よって，ブラシがセンスしている間に，1行分のデータを伝送する．

カード・パンチでは，1行のデータをパンチするごとに，割り込み信号を送って，次の1行のデータを要求する．

磁気テープ装置では，RCV や TRA で装置が動きはじめ，次いで SDT によりデータ伝送が行なわれるまでの間，約 5.5 m sec の間は計算機は，他の処理を実行できるのであるが，現在の装置ではこの場合，割り込みは関与しない．しかし，これでは不便なので，近いうちに次のように改良されるはずである．すなわち，まずテープをスタートさせ，データの読み書きができる時期になったら，割り込み信号が出され，そこで RCV（あるいは TRA），SDT によって情報伝送を開始する．こうすると計算機は，テープのスタート時間を安心して他の処理にふりむけることができる．

ライン・プリンタについてはすでにのべたので省略する．

6. むすび

G-20 の Communication line system は，周辺装置結合の一つの考え方を示したものであり，ゆう通性と安価という点では たしかに一つの特徴を示していると思われる．しかし一方，多くの周辺装置が結合されたとき，Interrupt Service Routine がかなり面倒になることは明かで，Comm. line を時分割で能率よく用いるためにかなりプログラム技術上くふうしなければならぬと思われる．

〔注〕

（1） Cresap, McCormick and Paget: Central Processors for Medium, Intermediate, and Large Size Computers, Control Eng. Oct. 1962. p. 103～109.

（2） Werner Buchholz: Planning Computer System, McGRAW-HILL, INC. 1962.

（3） General Information Manual, 709～7090 Data Processing Systems, IBM, 1960.

（4） R.E. Porter: The RW-400-A New Polymorphic Data System, DATAMATION, JAN/FEB. 1962.

（5） Input-Output Specification For Contral Data 160 and 160-A Computers. C.D.C.

（6） Bendix G-23 General Reference Manual, Bendix Computer Division, Bendix Corp., 1962

（昭和37年11月9日受付）

モニタシステム*

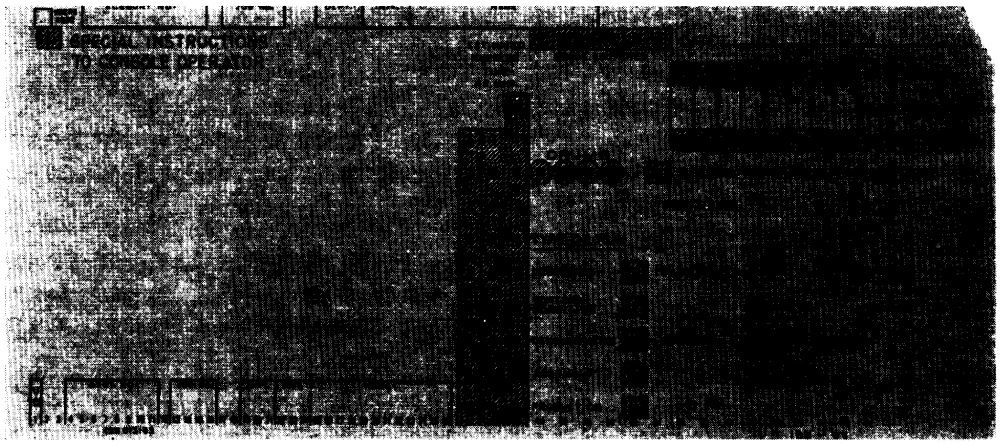
和田英一**

1958 年のとしのくれのことですが，おりあってボーキープシー（Poughkeepsie）にある IBM の計算センターを見学することができました．当時そのセンターでは 704 をつかっていました．オンラインのプリンタから結果がながれるようにでてくる様子は，まだ PC-1 しかつかえなかったぼくにとって，まったくショックでしたが，特に，そのハードコピーをみていて，おどろいたのは，ところどころに時刻がプリントしてあることでした．そこでさっそく，オペレータにたずねますと，なんでも，モニタシステムとかいうものがあって，プログラムがつぎつぎと実行され，その開始，終了の時刻が時刻をしめしているレジスタからよみだされ，おかねをとる都合上，プリントされているのだ．というふうにいわれたとおもいます．いずれにしても，こういう意味でのモニタということばをきいたのは，これが最初でした．

プログラムを順々に，自動的にかたずけていくというかんがえは，ぼくもそのまえからもっていたので，具体的にどう実現するかは別として，さしておどろきませんでした．また時計をつけることも，それほど奇抜なこととともおもいませんでした．そのときは，やが

* Monitor Systems, by Eiiti Wada (Data Processing Center, Onoda Cement Co., Ltd.)

** 小野田セメント株式会社



第1図 IBM 計算センターのモニタコントロール用カード 1958 年ころ

てそのうち，ほとんどすべての計算機が，こういうシステムでうごくようになるだろうと，おもっただけでした．PC-1 のように，使用料のやすい，そして速度のおそい機械でも，ときどき，プログラムとプログラムのあいだで計算機がとまるのを，もったいないとおもってみていたものですから，高速，超高速の計算機なら，プログラムを連続に実行するということくらい，だれでもすぐかんがえつくことです．

ポーキープシー(Poughkeepsie) の計算センターでは，おもしろいことをやっていたのをおぼえています．それは，モニタをコントロールするカードが，プログラムデックのまえに1枚つくのですが（1枚しかコントロールカードがないとは，モニタもずい分かんたんなものだったからでしょう），そのパンチのしかたは，さん孔機であけるのではなく，カードにはじめからパンチの部分にミシンがはいっていて，鉛筆でそこをつつくとあながあくというものでした．むかし，少年クラブなどの附録にあった，あつがみでふでたてや戦車をくみたてる模型が，はさみなしでかんたんにきりとれるようになっていましたが，ちょうど，それとおなじでした（**第1図**のハッチしてあるところがうちぬけます）．

余談はさておき，最近の大型計算機では，モニタがあるということが，しばしばみみにつくようになりました（ここで先日の電子計算機ショーであつめてきたカタログをしらべたところ，国産の各計算機のソフトウェアの項目にモニタありとかいたものはまったくありませんでした）．そこでモニタがどうなっているかを解説せよといわれたのですが，実のところ，ぼくもそうよくしっているわけではないので，むしろここでは，モニタはどうあったらよいかという点を主にしてなにかのべてみたいとおもいます．まず，モニタの定義がどうかかれているかを 2～3 しらべてみますと，

IBM の用語集[1] では

MONITOR: To control the operation of several unrelated routines and machine runs so that the computer and computer time are used advantageously.

