レジスタ記憶装置*

淵　一　博**

1. はじめに——「レジスタ記憶装置」とは

「レジスタ記憶装置」とは耳慣れない言葉であるが，その意味するところは，基本回路と主記憶装置の間に存在する，もろもろの特殊（内部）記憶装置群のことであるらしい．

確かに，記憶は計算機の本質的な部分をなしており，それらの特殊記憶装置もまた計算機の中枢（central processor）の制御方式と深く関連し合っている．してみると，この特殊記憶と制御方式の交渉の問題は，論理（方式）設計者にとっても，また記憶装置開発者にとっても興味ある問題にちがいない．

また，この問題を統一的に論ずるのには，計算機のシステムにおける諸概念（の規定）の再構成も必要のようである．古典的な「レジスタ」という概念にしてもまた最近クローズアップされて来た pushdown store の概念にしても，ハード，ソフトをひっくるめた計算（情報処理）システム上の概念とみるべきであり，現段階では，両分野をまとめた，一種のシステム理論が必要になってきているのではないだろうか．論理設計者と，システム・プログラマの機能というのは，意外に似かよったものである．

とはいうものの，以下の試みはそのような観点からみてもはなはだ不満足なものである．ただ，これが本格的な議論展開のきっかけにでもなれば，幸いだと思う．

2. 実　　例

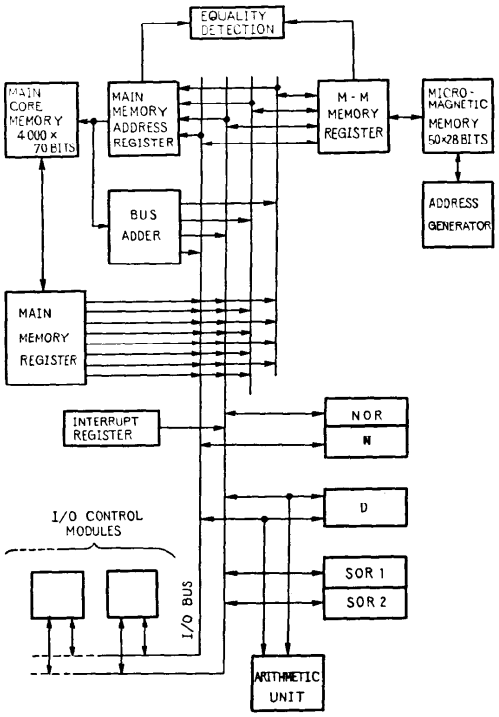
特殊記憶と方式の関連を考察するに当って，まず，いくつかの具体例を見てみたい．

2・1 RCA 3301

この機械は，外面的には既存の RCA 301 システムの拡張である．しかし機械内部に立ち入ってみると，論理構成がすっかり変っている．

その中心になっているのは RCA 自慢の micro ferrite core memory である．これは，語配列，2コア/ビット，線形選択の高速磁心記憶装置である．サイクル時間は 0.25 μs，容量は 200 字（1字7ビット）である．なお主記憶装置は，普通の磁心記憶装置で，サイクル 1.75 μs，容量 40,000 字（以上）となっている．

この micro magnetic memory が，それまでのフリップ・フロップによるレジスタ群の代用になっている．すなわち，アドレス・レジスタ，制御レジスタ，諸インディケータ，命令実行中の一時記憶用レジスタ，割り込み用の諸レジスタ，さらに指標レジスタなどがここに納められる．こによって内部方式はより洗練されたものになっている（**第1図**）．



第1図　RCA 3301 のブロック図

2・2 UNIVAC 1107

UNIVAC 1107 は，磁性薄膜記憶装置（サイクル

* Processor Organizations and Special Memories, by Kazuhiro Fuchi (Electrotechnical Laboratory, Tokyo)

