

最近の計算機方式の動向*

浦 城 恒 雄**

1. はじめに

計算機の最近の方式的な動向を survey して解説せよとのことですが、考えてみますと、ここ 2, 3 年に計算機利用は確実に実用化の時代にはいり、方式的な意味で革新的変化といったものはみられません、もはや大学、研究所といったところから閉じた形で計算機システムが開発され、発表されることは少くなり、主として企業ベースの、あるいは企業ベースを目標としたメーカから、経済性、市場性をふまえた計算機システムが世に提供されはじめたといえましょう。国内、アメリカ、ヨーロッパにおいて発表される計算機システムは、年に数十種に及び、その規模も、価格にして数百万円から数十億円に及びます。適用範囲ももはや枚挙にいとまなく、事務用、科学用、制御用といった分類に加えて、汎用という分類が手をふって通用しているような状態です。

こういったなかで方式的な動向といったものを組織的に導きだすことは、一人の作業としてはなかなか大変なことです。本誌の 62 年 7 月号の文献紹介にもありますが、IBM の Amdahl¹⁾ が、計算機方式設計上の新しい概念と題して、方式設計上の目標となることは、

- プログラムと編集の助けで動作効率と融通性を増すこと
- 計算機を高速化すること
- システムの金物を一層平行して利用するために多重プログラミング、多重処理をおこなうこと
- 高い融通性をもたせるためにシステムの変換が容易であること

であると述べています。ここでもこれらの問題に焦点を合わせて、最近 deliver したものの、また近くに deliver されるものにあたりながら、動向といったものをさぐってみたいと思います。

2. ソフトウェア技術との関連

計算機の方式設計は計算機システム全体の中では、

* Recent Trend in Computing System Design. by Tsuneo Uraki (Kanagawa Plant, Hitachi, Ltd., Yokohama)

** 日立製作所神奈川工場

入出力装置、回路などの純金物の設計と、ソフトウェアシステムの設計との中間にあって、両者の仲人役ともいえます。計算機システムが新しく誕生するには、仲人役は不可欠ですが、しばしば片一方からの強い要求で成立が要請されることがあります。かつては金物技術の方の要求から仲人を引き受けてうまくまとめていくといった場合が比較的多かったのですが、今ではソフトウェアの方の要求による場合がだんだん強くなっているのも最近の大きな特色といえます。

既にメーカ自体は金物とソフトウェアに同程度の金額を投じ、むしろ後者がふえる傾向にありますし、ユーザが投じている金額をふくめると、IEEE の委員会が²⁾、アメリカにおいては 1962 年度で、金物に約 1 億ドル、ソフトウェアに約 5 億ドルを投じたと発表していることは、この傾向をよく示しています。商業ベースにのせるには、小中形機では百台を、大形機でも十台を単位としなければなかなか pay しないということからしても当然なことかも知れません。

さらにこの傾向に拍車をかけているのは、金物の寿命に比べてソフトウェアの寿命がはるかにながいことです。ソフトウェアシステムをもっとも力をいれて大規模に作りあげた IBM から汎用の中大形機として発表された最近の機種が、7094, 7044, 7040 というシリーズであり、それらは今までの 704, 709, 7090 というシリーズの family に属し、ソフトウェア的にかなり高度の compatibility をもっていることを思いおこしてもうなづけます。さらに事務用を主眼として最近発表されたのは 7010, 1460, 1440 であり、これも 705, 7080, 1410, 1401 といったシリーズと同じような関係をもつものです。

IBM が STRETCH project 以来、方式的にも数多くの研究を重ね、また 704 開発当時にくらべると金物的技術にも格段の進歩をみせながら、いまだに数表現法、語長といった基本的なものをかえないばかりか、いままでのものとの compatibility にできるかぎり重点をおいているのは、あまりにも頑固だともいえますが、またあつぱれなものだともいえるでしょう。同時にソフトウェアシステムも、assembler, compiler さらには operating system においてできるだ

