概要を表1.に示す。コマンドには、モニタコマンドとエディタコマンドの2種類がある。モニタコマンドは、システムの初期化、テキストファイルの受信、リモートジョブ入力、およびテキストファイルの生成や削除に関するものである。エディタコマンドは、テキストファイルの内容の編集に関するものである。

3.1　モニタコマンドの機能

いくつかのモニタコマンドについて、簡単に説明する。I（Initialize）コマンドは、後述するテキストテーブル、テキストブロックテーブル、ディレクトリなどのテーブル類を初期化する。したがって、ディスケット中の全テキストファイルが削除される。

テキストファイルを生成するためには、E（Edit）コマンドまたはL（Load）コマンドを用いる。Eコマンドを入力すると、指定したテキストファイルがディスケット中に存在すれば、その先頭1画面分を表示するが、存在しなければ、新たにテキストファイルを生成し、エディタモードになる。ついで、利用者はエディタコマンドを用いてテキストを生成する。また、Lコマンドを入力すると、μTはホスト計算機からデータを受信し、新たにテキストファイルを生成する。データ受信の終了後はエディタモードになるので、Eコマンドと同様、エディタコマンドを用いてテキストを編集することができる。

J（Job）コマンドによって、指定されたテキストファイル（ジョブプログラム）がホスト計算機に転送される*。

C（Change）コマンドを入力すると、フロッピーディスクのロック状態が解除され、ディスケットの交換が可能になる。再度、Cコマンドを入力すると、フロッピーディスクはロック状態となる。

＊　ただし、現在のところホスト計算機に会話モニタがないため、ジョブを受信し実行させるための利用者レベルのプログラムをあらかじめ起動しておかなければならない。

表1. コマンドの概要　　＿：ストローク（打鍵数1）

コマンド種別		コマンド形式	機能
モニタコマンド		I	システムの初期化
		N	テキストファイル名の表示
		D <text-file name> CR	テキストファイルの削除
		R <old text-file name>,<new text-file name> CR	テキストファイル名の変更
		E <text-file name> CR	テキストファイルの編集開始
		L <text-file name> CR	ホスト計算機よりテキストファイルの受信
		J <text-file name> CR	ホスト計算機へのジョブ入力
		C C	ディスケットの交換
エディタコマンド	モード制御	S	スクロールモード（行単位移動）
		P	ページモード（画面単位移動）
	方向制御	[n] ↑	n画面(行)上方向へ移動
		[n] ↓	〃　下方向　〃
		[n] ←	n文字左方向へ移動
		[n] →	〃　右方向　〃
		GO HOME	先頭画面を表示
		CR	最左端を表示
	テキスト編集	[<even number>]␣<text> CR	テキストの挿入
		<odd number>[,<odd number>] CR	テキストの削除
		<odd number>␣<text> CR	テキストの置換
	終了	I	編集の終了

3.2　エディタコマンドの機能

以下に、エディタコマンドについて説明する。画面の移動モードには、画面単位の移動であるページモードと行単位の移動であるスクロールモードの2種類がある。画面の移動は移動させる方向と同じ方向の矢印コマンドを入力することによって行われる。また、矢印コマンドを入力する前にオペランドとして数字（1～99）を入力すると、この数字で示された行数、画面数または桁数だけ画面が矢印の方向へ移動する。CRコマンドでは、現在表示されている行の最左端へ画面が移動し、GOHOMEコマンドではテキストファイルの先頭の1画面分が表示される。

本システムのディスプレイ画面の構成は、図2のようになっている。これを機能別に分類すると、①行番号部、②編集部、③入力部およびシステムメッセージ部、に分けられる。行番号には、1～27の奇数番号が付けられている。また、テキストの挿入、削除、置換は、この行番号を指定することによって行われる。

テキストを挿入するときは、挿入する行の位置を示す偶数番号を指定する（図2参照）。また、行番号を省略すると、直前に挿入したテキストの下に挿入することができる。

テキストを削除するときは、テキストの行番号（奇数）のみを入力する。複数の行を削除するには <行番号>,<行番号> CR と入力する。

テキストを置換するときは、行番号（奇数）を入力し、続けて空白を1個入力する。このとき、カーソルが指定された行の先頭へ移動するので、画面上でテキストの置換を行うことができる。