** 電気試験所電子計算機部

0.6 μs)を最初に装荷した計算機として評判になった．その薄膜記憶は control memory として活用されている．

2・3 ILLIAC II

これには，flow gating memory と呼ばれるトランジスタを記憶素子とした特殊な記憶装置が使われている．容量は10語（1語52ビット)，呼出しは 0.15 μs である．これは，指標レジスタ，一時記憶用レジスタ，命令レジスタ，特殊レジスタとして使用されている．

2.4. ETL Mk-6

ETL Mk-6 では，トンネルダイオード記憶装置(0.25 μs) および薄膜記憶装置(0.5 μs)が特殊記憶装置として使われている．前者は指標レジスタ，演算の pushdown store，後者はプログラムスタック(ループをとらえるのに利用される，一種の大容量命令レジスタと考えてよい）として使用される．

2.5. HITAC 5020

HITAC 5020 には，遅延線を使った高速呼出しのレジスタ群がある．指標レジスタなどがここに納められるほか，これを利用した「多アキュムレータ」方式が採用されている．

3. 記憶と方式の一般論（1）

記憶と方式の相互作用は，大体次の三つの側面から観察できる．

3・1 記憶領域の構造化のレベル

central processor 内には機能からみてさまざまな記憶が存在する．

（1） まず，制御カウンタ，アキュムレータ，その他アドレス・レジスタなど制御に必要なレジスタ類．これらはそれぞれ個別的な機能をもった独立した記憶領域をなしている．最も高度に構造化された部分である．これを個別的記憶領域と呼ぶ．

（2） 一方，主記憶装置内の各記憶場所は，一様な構造の中にあり，機能もほとんど特殊化されていない．この領域を主記憶領域と呼んでおく．その構造の問題については，**5** および **6** で触れる．

（3） 上の二つのレベルの領域の中間に，いわば，特殊記憶領域と呼べる領域が存在する(ことがある)．たとえば，指標レジスタ（群）は，特殊な機能を持った領域である．普通には複数個あり，その中では交換使用が可能である．この類に入れられるのは，多アキュムレータ方式の場合のアキュムレータ群，一時記憶用レジスタ群，がある．また pushdown store もこの類に入れられる（B-5000, KDF 9, ETL Mk-6 などの例)．

3・2 記憶装置でのレベル

装置や素子においても次の三つのレベルが認められる．

（1） トランジスタ・フリップフロップのような基本的能動素子のレベル．利用上は最も自由度がある．一般に他の装置の素子に比べて高速であるが，また高価でもある．

（2） 主記憶装置．現在の標準的な構成では，これは磁心マトリクス装置である．

（3） 以上の中間にさらに特殊な記憶装置が存在することがある．第2節で出てきた，高速磁心，トンネルダイオード，磁性薄膜，特殊遅延線，トランジスタなどによる高速記憶装置がそれである．

その積極的存在理由は，「主記憶より高速」であるという点にある．また一方，主記憶より小規模，基本素子より安価かつ主記憶より高価ということであるという特徴もある．これらは技術的制約から来るのであるが，そういう特徴がなければ，基本回路あるいは主記憶装置によってとって代わられるか，または主記憶装置にとって代わることになるであろう，という意味で消極的存在理由でもある．

装置の方式からいうと，主記憶装置と同じことが多い．すなわち，アドレスによる（即時）呼出し方式がほとんどである．

特殊記憶装置を利用するのは，上の高速性と安価性を生かして，計算機全体を経済的にすることや機能を拡大することが可能になり，また計算機方式をより洗練されたものにできるからである．

3・3 構造化された記憶領域の機能の具体化のレベル

これには（1）純回路的な実現（2）論理回路的実現（3）ソフトウェアでの実現 の三つのレベルが認められる．各レベルはそれぞれ前の段階を前提としてなり立つものである．あとの段階になるにしたがい，具体化は容易になるという特徴があり，それから方式上の多様化の実現はソフトウェアから始まることが多いということになる．また各レベルの境界は固定したものではなく，相互に滲透し合っている．