となっています．また SOS のマニュアルでは

MONITOR: A routine which exercises supervisory control over some other program or collection of programs. When the collection of routines comprises all those normally used in the operation of a computer the Monitor and the entire collection is called an operating system.

となっています．まえのが動詞で，あとのが名詞になっているのが愉快です．またあとのが Operating System の説明までやっているのは SOS が SHARE Operating System の略であることをおもえば，当然かともおもいます．ところが，ひとつ，ちょっとちがった定義をかいたものがみつかりました．D.D. McCracken によれば

MONITOR: (verb) To control the operation of a routine during execution by a diagnostic routine, often selectmely. Used in debugging.

となっていました．

むかし，電子計算機が，はじめてこの世にあらわれたころ，von Neumann とか，W. Weaver とか，N.

Wiener とか，V. Bush とかでやりとりしていた電子計算機をいかにつくるかについての討論のなかに，電子計算機をつくるなら人間の関与する部分をなるべく除外せよという方針がありました．いま，計算センターで電子計算機をつかっているのをみていると，ひとつひとつのプログラムが自動的にいっているのはたしかですが，そのほかにオペレータという人間がいて，プログラムをかけている風景がみられます．これを人間がやっているかぎり，誤操作があり，まち時間ができることはある程度さけられません．これを，モニタプログラムで，できるだけへらして，効率をあげようというのが，モニタシステムのねらいです．要するに，ふつうのプログラムが，研究室の計算手のかわりをするものなら，モニタは，計算センターのオペレータのかわりをするものといえます．

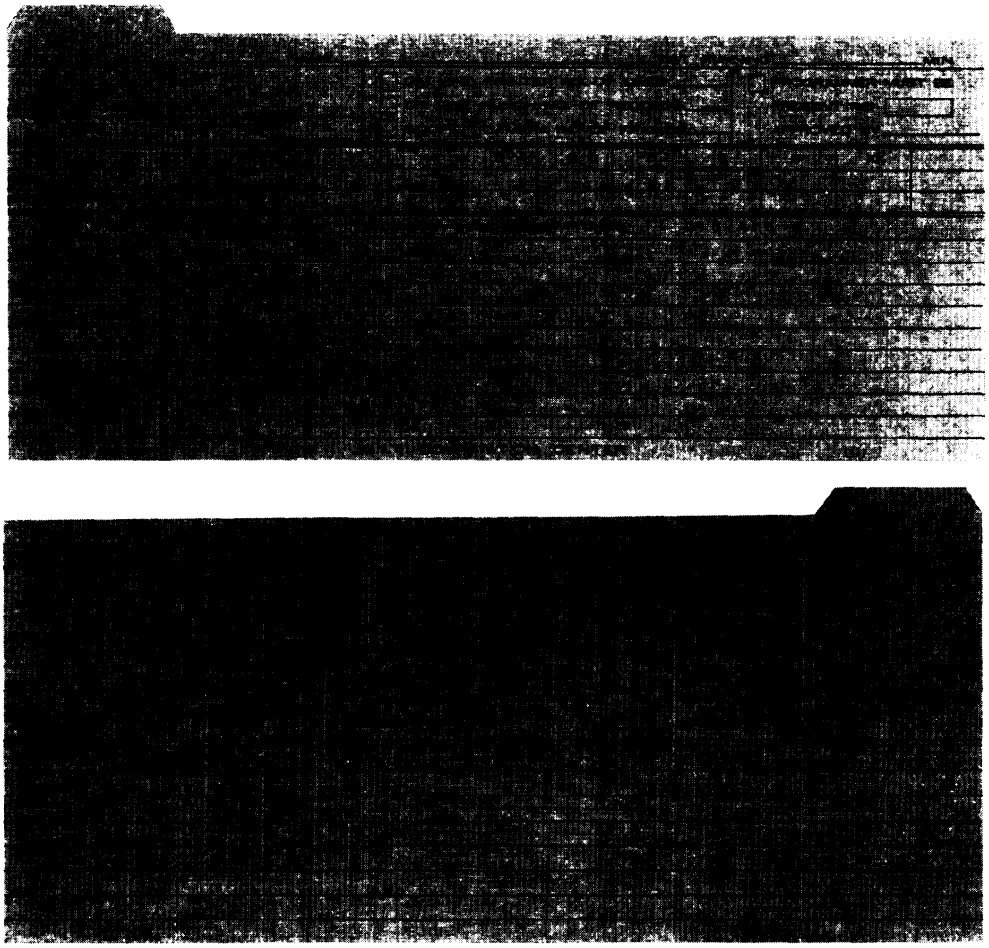
ここにモニタとして，もっとも有名なもののひとつに，IBM の FORTRAN MONITOR があります．このモニタの効能はまったくたいしたもので，FORTRAN でかかれたプログラムのコンパイル，FAP でかかれたプログラムのアセンブル，こうしてコンパイルしたりアセンブルしたプログラムのつづいての実行，計算機の磁気コアに，はいりきらないような巨大なプログラムをリンク（link）と称するブロックにわけ，磁気テープにいれておき，必要なたびに磁気テープから磁気コアによみだす，チェインジョブ（chain job）の実行，さらに，これらがどんな順でどうつづいていてもよく，またひとつのプログラムに，FORTRAN でかかれている部分，FAP でかかれている部分，すでに二進法に変換されている部分がどうはいっていてもよいといわれ，「全知全能」の形容詞がふさわしいようなモニタです．しかしモニタ本来の機能はプログラムがつぎつぎとかけられても，おどろかないという点にあります．コンパイルする，アセンブルする，実行するというのは，かけかたの問題です．