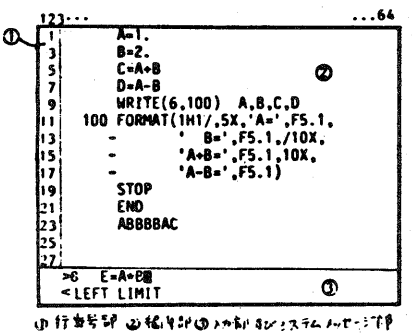


図2. 編集画面の例

編集の終了時には、Tコマンドを入力する。以後、再びモニタコマンドモードに戻る。

4. データ構造と仮想レコード方式

4.1 テキストファイルの仮想的データ構造

テキストファイルは、テキストを要素とする線形リストで表現することができる。線形リストを計算機上にデータ構造として実現する方法は、次の2種類に分けられる。第1は、連続配置法（Sequential Allocation）と呼ばれ、記憶領域上の連続した領域に要素を順次、格納する方法である。第2は、リンク配置法（Linked Allocation）と呼ばれ、ある要素にその要素の先行要素や後続要素の記憶番地（リンクと呼ばれる）を入れておく方法である。前者の表現法は、線形リストの構造が変化しない場合に記憶領域や探索速度の点で効率的であり、後者は、線形リストの構造が変化する場合に、効率的に各要素間の配置関係を変更できる。ただし、リンク配置法は連続配置法に比べて、リンク部の部分だけ余分な記憶領域が必要であり、リンクをたどらなければ目的の要素に到達できないという欠点がある。

本システムのようにプログラムファイルを扱うエディタでは、プログラムの完成に至るまでにテキストの挿入、削除が頻繁に実行されると考えられる。このため、本システムでは、テキストファイルをリンク配置法で実現した。また、このデータ構造に対する操作ができる限り簡単に行なえるよう、双方向、頭付き、環状構造をもつリンク配置法を採用した。なお、1つのテキストを含むデータ構造の単位を節（node）と呼び、その並びをテキストファイルリストと呼ぶことにする。図3に節とテキストファイルリストの構造を示す。

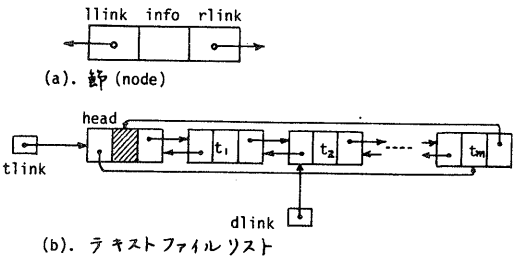


図3. 節とテキストファイルリストのデータ構造

ここで、tlink は現在編集中のテキストファイルリストを指示するリンク変数であり、dlink は CRT に表示するテキストのうち画面の先頭に表示されるべきテキストの入っている節を指すリンク変数である。dlink は次のように使用される。例えば、スクロールモードにおいて、画面を1行上げるとき、次の処理が行われる。

```
dlink := dlink↑.rlink ;
display ;
```

手続き display は、dlink が指す節のテキストから1画面分（最大14テキスト）を CRT に表示する機能をもち、基本的アルゴリズムは次のように表わされる。

```
procedure display ;
  begin
    display-clear ;                    "画面の消去"
    link := dlink ;
    i := 1 ;
    while (i≤13) and (link≠tlink) do
      begin
        video-display(link↑.info) ;    "テキストの表示"
        link := link↑.rlink ;
        i := i+1 ;
      end
  end
```

また、未使用の節は、図4に示すように flink に接続されてフリーリストを構成する。テキストの挿入時には、フリーリストから節を1つ取出し、info欄にテキストを置き、これをテキストファイルリストに挿入する。また、テキストが削除されたとき、このテキストを含む節はフリーリストに接続される。

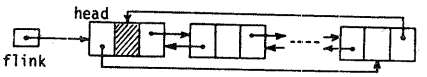


図4. フリーリストの管理

テキストファイルリストは、ディレクトリと呼ぶテーブルによって管理される。図5にディレクトリの欄構成とテキストファイルリストの関係を示す。text-file-name 欄には、テキストファイル名が格納される。used欄には、テキストファイルが有効であるとき"1"が格納され、削除されたとき"0"が格納される。text-link 欄には、テキストファイルリストにおける頭の節を指すリンクが格納される。text-link 欄の値は、そのテキストファイルの編集が開始されるとき tlink（前述）に代入される。