けの **compatibility** を確保して実質的に使用者に十分に **family** としての意識をもたせるように努力してきたあとをみると、ソフトウェアシステムの連綿たる力が方式設計者にある意味でしめつけるさまが想起できるというものです。

逆に 50 年代に IBM ほどの実績が獲得できず、したがって強大なソフトウェアシステムを作るにはいたらなかったところでは、現時点にたつて今後の技術的進歩をある程度想定し、十分に検討を加えて、規模、性能は異なるけれどソフトウェア的にはかなりの **compatibility** をもった 2, 3 種でラインを作り、今後当分の間はそれでおしすすもうとしているところがあります。SDS の 910, 920, 930 のシリーズなどはその典型例で設計もほとんど同じに出発しています。

通常はいわゆる **upward compatibility** が問題となり、より小さいもののプログラムがより大きいものにかかるようにするわけですが、SDS のシリーズにみられるように逆もある程度可能であるように考慮するのも最近の特色ある動向といえます。たとえばある **instruction** は大形のものには金物的に実行できるが、小形のものではその **instruction** の実行に際してはいわゆる割りだし (**program interrupt**) がおこり、大形のものが金物で実行することと同じことを **subroutine** で処理できるようにするといった方法がとられています。

一方、**operating system** の方からはたとえプログラムの誤りといえども **instruction** の **code** 表の空のところできるとまるとなことは困ったことで、メッセージを印刷するなどのなんらかの手当てができるようにしてほしいという要求があります。これらの要求は実は一致するわけで、これを **extra code** などと呼んで処理する方式がいくつかの計算機で採用されています。たとえば金物で実装されていない **instruction** が実行を指定されると特定アドレスへ割りだされ、そこで割りだしをおこした **instruction** の **code** をしらべて必要な **subprogram** へ **jump** するということで二つの要求を共通に満足させることができます。

計算機の利用が完全に実用段階に入ると、使用者側もそれを商業ベースで使おうとします。その結果は計算機に最大の能力を発揮させようと試みるのは当然なことです。ソフトウェアシステムも単に **aid to programming** という段階から **aid to operating** という段階にまできたといわれていますが、金物も当然その影響をうけています。

使用者は計算機をソフトウェアシステムをとおしてマクロに評価するようになりました。その結果 **assembly system** も従来のものにくらべてはるかに機能がふえ、逆に **assembling** そのものに時間がかかるということになりましたし、さらに **problem oriented language** を使用する傾向がますますふえていきます。ある統計では使用時間の 30% を **assembling, compiling** で占めると報告しています。

したがって方式的にもこの点に着目して **programming cost** を減らすことに努力するわけですが、従来ならともすると **sophisticated** な **instruction set** を設ける方向にいきがちでした。たしかに有能なプログラマが機械語またはそれに近い形でプログラムをしたときは、それらを駆使して能率を向上させうるにちがいありませんが、大部分のプログラムが **problem oriented language** で書かれるに従ってかえって **source language** を optimum な **instruction set** の組み合わせに変換するのが面倒になり、宝のもちぐされになりかねません。

むしろ傾向としては **assembling, compiling**, さらに **debugging**, そしてこれらを間断なく続けて実行するための **supervisory** な **operating system** に必要な機能を重要視するようになりました。従来のともすると **raw arithmetic** の速度を性能の **measure** としての傾向はうすらぎ、また単純な意味で科学用といった概念もうすらいで、効率よく可変長情報を扱うこと、**table search, data record** の **scan, list processing, out-put** のための **editing** といったことが、主として科学用をねらったシステムにおいても重要視されはじめたわけです。

一方、事務用といわれる計算機ではもともと上記の機能を重視していましたが、事務計算も単純なデータ処理から、次第に企業全体をシステムと考えて経営能率を高めようとはかりはじめました。そのため多量の算術量を含むデータ処理をおこなうようになり、科学用に要求されていた性能、さらに **compiling** に必要な機能も要求され、前にも述べたように二つの使い方を融合してひと口に汎用計算機という名でよぶようになりました。