モニタのはたらきは，まえにもかいたように，オペレータにかわることです．だから，プログラマは，オペレータへの指示とおなじことをモニタへ指示してやる必要があります．プログラムの言語，磁気テープのユニット，リールの番号，スイッチのセット状態，予定の所要時間，エラーの場合のダンプの場所，などです．

これらのフォームは，プログラムとデータとが入力機器に理想的にかけられ，理想的にプログラムができていれば，必要最小限の情報でよいわけですが，なにしろこのプログラムがはじまるとき，計算機は，いかなる状態かもわからないので，それでも OK になるようなシステムにしておかなければならないとおもいます．ということはもうすこし具体的にいえばうえに引用した Fortran Monitor では，Control Card は，1行目に＊をうった，I.D. Card と称するもので，はじまっています．こういうものは，プログラムデックの途中に，データとして（本当のデータならプログラムのデータとして，またプログラムならモニタのデータとして）あらわれてはこまります．つまり，そのプログラムがにせの I.D. Card を処理しておわってくれれば問題ないのですが，たとえば，にせ I.D. Card のまえで，プログラムが暴走するとモニタがコントロールをとって，つぎのプログラムにはいろうと，I.D. Card をさがしだすわけです．そのときにせの Card がつかまると，いずれは，原因不明の妙なことになって，おちつくかもしれませんが，同期くずれのようなことにならないともかぎりません．モニタへの情報として予定される入力カードの枚数，磁気テープのブロック数などがあるのは，それをすくう予備手段でもあります．モニタは暴走でもこわれない．入力カードはモニタでよむという考察は，あとでのべます．

モニタプログラムからの出力．これも，モニタシステムが，計算センターのオペレータであるなら，どんなものが，センターからもどってくればよいか，ということをかんがえれば，わかる問題です．計算結果がわたされるのは，プログラム，データのみかえりとしていわずもがな．そのほか冒頭にかいたような，開始と終了の時刻，所要時間(料金(請求書))，コンパイル，アセンブルしてできたプログラムならそのカード枚数，計算の出力を磁気テープにかいたのなら，そのリール番号，途中あるいは最後のダンプの結果，エラーがおきたならその状態(後述のリポートストップ参照)最後にとったリランポイントに関する情報など，枚挙にいとまありません．

うえにのべた入出力の項目は，入出力で対応のつくものは対応ずけるといった考慮ものぞましいことですが，いずれにしろ，計算センターに計算を依頼するときの様式をデザインするのとたいしてちがいありません(**第2図**)．計算依頼のフォームなら，どこかにオペレータがサインする場所があるかもしれません．それもモニタでやらせるなら，出力関係の欄のどこかにモニタネームをつくっておき，モニタが自分のなまえをプリントするのも愛敬があっていいとおもわれます．



第2図 MIT 計算センターの計算依頼カード 1958 年ころ

もっとも，そのセンターにモニタがひとつしかないときいつもおなじなまえでつまらないかもしれません．でもそれはオペレータがひとりしかいないとき，いつもおなじサインしかもらえないのとおなじで，それをさけるには，おなじ程度の能力をもったモニタをたとえばみっつ用意し，それが8時間交代で，モニタしている．どのモニタでもよいプログラムは，いつでもかかりますが，特殊なプログラム，たとえばリストプロセッサを必要とするものは，モニタAでなければならない，そこでモニタAのときにプログラムがかかるようにしておく……とでもしておけばおもしろいかとおもいます．附属病院の外来をみる医局が月曜，火曜，水曜で，それぞれA内科，B内科，C内科とかわるようなもので，一般的なプログラムはどのモニタにもかかるのですが，特にリストはA，コントロール系のシミュレーションはB，というふうに分担したほうが，特殊な問題はプロセスまでに時間がかかりますが，どのモニタも，全レパートリーをおおうのでないため，磁気コアを占有する部分がへって，好結果かもしれません．もっともモニタのいれかえがかんたんならべつに8時間で交代というものでもありません．ところで余談がつづきますが8時間であるにしろないにしろ，モニタをいれかえると，それをコントロールするスーパーモニタがいるという事態にもたちいたります．これもパーキンソンの法則でしょうか．

出力について別の問題をひとつのべますと，いわゆる科学計算ではそう問題にならないのですが，事務計算ではオンラインでつくるハードコピーの用紙を，プ

ログラムごとにかえ，そのたびに計算機がとまるのでは，モニタをつかった意味がなくなってしまいます．磁気テープ装置のように，ラインプリンタがなん台もついているとか，1台のラインプリンタでも充分バッファがあり，用紙をとりかえているあいだはバッファにいれておき，とりかえがすんだら出力するというわりこみにでもなっていればよさそうでもありますが，モニタからオペレータに対する出力もこのチャネルからでてくるのであれば，やはり出力まで時間がかかるのはいただきかねます．とにかく，どれも標準の用紙に出力されるようにこころがけるべきでしょう．

ところでモニタシステムは，ハードウェア（「かなもの」）に対して，ソフトウェア（「かみもの」とよぶのはどうでしょうか）といわれています．そこで，最初に引用した IBM の用語集で，もういちどソフトウェアの項をみてみますと，

SOFTWARE: A colloquial term for any program or method of use which can perform hardware funnctions.