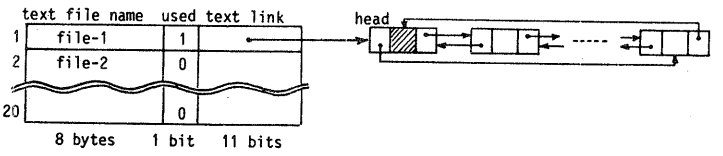


図5. ディレクトリの欄構成とテキストファイルの関係

4.2 仮想レコード方式

ここでは、前節で述べたデータ構造（仮想的データ構造）を計算機上で実現するための内部データ構造（具体的データ構造）、および、効率的なデータ構造の

操作を可能にするための仮想レコード方式について述べる。

4.1節で述べたデータ構造を実際に計算機上で実現するには、いくつかの問題点がある。第1は、主記憶容量の問題である。すなわち、テキストファイルの編集中、すべてのテキストを編集のために主記憶に常駐させるのは、一般の計算機の場合では、主記憶の使用効率を低下させる。また、本研究ではマイクロコンピュータ上のエディタを目標としているため、すべてのテキストを常駐させるだけの主記憶を用意することは困難であり、コスト上問題がある。

こうした主記憶容量の問題を解決する手段としては、マイクロコンピュータの仮想記憶化が考えられる。しかし、仮想記憶上に本システムのようなリンク構造をもつデータ構造を実現することには問題がある。すなわち、節を探索するために、頻繁なディスクのアクセスが必要になり、システムの効率が極端に低下することになる。

そこで、本システムでは、仮想記憶方式の利点を活かし、かつ、効率の低下を招かないために、節におけるデータ部（info欄）と構造部（llink,rlink欄）を分離し、これらをアクセス速度の異なる媒体、すなわちディスクと主記憶にそれぞれ置くことにした。この結果、テキストの探索は主記憶上の構造部を用いて高速に実行することができ、かつ、主記憶容量をはるかに越える大量のテキストを扱うことが可能になった。また、テキストはブロック単位（本システムでは、3テキスト／ブロック）にフロッピーディスクに格納されており、必要に応じて主記憶上のバッファに取出される。このとき、バッファ中の不要なブロックはフロッピーディスクに退避される。

この方式を仮想レコード方式（Virtual Record Method）と呼ぶ。仮想レコード方式と呼ぶ理由は、次の2点に基いている。

(1) 本方式が、複数の欄（field）から構成されるレコード型データ構造を仮想記憶的に実現する1つの方法であること。

(2) テキストは、テキストファイルの構成要素としてのレコードに相当する。本方式では、節へのリンクが仮想番地（virtual address）に相当し、テキストのバッファ中の番地が実番地（real address）に相当する。したがって、仮想記憶から実記憶への変換は、テキスト単位（レコード単位）に行われる。これに対して、一般の仮想記憶では、1記憶単位毎に番地変換が行われる。

4.2.1 内部データ構造

4.1節で示した仮想的データ構造の例と、これを計算機上で実現するための方法を図6に示す。

TT（Text Table）の各行は、それぞれ1個の節（正確には、節のリンク情報）を表す。また、各行はllinkとrlinkのリンク欄、およびR（Real memory）とCT（Current Text）の各1ビットから成る欄をもつ。本システムでは、節は3個単位でブロック（ディスクのアクセス単位）を構成する。

TBT（Text Block Table）の各行は、それぞれ1個のブロックを代表している。したがって、節番号とブロック番号の間には、図6に示すように固有の対応関係がある。TBTの各行は、TBL (Text Block Link)，TBI (Text Block Information)，MBA

(Memory Block Address)、およびDA (Disk Address)欄から構成される。このうち、MBA欄には、ブロックが置かれている主記憶のバッファ中の位置（バッファの先頭からの相対ブロック数）が格納される。また、DA欄には、このブロックが置かれているディスク中の物理番地（シリンダ、トラック、セクタ番号）が格納される。TBL欄とTBI欄については後述する。