4. 記憶と方式の一般論（2）

前節に述べた，三つの側面の各レベルは，互に関連し合ってはいるが，完全な従属関係にもない．たとえ

ば，特殊装置は特殊領域でなければならぬ，という理由はない．そこからも方式上の多様性が生れる．各側面の組合わせによって，記憶と方式の諸相が現われるが，そのうち特徴的なものを論じてみることにする．

4・1　特殊装置が特殊領域である方式

これは，むしろ常識的な形である．指標レジスタなどが，高速記憶装置に割り当てられているのはこれに当たる．前に指摘したように，特殊記憶装置も，装置方式としては普通の記憶と同じであるから，指標などの特殊な機能を実現するには，論理回路による具体化法を利用しなければならない．

4・2　特殊装置の個別領域化

制御上必要なジレスタ類を，特殊装置に納めることによって方式の洗練化をするのはこのカテゴリーに入る（実例の RCA 3301 など）．

4・3　特殊装置の主領域化

特殊装置を主領域化して使用することも方式の一つとなる．この場合，二つの行き方が考えられる．

（1）　主領域中の一部が実は特殊装置になっている形．たとえばアドレス 0～127 が磁性薄膜装置となっていて，そこの呼出しは特に速い，残りは磁心という構成は，これに当たる．方式としては原始的なものに属する．

命令構成の面からみると，その特殊性（この場合高速呼出し性）は表面に出てこない．特殊性の活用は，プログラミング技術（記憶割当てを通じて）にかかってくることになる．

（2）　特殊装置をかくして使用する方式

ATLAS や ETL Mk-6 などのページアドレス方式（前論文）はこれに当たる．ただし，それらではドラムが主領域で磁心を特殊装置として使う形になっている．もちろん，この手法は，磁心を主としトンネルダイオード装置を特とした場合にも適用できる．

ページアドレス方式の具体化の場合は，さらに連想記憶装置が必要であるが，ETL Mk-6 などでは，これは論理回路によって実現されている．

この方式では，特殊性の活用がハードウェア的に実現され，プログラミング上ではあまり考慮する必要がないという利点がある．

4・4　主記憶装置の特殊領域化

指標レジスタ（群）などは特殊領域に属するが，これを，特に，特殊な装置にとらなくても，主記憶装置の一部を借用しても実現できる．この場合そのレジスタの呼出しが高速であるという利点はない．しかし主記憶装置の一部を流用することから，（指標）レジスタの数を簡単にふやせる．また，ベース・アドレスの手法をつかえば，その領域を自由に移動できる．指標レジスタの場合，指標そのものの有効性と合わせて，この方式も最近よく採用されている．

また pushdown store の領域を主記憶の一部を借りて実現することも行なわれている．B 5000 などはそうであるが，ETL Mk-6 でもトンネルダイオード装置が使用不可能のとき，そのモードに切換えられるようになっている．

主記憶の特殊化は，実行が容易であり，ソフトウェアの開発から生れた方式上の諸概念をすぐさまハードウェア化するのに適している．

5.　主記憶の構造——アドレス型記憶

これは von Neumann の機械の流れをくむもので各記憶場所はアドレスが割りふられているものである．現実の（内部）記憶装置はアドレス型で構成されているのがほとんどであるから，装置の性質にマッチした方式だといえる．方式上，各記憶場所は平等であるが，この平等性は，特に即時呼出し型記憶装置の性質に合っている．

一方，アドレスがあるということ自体がこの方式の難点でもある．記憶容量が大きくなればなるほど，アドレスに必要な情報量も大きくなり，これが欠点になってくる．アドレス型の記憶を持った計算機の方式上の工夫は，この難点の回避に向けられる．

5・1　命令中のアドレス数を減らす

命令形式として基本的なのは（現実には行なわれないが）次の4アドレス形式であろう．

$$F-A_1-A_2-A_3-A_4$$

add 命令なら（記憶は一次元 array として ALGOL 式に書くと）

$$M[A_3]:=M[A_1]+M[A_2],\quad \textbf{go to } M[A_4]$$

アドレス部の数を節約するには，特殊レジスタを利用することである．制御カウンタ（CC），アキュムレータ（ACC）を設け，store 命令を分離すると

$$F-A_1$$

の1アドレス形式にまで持っていくことができる．これは add 命令の場合，

$$\begin{cases} CC:=CC+1 \\ ACC:=ACC+M[A_1], \quad \textbf{go to } M[CC] \end{cases}$$