こういう傾向に対処してどういう動向がみられるかあたってみることにします。

このための一つの方法としては、必要な **functional sequence** になるべく効果的に構成できるような、論理的にはきわめて原理的な **instruction** の **set** をつく

る傾向です。プログラミングでも複雑な coding technique よりもむしろ簡単なものよしとする傾向が金物の方式でもいえるわけで、いわゆる microprogramming の技術とも結びついて既に RW 530⁹⁾ などで実現しています。

もう一つの方法としては既にいろんなところで興味深く紹介されている Burroughs の 5000¹⁾ にみられるものです。execution の段階において polish notation を採用するような、problem oriented language oriented な machine とでもいうべき計算機があらわれ、problem oriented language が直接実行されるような計算機の可能性すら示しているように思われます。arithmetic expression の compiling technique にあらわれた概念である push down store, pop up load を金物的に実行するものです。この点のみならず B 5000 は von Neumann 以来のいくつかの思想や伝統や技術を大胆にかなぐりすてた感じを与えます。ある意味でソフトウェア設計者に屈服した、ある意味で挑戦したともいえる計算機です。COBOL, ALGOL を基調に、arithmetic operation, data handling のいずれにも重点をおいた汎用の計算機として今までのべたいくつかの動向を典型的に示しているといえます。

storage allocation を自動的におこなって能率を向上させようという方向もあります。storage にいくつかの種類があり、access time にも hierarchy があるとき、通常ならソフトウェアシステムでカバーするところを金物との協力でやろうとする、よく知られた Ferranti の ATLAS⁹⁾ の page address 方式が代表例です。またこれと類似なものとして B 5000 はプログラミングのとき pseudo memory array を symbolic に address し、compiling のときに symbolic address を storage 内の actual address の array table の entry に replace します。すると execution time では compile time に set された table によって memory refer が indirect におこなわれます。こうした方式は indirect addressing のもっと徹底したものといえますが、やはり direct refer にくらべると 1 サイクル余計かかっている感じで、いつもこうすることには problem oriented language が支配的になっていく傾向は認めつつも方式設計者としてはいささか抵抗を感じないわけにはいきません。

3. 多重プログラム、多重処理の技術との関連

本誌に、かつて高橋茂氏⁶⁾ が時分割処理の動向を解説され、和田英一氏⁷⁾ がモニタシステムの解説をされて、多重プログラム、多重処理の技術と計算機方式の関連が説明されています。多重プログラムと多重処理について Amdahl¹⁾ はつぎのように定義しています。

Multiprogramming: Time sharing of a single CPU (Central Processing Unit) for a variety of tasks.

Multiprocessing: Load sharing on one or more problems by more than one C.P.U.

多重プログラミングも最初はおもなねらいが入出力装置と C.P.U. が記憶装置を時分割で使おうとすることに端を発し、internal time を有効に使用し、いわゆる SPOOL を実行するために方式的には priority interrupt の技術を取り入れて効率をあげようとした。その結果、原理的には多重プログラミングの可能性がでてきたわけですが、当初はソフトウェアの方も follow できず、またやってみようとしても実際には方式的にも欠陥があらわれてすぐには実用になりませんでした。

IBM の 7070, 7090 においても入出力の control system をいわゆる buffering technique を駆使して非常に便利な operating system にまで作りあげてはいますが、一応入出力との同時操作という目的ははたしているもののまだむだ時間が多いようです。7090 のある設置場所での統計によれば計算機の動作時間の約 50% は磁気テープのみが動作しており、約 35% は入出力装置と計算が同時に処理されている時間、計算のみが 15%, 残り 5% がカード入出力装置、プリンタのみが動作している時間といった内訳を示しています。IBM の IOCS をもってしてもまだ過半の internal time を遊ばせているわけです。