主記憶のバッファ領域は、ブロックサイズ（256バイト）毎に区切られ、ブロック番号が付けられている（本システムでは、0～63）。

ディスクには、1セクタ（256バイト／セクタ）に1ブロックが格納される。

4.2.2 テキストの参照

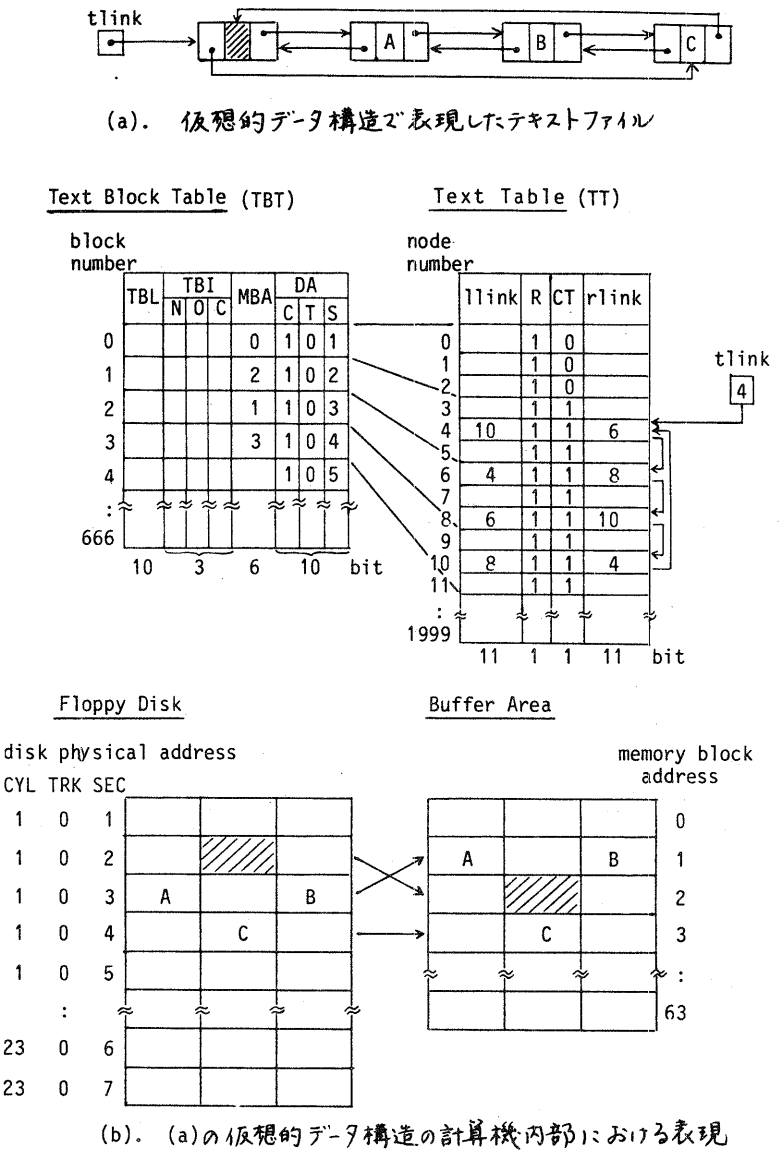


図6. 仮想的データ構造とその内部データ構造表現

テキストの参照は、以下に述べる手順で実行できる。

① ある節へのリンク、すなわち節番号をもとに、その節のテキストを知る番地変換の方法は次のようになる。すなわち、節番号に簡単な計算（節番号／3）を施してブロック番号を求め、そのMBA欄の値からテキストのバッファ中の位置を求める。

② 求めるテキストを含むブロックがバッファ中に存在するか否かを知るためにTTのR欄を使用する。R欄の値は、そのブロックがバッファ中に存在するとき"1"であり、さもないとき"0"である。求めるブロックがバッファ中にないとき（ブロックフォルトと呼ぶ）、ディスク中から必要なブロックをバッファに

取出さなければならない。求めるブロックのディスク中の位置は、TBTのDA欄に格納されている。したがって、ブロック置換を行ったのち、初めてテキストが参照可能となる。