と考えられる．

5・2　アドレスの長さを減らす

一般領域の広さを制限することによりアドレスの長さを短くする手法がある．しかしこの場合は実質的にアドレス長を回復するために「間接アドレス法」を使わなければならない．制限された領域名をIとすると

M[I[i]]

5・3　多アドレスの復活と複合命令化

上記手法によりアドレス部を短縮すれば，多アドレス形式を復活できる余地が生じる．多アキュムレータ方式もその一つと考えてよい．指標レジスタ手法も同じであるが，この場合はむしろ複合命令化と見た方がよい．**add i, a** は実質的に

ACC:＝ACC＋M[a＋I[i]]

である．

アドレス型方式の問題は結局，間接アドレス，指標などの手法で，如何にして一語長の命令中に，複雑で（ある意味では一般的な）機能を表現させるか，にあるといってよい．

アドレス型記憶では，プログラミング上，記憶場所の割当て（storage allocation）という問題があるが，これは自由度が大きすぎること（構造化の不足）から来るものである．

6.　主記憶の構造——リスト型記憶

この原型をなすのは Turing 機械である．しかし現実には，記号処理（コンパイルを含めて）の問題構造の分析から，まずソフトウェアの分野で問題にされ，その後機械構成としても，その利点が再認識されるようになった．といった方がよいようである．B 5000, KDF 9 などはそのような構成をとり始めた最初の計算機であるが non-von Neumann computer であると自称している．また演算がアドレスと分離されるところからゼロ・アドレス型だと呼ぶ人もいる．

6・1　ポーランド記法と pushdown store

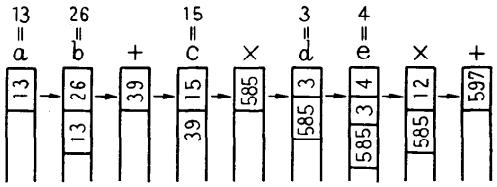
数式は演算子後置（前置でもよいが）の方法によるとカッコなしで，しかもあいまいさなく書ける．

〔例〕 (a＋b)×c＋d×e ⇒ ab＋c×de×＋

これを（逆）ポーランド記法（inverse Polish notation）と呼ぶ．

pushdown store（棚）というのは先入後出法（last-in first-out）の原理による作業領域である．「棚」の中での処理にはアドレスは必要でなく，容量も理論的には無限であるとしてよい．

ポーランド記法で書かれた数式は「棚」を使うと極めて自然に左から右へ実行できる（**第2図**）．a, b……



第2図　「棚」の動作

などが来れば，その数値を棚にのせる．＋×のような（2値）演算子が来ると，棚の上の二つの数値について演算し，結果を棚に残すのである．これによれば中間結果の貯蔵の問題は自動的に解決される．

棚を持った計算機ではポーランド記法の数式がそのままプログラムになる．

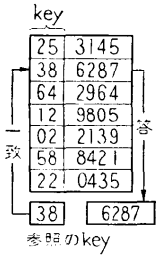
6・2　Turing 機械

上の例ではまだアドレス（a, b など）が残っていたが，複数個の pushdown store を使えばアドレスを消すことができる．Turing 機械は2個の pushdown store をもった例である．複数個の pushdown store を持った機械の応用については文献 1）参照．

6・3　連想記憶（associative memory）

アドレス型記憶では，アドレスによって内容が引き出されるのであった．その逆の操作も有用であり，table look up 命令はその操作をするものである．

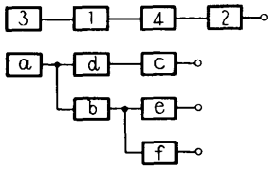
それを記憶装置の機能そのものに組み込もうというのが連想記憶の思想である．一般には，内容の一部を key とし，key に合ったところの残りの内容を引出せればよいわけで，アドレスそのものは必要でない（**第3図**）．