そこでさらに効率をあげるために、または実時間処理と同時に他の処理をするために supervisory program の監視のもとで多重プログラムを実行するようにもくろむことになりますが、方式的にもいくつかの問題点があります。

まず和田氏も指摘された memory protection の問題があります。多重プログラムの一つに debug 中のプログラムなどがありますと他のプログラムまたはそれに属する記憶場所をおかす可能性があり、さらに悪

質なものも **supervisory program** をおかすことになり、せっかくの **monitor** も泣いてしまいます。この点に関してたとえば **RCA 601**⁹⁾ では **memory access** に際し、上限、下限をもつ **limit register** でしるようになっていて、おかすと割りだしがおこります。IBM 7044, 40¹⁰⁾ についている方式を説明しますと、このために二つの命令があり、状態に **PROTECT MODE** と **NON-PROTECT MODE** があります。命令 **SET PROTECT MODE** により命令の有効アドレスの上位7ビットが **field register** にセットされ、さらに比較されるべき上位からのビット数、比較の結果等しいときに割りだす **mode** か、等しくないときに割りだす **mode** かを指定します。なおこの命令自身が既に **protect mode** にあるときだされると命令自身が割りだされます（これは重要なことです）。そして **protect mode** では **store type** の命令が実行されるとき指定されたビット数だけ **field register** と比較され、指定 **area** 外の場合は割りだしがおこります。**RELEASE PROTECT MODE** により **PROTECT MODE** は解除されます。

多重プログラミングに関連した方式的な問題点としては他にプログラムとデータの **relocatability** をソフトウェアだけの力で処理させるか、または金物がそれに協力するような方式にするかといった問題もあります。

supervisory program が多重プログラムを制御するとき、計算機のいろいろな内部状態を **refer** し、さらに変更する必要がおこります。完全に多重プログラムを実行させるにあたっては、**supervisory program** は中核になるわけで、その作成には細心の注意が必要ですが、金物の方式も確実にそれを支持するものでなければならず、方式設計者にとってはもっとも頭をひねる仕事となっています。

interrupt の方式そのものにいろいろのものがあリ一重レベルから多重レベルのものまであります。また **interrupt** の要因としては通常は入出力装置の処理終了、入出力装置の異常発生、外部からの信号、内部演算において発生した異常などがおもなものです。前記の **extracode**, **memory protect** もそうですがさらに **tag bit** によるものがあります。**von Neumann** 以来記憶装置には命令とデータが区別なく入っており、命令の読みだし **stage** で **refer** されたときに命令情報と扱うのが大部分でしたが、記憶装置において従来までの情報に加えて、さらに 1 word に 1~数ビットの

tag bit をつけておき、これにより情報の分類、さらには境界を示す目印といった目的に使用します。特殊の命令でそれを操作したり、記憶装置を **operand** として **refer** したときにはあらかじめセットされている **tag register** を調べることを演算とは別に実行してその結果を **interrupt** の要因に加えるような使用法があります。**RCA 601** は半語（情報 24 ビット）ごとに 3 ビットの **tag** をもち、種々の目的で利用しています。**Telefunken** の **TR-1** のように情報の種類わけに使う程度の簡単なものもありますが、これすらも **tracer** や **dump routine** には役だつことでしょう。

次に多重処理の問題にも少しふれてみます。多重処理という言葉にはまだ概念も明確でない感じがですが、**multiple computer** といったものもこの範疇に入ると考えられます。この問題もなかなか議論も多いところでして、はたして **multiple-computer system** にすることによって能率が向上するだろうかという素朴な疑問がまずでることでしょう。一つの **job** を同時にいくつかの計算機で協力してやるというのも考えられますがなかなか大変でして、むしろ **job** を、ある程度大きく **phase** で分割して、いわば **separate job** にしていくつかの計算機に分担させ、それらの調停役、仕事の割り振り役、入出力装置の割りつけ役などをする **monitor** の機能をもった計算機に統括させて仕事をおこなえば、効果的なシステムもできるかも知れませんが **CDC** が 6600 システムでそういったものを計画していますが完成がみものです。