とあります．つまりプログラムならなんでもソフトウェアかといえば，そうでなくて，かなものの機能をはたすもの，であるわけで，モニタの機能がかなものにくみこまれてはいない．だからモニタはかなもの機能をはたすものではないというロジックをつかえばモニタシステムはソフトウェアでなくなってしまいます．念のためハードウェアになんとかいてあるかというと

HARDWARE: The mechanical, magnetic, electrical and electronic devices or components of a computer.

とあります．

うえのソフトウェアの解釈を，もっと善意におこなえば，プログラムをつくるひとが，この機能まではかなものだとおもう，そういう範囲にはいるプログラムのことです．だから，計算機の解説書で，モニタ，コンパイラ，アセンブラなどの説明をするにあたって，これはプログラムだということをことさらのべる必要はないわけです．もっともそれは，ソフトウェアが，かなものの機能を充分にそなえているときのはなしです．

かなものの特長，そのひとつは，ユーザがどんなプログラムをつくっても，絶対にこわれてはならないということです．俗にやまいは気からっといいますが，機械のダウンがプログラムからではこまります．したがって，かなものとみなされるソフトウェアは，プログラムがどんなに暴走しようとも，決してこわされないようにできていなければならず，またこうなっていれば，いちいちプログラムだとりきむ必要がなくなります．ところが，暴走の次第によっては，ソフトウェアがあえない最期をとげるというのであれば，解説書なりなんなりで，これこれの部分はプログラムであるということを力説しておかなければなりません．このどちらがいいかといえば，前者の方に軍配があがるわけで，一天万乗のモニタは，プログラムがどう動作しようと，つぎつぎとプログラムをこなしていく権限をもたされていることが大切です．それなら，もっと基本的な問題として，モニタをかなものにくみこむのはどうかという点になると，それはつまらないことだといわなければなりません．モニタがはたらくのは，インタープリティブなモニタでなければ，デューティレシオはわずかなもの，わずかであるほどのぞましいもので，それなら，万能計算機をプログラムでモニタにしてしまうのがどうかんがえても得策です．

こういうモニタのプログラムは，メモリにあれどめ（memory protection）をかけて磁気コアにいれておくことが必要です．リードアウトメモリでもいいようですが，それにしても，モニタのつかっている作業番地は，あれどめでまもってやる必要があります．この点くわしくはあとであげつらうつもりです．

かなものの特長，もうひとつは，パラレルオペレーションというか，計算機のどこかでおこった必要な事態が，間髪をいれずわかるようにデザインされるということです．たとえばわりこみ機能はプログラムの進行のあいだ，たえず入出力機器を監視し，行動をおこすときがくれば，あたうるかぎりはやい時刻に，必要な処置をとります．これとおなじような機能をモニタなど，ソフトウェアにもたせるなら，たとえば，あるプログラムが，まもられているメモリになにかかこうとしたようなとき，すぐにわりこみで，モニタにコントロールをかえし，必要な処理をして，ふたたびまえのプログラムにかえるなりなんなりするということになります．つまり，かなものが平常，触手をのばしているような状態は，ソフトウェアでは，探知されればわりこみで，それに関係したプログラムにコントロールをわたすことになります．

この辺で，ソフトウェアが健全であるためには，かなものは，どの程度健全であればよいかということにひとつの試案をのべてみましょう．

Mens sena in corpore sano!

かなものが，ソフトウェアのことをなにもかんがえていない場合，ただひとつのてはすべてをインタープリートすることだとおもいます．これなら，わりこみの機能もかなものになくてすみます．しかしこれでは時間がかかって大変ですから，代案としてソフトウェアに必要なオーダーだけが，エキストラコード（ソフトウェアオーダーというひともあります．）になるようにしておくというてがかんがえられます．メモリをこわされる心配がある場合，ストア関係の命令は，全部エキストラコードでわりだしてまず番地をしらべ，合格すれば本当にストアするというものです．メモリのあれどめとまったく同様にかんがえなければならないのは，磁気ドラムへのかきこみ，磁気テープへの出力で，これらの命令も，一応エキストラコードで，テストをうけるということもすぐにみちびかれてきます．このかんがえがすでに実際の計算機でつかわれていることは，ご存知のとうりです．すなわち，IBM 7090にあるトランスファトラップで，トランスファの命令は，ひとつの種類をのぞいてすべてわりだします．そののぞかれるひとつはトラップトランスファで，
Trap Transfer is immune to the trapping mode
というわけです．

されば，うえのストアなども，テストで合格した場合本当にストアするため，トラップストアとかいう命令がいることになります．ほかの命令についてもみんな，トラップなんとか…が必要になります．ここでかんがえなければならないのは，こうい命令をいちいちつくっていたのでは，1．特殊な命令の種類が激増するということ，2．プログラムが暴走して，トラップなんとかの命令が実行されると，運のわるいときは，モニターが殉職してしまう場合もおこりうるということです．

このひとつの解決はつぎのシステムデザインです．このデザインでは計算機の状態をまず大別して，「一般の場合」と「特殊の場合」とのふたつにします．ふつうのプログラムがうごいているときは，一般の場合に相当するのですが，わりこみ，わりだしなどによって，特定の番地にジャンプすることにより，制御のプログラムへくると特殊の場合になります．この制御プログラムがすんで，さっき不意にとびこんできた番地にかえると，計算機はふたたび一般の状態にもどります．このふたつの状態では，いろいろなことがちがったふうにおこなわれます．きがついたものだけを，かきならべてみると，つぎのようになります．