4.2.3 テキストの挿入

表2. R及びCTフィールドの意味

prty	R	CT	意味
1	1	1	このブロックは、主記憶上にあり、tlinkにつながれているテキストをもつ。
2	1	0	このブロックは、主記憶上にあり、tlinkにつながれているテキストをもたない。
3	0	1	このブロックは、主記憶上になく、tlinkにつながれているテキストをもつ。
4	0	0	このブロックは、主記憶上になくtlinkにつながれているテキストをもたない。

テキストの挿入方法について述べる。テキストの挿入は、通常、自由リストから自由節を取り出し、そのinfo欄にテキストを格納し、テキストファイルリストの指定された位置にポインタを変更することにより挿入できる。その際、不要なブロック置換を発生させないために、表2の優先順位で自由節の選択を行う。ただし、あるブロック中に、現在編集中の（Eコマンド等で編集指定され、tlinkにつながれている）節が少なくとも1つ含まれるとき、そのブロックに属するすべての節のCT欄が"1"になる。

たとえば、図7に示す状態でテキストを挿入する場合、自由節eが最優先で選択される。また、選択された節が主記憶にない場合は、その節を含むブロックを主記憶に取出す操作（ブロックインと呼ぶ）が必要となる。

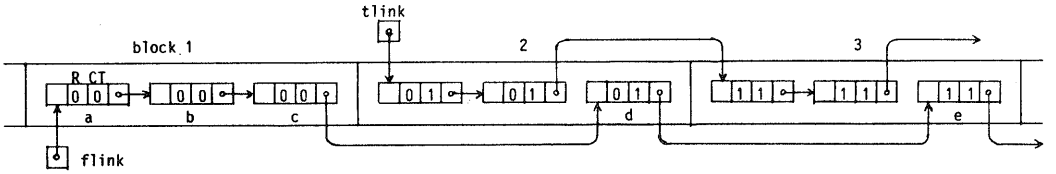


図7. テキストの挿入

4.2.4 ブロックアウトアルゴリズム

テキストの参照（置換を含む）および挿入を実行するためにブロックインが必要になったとき、バッファ領域に空き領域がない場合は、不要なブロックをディスクに追い出す必要がある。これをブロックアウトと呼ぶ。このためのアルゴリズム（再配置方策のアルゴリズムと呼ばれる）を以下に述べる。

ブロック置換において、置換回数はできる限り少なくて済むよう再配置方策を考える必要がある。これによって、①テキスト編集を高速化でき、②フロッピーディスクのヘッドの不要な摩耗を防ぐことができる。

再配置方策として、ここではLRU (Least Recently Used)法を少し修正した方策を採用した。すなわち、過去2回のブロックフォルトが生じ、従って2回のブロック置換が行われた後、現在のブロックフォルトが生じるまでの期間に、主記憶にあるそれぞれのブロックに、どのようなアクセスがあったかという状況を基にして追い出すべきブロックの選択を行う。この状況を記録するために、ブロック毎に次に示すようにnew-ref,old-ref,changedと呼ぶ各1ビットの欄を設けた。

new-ref欄：１回前のブロック置換が行われてから現在のブロックフォルトが生じるまでにテキストブロックへのアクセス（参照または変更）があれば"1"B、そうでなければ"0"B。

old-ref欄：２回前のブロック置換が行われてから１回前のブロック置換が行われるまでの期間にテキストブロックへのアクセスがあったときに"1"B、そうでなければ"0"B。

changed欄：ブロック置換に関係なく、この項目が示すテキストブロックの内容が変更されたとき"1"B、そうでないとき"0"B。

今、おのおののブロックの状態を(new-ref,old-ref,changed)で表わすとすると、これは８つの状態値のうちの１つを取り、その状態遷移は図８に示すようになる。あるブロックが主記憶上に留りうる権利の大きさ、すなわち優先度は、この状態値を２進数とみなしたときの値とみることができる。

しかし、このような選択基準を用いても、最小の優先度をもつブロックが２個以上存在することがある。このとき、次段の選択基準としてFIFO (First In First Out)法を採用する。すなわち、そのようなブロックのうち、最も早期にブロックインされたブロックを追い出すということである。この方策を実現するためにブロックインの時間的順序を示すフィールドtext-block-linkを設けた。