一方、信頼度を向上させるため **dual system** としての **2-computer system** を採用し、主として実時間処理をおこなうものがあります。**SAGE** などの軍用のものから、**banking system**, **reservation system** などに実用化されています。

これらにくらべると弱い結合であるところの大形計算機—衛星計算機システムは非常に普及しています。**IBM** の 7090—1401 システムがその代表例です。この場合などは磁気テープを介して接続されているだけで、結局は低速入出力装置（カード入出力装置、プリンタ）と磁気テープの変換装置として小形機を使用しているだけで、むしろ大形機に多重プログラムの一部として十分に吸収可能なように思えます。こうしたところから、大形機でありながら、編集能力も兼備したような計算機も **RCA 601**, **B 5000** などいくつかに見られます。しかし安易に衛星計算機システムを採用している方が多いのは実際問題として保守の問題、oper-

ating の方法の問題、多重プログラムをこなすソフトウェア技術の問題などの関連からそうなるのでしょう。

4. 高速性の追求、金物技術との関連

計算機システムの性能の評価に、最近では **cost performance** とよばれる **measure** を導入しようとしています。この単位コストあたりの **performance** の決めてになるのはなんといっても速度です。Grosch の法則¹⁾とかがあって、計算機の速度はコストの2乗に従って増加するという経験則が広くいわれています。ここでこの法則の確かさを議論はしませんが、ここ2, 3年の計算機の速度の向上は特に中、小形ではめざましく、比例定数が **first delivery time** の関数となっています。

ま先に計算機の方式設計はソフトウェア勢力にいささか押されきみのようなことをのべましたが、それはやはりマクロなはなしで、もちろんいままで **deliver** してきた計算機、ソフトウェアシステムが大きい影響を与えますが、日進月歩の金物の技術をその時点において **available** なものをもっともバランスした形でまとめあげる方式設計の腕のふるいどころも残っています。

計算機の方式設計に際して、算術演算の方式とか、数表現法とかいったことも重要ですが、なんといっても入出力装置、記憶装置、演算装置をどのような方式で結びつけるかということが最大の眼点となります。入出力装置自体も、計算機の利用の増大、処理速度への要求増大により、単に速度が向上するという以外にもいろいろな動向が見られます。ここでは大容量のランダムアクセスメモリと実時間に多数の低速入出力を接続し、処理することが最近の大きな問題であることを指摘するにとどめましょう。

すると、計算機の方式におかまいなくでてくる入出力装置をかかえて、次々開発されるいろいろなものを接続可能とする問題、さらには使用者がシステム設置後に処理量がふえたのにともない入出力装置を追加することに対し、あらかじめある程度手をうっておくこと、さらにこれら種々の入出力装置をできるだけ低コストで速度に見合った形で計算機に接続し、利用できる状態にしておくこと、などの方式的な問題がいろいろあります。事務用の場合はとくに重要なポイントといえましょう。

ついで重要なことは主記憶装置と演算制御装置の関

連でしょう。ここにおいても、両者のバランスをとり、計算機の全体のコストに対して処理能力をどの程度のものにするかという方式検討がなされます。主記憶装置は磁気コアがほとんどをしめており、サイクルタイムも50ミル径のもので4~12 μ s、30ミル径のもので電流一致方式で2~4 μ s、語配置方式で0.7~2 μ s程度のものが普通になりました。しかも次第に高速化の傾向にあり、コストもかなり低減していることは方式的にもきわめて影響の大きいことです。磁気コア以外にすでに薄膜メモリも一部実用化し UNIVAC 1107 が商業ベースとしての最初の計算機になりました。1107にも見られるように数十~数百十語程度の高速(主記憶装置に対しさらに)記憶装置をもち、方式的にそれを有効に利用するという傾向はますます強くなっています。いわば **register memory** として使用するわけで、各種演算 **register**, **index register**, さらには入出力制御用の各種 **register** をそれにあて、**scratch pad memory** というおもしろい命名をうけています。