第3図
連想記憶

6・4　リスト処理言語

記号処理のためには，線状あるいは樹状につらなったデータ（**第4図**）を処理することが必要である．そのようなデータの構造をリスト構造と呼ぶ．IPL V[2]



第4図　リスト構造

などは，リストの処理手順を表現するために開発された言語である．リスト構造は極めて自然なもので，文章などもリスト構造を持っているから言語の翻訳（その一部としてコンパイル）手順の表現にもリスト言語が有効である．

リスト処理では，リストへの項目の追加，リストからの項目の除去，ある項目があるリスト内にあるがどうかの探索などが基本的な操作であるが，その中に今まで述べた，pushdown store，連想記憶の概念が自然に含まれることになる．

6・5　リスト型記憶の具体化

リスト型記憶はアドレス型に比べ，記憶が構造化されているだけ有効であるが，そこに難点もある．本質的にリスト型の記憶「装置」が現実には得難いのである．記憶自体が構造化され，より多くのロジックを含んでいるだけに，能動記憶素子を使いたいところである．そういう観点からか，クライオトロンを使用した連想記憶装置の試作（案）[3,4,5] が発表されている．また磁心を利用した試作もある．しかし，いずれもまだ実用段階ではないようである．現在の段階では，むしろ普通の磁心マトリックス装置を使って，論理回路的に具体化する方が現実的だと思われる．

pushdown store を具体化するには，棚のあたまを指すレジスタ（P）を設け

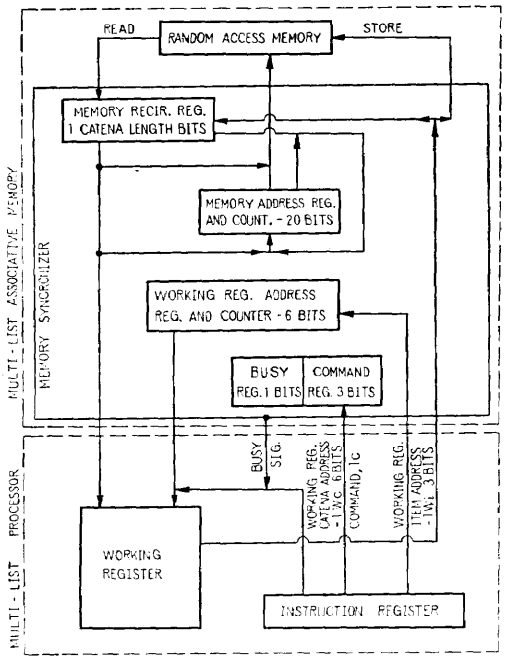
bring の場合　　＋operator の場合

$$\begin{cases} P:=P+1 \\ M[P]:=X \end{cases} \qquad \begin{cases} M[P-1]:=M[P]+M[P-1] \\ P:=P-1 \end{cases}$$

のようにすればよい．連想記憶の場合は，table look up 命令と同じようにすればよい．

リスト処理機械として IPL V を基にした機械の設計[6] が発表されている．その連想記憶の部分のブロック図を**第5図**に示す．

もともとアドレス型の記憶装置をリスト型に変成してしまうことについては，まだ議論があり，その利害得失については，今後の実験に待つところが多いようである．



第5図　連想記憶の構成

参考文献

1) R.J. Evey: Application of Pushdown Store Machines, Proc. FJCC 1963
2) A. Newell (ed): Informatioin Processing Language-V Manual (Prentice-Hall 1961)
3) P.M. Davies: A Superconductive Associative Memory, Proc. SJCC 1962
4) J.L. Newhouse et al: A Cryogenic Data Addressed Memory. Proc. SJCC 1962
5) R.F. Rosin: An Organization of an Associative Cryogenic Computer, Proc. SJCC 1962
6) N.S. Prywes et al: The Multi-List Central Processor, Proc. 1962 Workshop on Computer Organization