	一般の場合	特殊の場合
わりこみ	可	禁　止
ストア	わりだし	実　行
ジャンプ	わりだし	実　行
エキストラコードのセットリセット	わりだし	実　行

これらは，たいして説明もいらないでしょうが，わりこみに関しては，わりこみがいちどおきるとそれを処理しているあいだ，別のわりこみがおこって，リターンアドレスをこわすのを防止するため，わりこみ禁止がかかるのが常識になっています．これを解除するのは，通常，解除の命令があって，わりこみ制御のプログラムからもどるときにそれを実行するのですが，禁止が自動的にかかるのに，解除は命令でやらなければならないのは，気がききません．PC-1 の例では，わりこみがおきると，510 番地のアドレス部につぎの命令の番地をいれ，511 番地へとんでくるのですが，511 番地の命令実行が禁止，510 番地の命令実行（つまりジャンプ）が解除となっているとうまくいくような気がします．

ストアやジャンプは，ふつうのプログラムではすぐトラップされ，これが PC-1 なら 511 番地にとぶという想定です．それから，トラップの原因をしらべ，ストアならストアの制御プログラムにふたたびジャンプし（ここはトランスファトラップがきかないので，ジャンプできる），番地のテストをし，目的のストア（これもトラップなし）をして 510 番地からかえります．このジャンプまでトラップなしです．

こういうふうに，命令がエキストラコードになっているとよいものの例をもうひとつあげると，それは停止命令です．Fortran Monitor などよみますと，モニタされるプログラムは最後に停止命令をつけずに CALL EXIT（すなわち，サブルーチンにジャンプする）をつかえとありますが，うえの流儀では停止をかいてよく，一般の停止ではいつもわりだしがおこり，モニタによりプログラムのおわったことが検知されます．これだけでは，本当に計算機をとめてしまうことができませんから，特殊の場合の停止命令は停止するというふうにしておきます．

こうかんがえてくると，うまくいきそうですが，さしあたって，ふたつの問題があります．ひとつは，ストア関係は，エキストラコードにするよりも，かなものくみこみにしたほうが効率がよいのではないかということと，もうひとつは，特殊の場合の制御プログラ

ムをいれるまえは，特殊の状態においておかなければ，なにもすることができないということとです．

第1の問題は，ストアは，プログラムのなかでも頻度のおおきい命令で，1）これをエキストラコードにすると，速度の減退がいちじるしいであろう．2）ブロックトランスファや磁気テープのリードのように，ひとつの命令で連続してストアするとき，その全領域にわたって，検査するのもたいへんである（もしかしたら不可能かもしれません）ということなどにもとづきます．

第2の問題は，プログラムロードもできないような自縄自縛のコンピューターをデザインしないようにという簡単ないましめにすぎません．

かなもので，メモリのあれどめをするとなると，その番地の範囲をかなものにおしえる命令が当然いるわけです．これは，ストアの命令で兼用することもできますが，いずれにしても，ここで気をつけるのは，一般のプログラムが，あれどめのセットリセットをやってはこまるということです．ここで一歩をゆずったら，橋頭堡は，どんどんくずされてしまいます．IBM 7040 にはあれどめに関する命令がありますが，これはトラップをおこします．要するにあれどめのセットリセットも，特殊の場合にはできて，一般の場合にはできない，もしセットリセットがこころみられたら，それは暴走だとみなすことにします．

命令をエキストラにしたりしなかったりも，実は命令でできるとおもしろいとおもわれます．この点はよくかんがえないと，矛盾をおこすかもしれません．しかしここでいいたいことは，そういうエキストラのきりかえ命令も，一般の場合にはおこなわれてはならないということです．

それから，異状がおきたことをすぐモニターが探知するために必要なわりこみは，わりこみのない計算機に一応そろっているセンスの命令，たとえば，トランスファオーヴァーフロー，トランスファチャネルオンオペレーション，トランスファインディケーターオンなど，これらの命令のジャンプの条件がそろったときにすぐわりこむようにしておけば，充分だとおもいます．

要するに，いまのべたのはプログラマ生悪説にもとづくものであり，モニタは完璧であって特殊の場合にはたらき，なんでもやってよく，ふつうのプログラムは，一般の場合にはたらき，かぎられたことしかできません．特にモニタをおかすことは，絶対に禁止されているという思想が，解決篇のねらいです．

モニタの安全が保障されたら，つぎにモニタはどんなふうにはたらくべきかをかんがえてみます．これも，計算センターのオペレータがどんな作業をしているかの分析することにほかなりません．そしてかんがえたモニタのひとつの例をご覧にいれましょう．

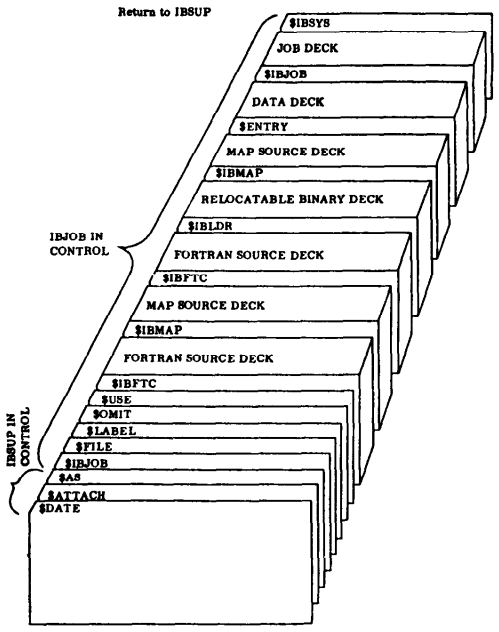
まずオペレータは，依頼されたプログラムやデータとそれに附された依頼用紙の記述から，プログラムをいれる方法，いっしょにいれておくべきライブラリルーチンをしります．たとえばコンパイラをつかっていれるならコンパイラをロードし，ライブラリルーチンをいれますが，モニタもこれとおなじことをします．まずコントロール部分（これがモニタへの入力です）を，特定の入力チャネルからよみこみ，コンパイラをつかうことをしり，コンパイラをロードし，コンパイラに，プログラムをよむチャネルをおしえて，コントロールをわたします．コンパイラはコンパイルしたものを磁気テープにいれ，同時に必要なサブルーチンのリストを用意します．これがすむと，コントロールはモニタにかえり，モニタはローダをロードします．ローダは必要なルーチンをすっかりそろえて，プログラムの実行をはじめます．いまのかんがえでは，ローダもあれどめをかけられて磁気コア内にとどまります．それは，主プログラムが磁気コアにはいらない場合，こまぎれにし，つぎつぎとロードする必要があるためと，プログラムがすんだとき，コントロールはいったんローダを経由して，モニタにかえるのがスマートだとおもっているからです．