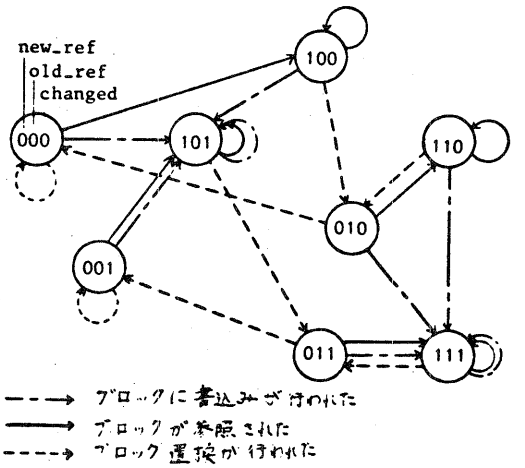


図8. 各ブロックの優先度の推移

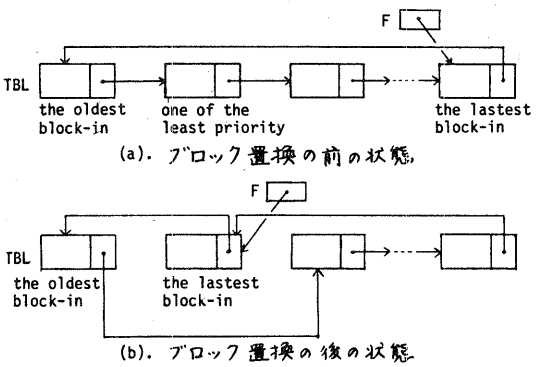


図9. ブロック置換前後の各ブロックの状態

このtext-block-linkフィールドには、その項目が対応するブロックがブロックインされた後、次にブロックインされたブロックのblock-numberが記入される。そして、ポインタFにブロックインされたブロックのblock-numberが記入され、そしてその block-number（Fの内容）に対応する項目のtext-block-linkフィールドに最も早期にブロックインされたブロック番号が記入される。すなわち、Fは最後にブロックインした項目を指し、最後にブロックインした項目のtext-block-linkは、最初にブロックインした項目を指す。

5. あとがき

マイクロコンピュータをホスト計算機と結合することによって、ジョブプログラムの処理（媒体変換、編集、遠隔ジョブ入力、等）に適した効率的なデータエントリ端末を実現することができた。

本稿では、実現したシステムの構成，コマンド、およびテキストの編集方法などについて述べた。また、本稿で示した仮想レコード方式は、データ構造における構造情報とデータを分離し、これらを異なる媒体上に置き、データのみを仮想記憶的に管理しようとするものである。これによって、主記憶容量を越える大量のデータをあたかもすべてが主記憶上に置かれているかのように取扱うことが可能になるとともに、データ構造に対する操作を高速に行うことができるようになった。本研究では、仮想レコード方式に基づくマイクロコンピュータのためのテキストエディタを実現することによって、本方式の有効性を確認した。なお、編集（テキストの挿入，削除、画面の移動など）に要する時間は0～3秒であり、実用上、充分効率的であると思われる。

本システムの今後の改善点としては、エディタの機能の拡張、マルチボリュームファイルの取扱いなどが挙げられる。なお、本システムでは、テキストを双方向リンク付きの節で実現したが、これを片方向リンクのみで実現すれば、リンクの操作は複雑になるが、主記憶領域は少なくてすむ。

謝辞

本システムの研究、開発に御協力いただいた本学情報工学科の磯泰行教授、および、情報工学科大学院生の石田博明君に感謝します。

参考文献

1) 相磯：機能分散型計算機システム，情報処理，vol.18,No.4,pp.325~333(1977)。
2) 田沼，日比野，小林：機能分散化とインテリジェントターミナル，情報処理，vol.18,No.4,pp.368~375(1977)。
3) 小林：情報構造，サイエンス社(1977)。
4) 野口：データエントリーシステム入門，オーム社(1978)。
5) 安在，工藤，重松，木村：仮想配列を割付けるための仮想記憶の実現，九州工業大学研究報告（工学），第38号，pp.81~88(1979)。
6) 野上，重松：マイクロコンピュータを用いた出力検索システム：FAMOUS1，情報処理学会論文誌，vol.21,No.2,pp.167~170(1980)。
7) 野上：マイクロコンピュータを用いた知能端末の研究，九州工業大学情報工学科修士論文(1980)。