さらに固定または半固定の記憶装置の利用も実用化の段階にはいり、前記の **microprogramming** と結びついたもの、あるいは **supervisory program** に利用するものなど固定メモリとしての機能を有効に利用しています。固定メモリはそのねらいは **read access** の高速性をねらうもの、非破壊であることを利用するもの、コストが安いものを利用するものの三つがあり、今後ますます利用される傾向にあります。

現在 **available** な回路、記憶素子を利用するという条件内でできる限りの速度をえようとする試みとしては、いわゆる **overlap** と **look ahead** の技術が典型的です。**overlap** は **full memory cycle** 対してデータや命令の **effective access time** をはやめるために **memory bank** ごとに **access** の機能をもたせ、それを **overlap** して使用方法です。**STRETCH**, **LARC** で採用されましたが、その後 **PHILCO-2000** (212型)、**CDC 3600**, **1604**, **UNIVAC 1107**, **RCA 601** などがとりいれています。この方式は中形以下では必ずしも採用しがたく、記憶容量を **expandable** にするとき、**overlap control** に影響を与え、むしろかしくなります。

一方 **look ahead** も **STRETCH** で試みられたようなものから、いわゆる **advanced control** の程度のものであります。前記の **register memory** として以外に、プログラム、データも高速の記憶装置にもって

きて処理する方法は相当の能率向上となります。この際、高速メモリは命令に関しては **first in first out type** になり、**instruction** の **look ahead**、さらには **loop** を捕える試みがあります。電気試験所 MK-6¹⁰⁾ が前から計画しています。一方データは **last in first out** の構造にすると前記の B 5000 のように **arithmetic** の際の間中間結果の **store** に役立ちます。あくなき高速性への追求はこのようにしていろいろの方式的 **idea** を生みだすもので、今後もいろいろ試みられるでしょう。

5. おわりに

組織だった **survey** からはまったくほど遠い、漫然たるおはなしになってしまいました。ふれ残した問題点も多くあり、需要のもっとも多い事務用小形機についてはほとんどふれなかったのは心残りです。この問題はむしろもっと精通された方に、問題を限定して解説していただきたいと思います。

参考文献

- 1) G.M. Amdahl: "New Concept in Computing System Design" Proc. IRE, Vol. 50, No. 5, 1962, pp. 1073~1077
- 2) "1963 Winter General Meeting of IEEE. Ad Hoc Group of Computer Devices Committee" Datamation Vol.9, No. 3, 1963, pp. 19

- 3) H.M. Senarne, et al: "A Stored Logic Computer" Datamation Vol. 7, No. 5, 1961, pp. 33~36
- 4) W. Lonergan et al: "Design of the B 5000 System" Datamation Vol. 7, No. 5, 1961, pp. 28~32
"The Descriptor — a definition of the B 5000 Information Processing System" (Manual)
- 5) T. Kilburn, et al: "The ATLAS Supervisor" Proc. E.J.C.C. Vol. 20, Dec. 1961, pp. 279~294
T. Kilburn, et al: "ATLAS Operating System Part I" Computer Journal Vol. 4, 1961, pp. 222~235
- 6) 高橋茂: "電子計算機技術の動向" 情報処理, Vol. 1, No. 3, 1960 pp. 125~131
高橋茂: "並列プログラミングと割り込み" プログラム懇談会資料, 1962-2-20
- 7) 和田英一: "モニタシステム" 情報処理, Vol. 3, No. 5, 1962, pp. 267~277
- 8) A.T. Ling, et al: "The RCA 601 System Design" Proc. E.J.C.C. Dec. 1960, pp. 173~177
- 9) IBM 7044 General Reference Manual
- 10) 相磯秀夫, 石井 治, 吉広和夫: "ETL Mk-6 の先廻制御について" 電気通信学会, 電子計算機委資料, Oct. 1962

(昭和 38 年 5 月 30 日受付)