無事に計算がおわれば，コントロールはモニタにかえり，モニタは，所要的間，出力の量などの情報をプリントして，つぎのプログラムにうつるという次第です．

どのプログラムも無事にいけば，はなしは簡単ですが，そういかないプログラムがあるのがオペレータのなきどころ．電子計算機でひとが計算しているのをはたからみているとき，くびをかしげ，うでをくみ，かんがえこんでいる図は決してめずらしくありません．でてきた結果が数値的な期待はずれなら，オペレータは，どうすることもできず，これはモニタにとっても関係ないことですが，とまってしまった．ループにはいったきりで，でてこなくなったという種類のは，だめなことがわかっていながら，つぎのプログラムにすすめないというこまった問題です．このこまりもののふたつのうち，とまってくれるほうは，オペレータにも

処理しやすいように，モニタにも簡単で，なにしろとまればわりこみがかかるという計算機を仮定していましたから，モニタはコントロールをえて，とまった原因をしらべます．そして，デバッグに必要な情報をできるだけうちだして，そのプログラムは中止ということになります．うちだす情報はコントロールカウンタ，命令レジスタ，アキュムレータ，インデックスレジスタの内容などで，アキュムレータや，サブルーチンリンクアドレスのレジスタが，プッシュダウンメモリになっていれば，そういったものも必要でしょう．このリポートストップは，別にモニタがなくても，PC-2 のようにオーバーフロー，ノーコードでわりこみ，わりだしをおこす計算機では，わりこみ制御のルーチンに，リポートストップの機能をつけくわえておくと，デバッグで失敗してひきさがるとき，情報のとりわすれでくやしがることがずっとすくなくなり，また交代の時間も減少する効果をもっているものです．

一方，ループにはいってでてこなくなるのは，始末のわるいもので，どう処置したらよいか，あまり決定版がありません．しかしかんがえられる手段のひとつはつぎのようなものです．計算機には時計がついているものとします．冒頭のべたように，おかねをとる都合でも時計は必要なものですが，ここでいう時計は，ただよみだせるだけでなく，一定の時間間隔でわりこみをおこすことができるようなものです．これがあれば，モニタは，まず開始の時刻をよみ，コントロール部分から予定の所要的間をよみ，終了の予定時刻をきめます．プログラムはこれからスタートするわけですが，時計のわりこみで，コントロールはときどきモニタにもどります．モニタは，時計のわりこみでコントロールをもらったときは，時計から時刻をよみ終了予定時刻になっていないことをたしかめます．しかし，予定時刻をすぎたことをしれば，職務訊問の用意にかかります．つまりこのあとは，条件つきジャンプがあるたびに，自分でつかえる磁気テープ（システムテープ）に，その命令の番地，アキュムレータテストならアキュムレータの内容，インデックステストならインデックスの内容をだしておき，適当な時間だけそれをおこなうとプログラムを中止し，磁気テープのデータをハードコピーにしてだします．この資料からプログラマはどのループで循環しているかがわかります．たとえば所要的間 50 分のプログラムが，はじめのちいさなループからでなくなって，50 分もむだなループをまわることもこれではあるわけですが，そういうのはしかたがないとおもいます．



第 3 図　IBM 7040 IBSYS のいろいろなコントロールカードとデック

このようにループは時計からわかりますが，またべつに，入力データのほうからわかることもあります．たとえば，データがなくなって，つぎのプログラムがはじまったり，情報のないところまでテープをよむようなことになったり，はじめに予定した入力データの数量をはるかにオーヴァーしたりということがきっかけです．これらからプログラムのミスオペレーションをしらべるためには，入出力が，かなり複雑なコントロールになり，プログラマーのだれもがつくれるというものではありません．標準の IOCS が必要になるゆえんです．

かくして，ひとつのプログラムはおわりました．ここでモニタは，やすむ間もなくつぎのプログラムをはじめなければなりません．それができるためには，つぎのプログラムの入出がすっかり用意できていることが肝心です．ひとつのプログラムをやっているうちにつぎのを用意するのですから，入出力用機器はひとつでつかう予定の倍のものが必要になります．もっとも出力側は，紙テープやロール紙などひきちぎれるものだと，ひとくみ分でよいことにもなります．この辺の数についてのこまかい議論は，はぶくことにして，う

えのように，つぎのプログラムの準備をやっておく場合に問題になる点をすこしのべておきます．

それは，うえの方法で連続運転をした場合，プログラムやデータがどのユニットにかけられるかが，あらかじめはわからない．したがってプログラムがかけないということに対する処置です．もっとも柔軟な方法は，データにシンボルで名前をつけ，入出力命令はユニットでよばずに，シンボルでよぶものです．つまりCOBOLのファイルネームの流儀です．入出力の命令はエキストラコードになっていて，モニタがコントロールをとり，ファイルが実際にかかっているユニットとの対応表をみて，目的のリード，ライトの命令をだすというものです．ここに注意すべきことがふたつあります．1）入出力はかならずソフトウエアでやるというのは，IBM 7090のIOCSなどの思想で，それにはサブルーチンジャンプでリードあるいはライトの番地へとぶというふうになっているわけですが，もちろんプログラマはそれとは無関係に入出力の命令がだせます．リード，ライトのサブルーチンが，バッファリングなどやっているのに，プログラムがリード，ライトを自分でやれば，こわしてしまうこと，必定ですが，入出力命令をエキストラコードにしてしまえば，サブルーチンジャンプなどということをしなくても，プログラマのかいた入出力命令はかならずつかまり，しかも特殊の場合には，おなじ入出力の命令で，機器をうごかすことができるということです．出力をがっちりとモニタがおさえるということは，このほかリランポイントをつくるうえにも，また，暴走して，ほかの磁気テープになにかかく，あるいはかかないまでもうごかしてしまうのをふせぐ（メモリ保護とおなじ心配です）にもどうしても必要です．

2）ファイルのリスト，つまりどのファイルがどのユニットにかかっているというリストは，オペレータが磁気テープなどをユニットにかけ，ファイル1はユニット1に，ファイル2はユニット2にかけたというような紙テープをつくっていれてやるか，タイプライターでたたきこんでやればよいわけですが，オペレータがたたいているのでは時間がかかり，タイプミスなどもかんがえられます．むしろ，ひとつのプログラムのおわらないうちに，モニタはつぎのプログラムのコントロール部分をよみ，必要なファイル名をしって，あいているユニットをアサインし，それをタイプライタにだして，オペレータにそのとおりかけさせる．オペレータはかけたということだけをモニタにしらせるというほうがすぐれているようです．

磁気テープユニットとにたようなのが，センススイッチで，たとえばあるプログラム（Aとよびます）はセンススイッチ 1, 2, 3 をつかっていて，つぎのプログラム（Bとよびます）もセンススイッチ 1, 2, 3 をつかうつもりでいるという場合，しかも，セットのしかたがちがえば，プログラムのきりかわったときに，大至急でセットしなおさなければなりません．ここで計算機はまたとまるわけです．センススイッチをふやすよりも，とまったほうがいいといってしまえばそれまでですが，とめたくなかったら，センススイッチ 4,5, 6 を，Bの 1, 2, 3 にみたててセットしておき，Bのときは，1,2,3 がそれぞれ 4, 5, 6 に解釈されるよう対応のリストを用意します．そうすれば，プログラムは連続してAからBへすすみます．センススイッチを，シンボリックネームでよぶこともちろん可能です．

はなしをふたたびファイルのシンボリックネームにもどしますと，オペレータが，モニタの指定どおりにファイルのリールをかけたつもりでも，まちがったものをかけているおそれがあり（ファイルネーム，リールの順のちがった入力テープ，まだけしてはいけない出力テープなど），それをチェックする意味で，IOCSにはふつうラベルチェックという重要な機能をもたせています．IOCSは，ファイルがオープンされたときやひとつのリールからつぎのリールへきりかわったとき，ラベルチェックをやっていますが，そこでひっかかったら，オペレータがかけなおすあいだ，時間のロスがでます．これをふせぐには，テープがマウントされたら，モニタはひまなときにこれらのチェックをやってしまう．よければいつでもつかえる状態で，オペレータにはずさせない（プログラムがなにかのはずみで暴走して，うちきりのときははずしてもよいのです）．だめなら即刻かけかえの命令をだすということをやります．これで，うえで心配したロスはぐっとへってくるでしょう．

このようにモニタにさきまわり制御をさせると時間が節約になるもうひとつの例は，まえにもでてきたリランです．これは，よくいわれているのは，出力テープのきりかわるたびにリランの情報をだしておくものですが，このとき，入力テープや，ほかの出力テープは決してはじめのところに位置していないのですから，あるリランの点からやりなおすとき，その場所までテープをすすめてからはじめなければなりません．ですから，つぎのプログラムが，リランであるというとき

は，まえのプログラムの実行中に，つぎのリランの情報をできるだけよみ，必要なテープをオペレータにマウントさせマウントができたら，テープのチェック（ここまではまえの項とおなじ）それからリランがはじまるところまでテープをすすめておきます．ここまでがモニタのさきまわりの作業です．

ここまでかんがえてきたモニタは，計算センターのオペレータのシミュレータみたいなものでした．つまり計算センターは，つぎつぎとプログラムをかたづけていくというモデルのわけです．プログラムはこういうふうに，たてにつながると表現すべきかどうかわかりませんが，一度にひとつずつ処理していくのが初歩的なオペレーションですが，また別に，メモリと入出力装置に余裕があれば，バッファリングだけではさばけないような入出力機器のまち時間を，別のプログラムの処理につかうという，多重処理もあるわけで，ふつうこれは，わりこみ制御のプログラムがつかさどっています．これは，よこにプログラムを処理するモニタとみることもでき，メモリのあれどめその他の機能が有用なことは論をまちません．IBM 7070 にある SPOOL のプログラムもその一例ですが，これは，カードから磁気テープ，磁気テープからカードの両方向のおそいプログラムと，主要なプログラムの3種類をコントロールするもので，どの種類もプログラムのはいる場所はきまっていて，その種類のなかでは，たてにプログラムを処理していくものです．この3種類は IBM の最近のセンターでは，磁気テープベースの 7090 1台と磁気テープとユニットレコードベースの 1401 2台とのくみあわせがよくみられるようですが，その3台の作業を7070 1台でまにあわせ，SPOOL というモニタみたいなものが時分割で，3台の作業をきりかえているとみられるものです．SPOOL は Simultaneous Peripheral Operation On Line の略です．

多重処理も，もうすこしプログラムのおおきさ，はたらきを自由化すると，モニタはたてにつなぐのがむづかしくなります．たとえば，A, B, C, D のプログラムをモニタは別にして 6000 語のメモリの計算機にかけるとします．また入出力機器関係はうまくいくとします．それからうえのよっつのプログラムの占有する場所はそれぞれ 1000, 3000, 2000, 3000 語としましょう．これはプログラムに先行するコントロール部分にかいてあってモニタにはプログラムをロードするまえにわかります．まず A, B, C はこの順に 6000 語のメモリに無事におさまり多重処理がはじまりました．そのうち，AとCとがおわったとします．どちらかが当然さきにおわるわけですが，1000 語あいても 2000 語あいてもそれだけでは3000語のDプログラムをロードすることはできません．両方おわれば，3000 語のあき地ができたので，勘定のうえではDがはじめられることになりますが，なにしろまだBはメモリのどまんなかに陣どっているので，どうしょうもありません．これをまえのほうに移動させれば，Dがはいるというので，UNIVAC の CHIEF というモニタ（エクセキューティブルーチンというのがただしいそうです）はこの難事業をやっているとか．いつぞや説明をきいたのですが，どうしてうまくやっているのか，まだ充分納得できていません．リロケータブルバイナリというような表現のプログラムデックがつかわれることがあります．どこにロードされるかわからないので，リロケータブルになっているのですが，ロードされるとき，リロケーションビットがあれば，アドレス部にロードされる番地がたされるというものです．プログラムをロードしてもこのリロケーションビットをまとめてどこかちかくにストアしておけば，メモリのなかでプログラムを移動させるとき，このビットのあるなしで，ロケーションの差をたすかひくかすればよさそうなのですが，移動がはじまったとき，アキュムレータやインデックスレジスタにはいっている数もリロケータブルかどうかがわかっていなければへんなことになります．たとえば，インデックスがサブルーチンジャンプにつかわれたなら，そしてサブルーチン実行中に移動がはじまったなら，インデックスも補正しないと，「かえってみればこはいかに……」になってしまいますし，回数カウントのインデックスなら補正はやってはなりません．絶対番地のプログラムはないにしても，アドレスの計算でこったプログラムがはいってこないとはかぎりません．リロケータブルアドレスの和などというコンスタントには，リロケータブルビットをふたつつけて，倍の補正をしてやらねばならず，そんなことができるかとおもうのです．

これはまた余談になってしまいましたが，多重処理と別のかたちのモニタは，SOS といわれる SHARE モニタで，これは，またかわったアイデアでつくられているようです．きくところによると，SHARE モニタでは，入力のデータ，出力のデータが，10 進その他であって，SHARE のシステムのサブルーチンで変換しなければならないようなとき，どのプログラムもその部分を主プログラムの前後にまとめてつけておき

ます．主プログラムをE，入力側の変換をI，出力側の変換をOであらわすと，I-E-O というかたちにプログラムがかけます．システムサブルーチンによる変換がなければ，E-O とか I-E または E だけというプログラムもあることになります．いま，よっつのプログラム A(I-E-O)，B(E-O)，C(I-E)，D(E) が，それぞれかっこのなかのかたちで，これをつづけてランさせようと，まとめてモニタにかけると，モニタは入力側の変換ルーチンをロードし，A の I，C の I をやってしまいます．つぎに，A，B，C，D の E を順におこない，最後にAとBのOの変換をやるという次第です．なつやすみの宿題帳，1日分ずつかたづけるのがふつうのモニタなら，SHARE モニタは，わたされたぶんだけ，まず国語を最後までやり，つぎに社会を最後までやり，それから日記をはじめからかくというやりかたにたとえることができるでしょう．

モニタはいずれにしても，オペレーションがややこしくなればなるほど必要になるもので，これからはいくつかの計算機が電線でつながったような場合，その社会を円満にコントロールする作業もつけくわわってくることでしょう．なかの1台の計算機がモニタコンピュータなどということになって，全体を支配するようになるのかもしれません．

モニタはハードウェア，かなものとおもえとかきましたが，なかには，ソフトウェアであることがわすれられない面もあります．それは，変更することがかなものより容易だということです．計算機ができた，モニタができた，それがくばられて，世界中でつかわれるようになった．そうなってから，すこし変更したいということになりました．かなものだと，カストマーエンジニアに指令して，修理しなければならないわけですが，ソフトウェアならカードかテープをたくさんコピーしておくればそれで OK です．かなものでも，プラグインのシャシー内だけの変更なら，シャシーをたくさんつくって，それをさしかえろといえば簡単かもしれません．ソフトウェアの変更が，かなものより簡単でも，あたまのいたいことに，われわれは，ソフトウェアのはいっているところはメモリのあれどめができていて，そこにはそとからストアできないということにしてしまっています．この禁をとくのに，かなもののキイか，あるいは，アプライドプログラミングチームだけがしっている呪文かを用意しなければならないでしょう．IBM 7090 のモニタなどは，たいてい，モニタの改良ができるようになっていて，それをおこなう部分がモニタにビルトインされています．システムプログラムはにっちもさっちもいかないようになっては困りそうなので，なんとかモニタを修正するエレガントな方法をかんがえておかなければなりません．Orchard-hays は IRE 誌上で Dual Master Operator[3] という方法があることをすでに指摘していますが，これについては，ぼくはそこでよんだだけで，魅力は感じているものの，詳細不明．ご存知のかたからご教示をえたいとおもっています．

最後に，これからモニタ，あるいはシステムプログラムをつくろうというかたにもうしあげたいことがあります．それはまず人間工学的な配慮を重要視していただきたいということです．もっと簡単にいえば，つかいやすいかつかいにくいか，おぼえやすいかおぼえにくいかということが，システムプログラムを成功させるかさせないかということになるでしょう．ものすごいマンパワーをかけてせっかくつくりあげたシステムプログラムが，あまりつかわれないというのでは残念です．こんなにややこしいことをおぼえなければならないなら，モニタなんかはずしてゆっくりやろうというセンターだってでてくるにちがいありません．どうかその点を慎重にかんがえて，自動車の運転台でも設計するような調子でやっていただきたいとおもいます．第2の点は慎重にということとは矛盾しそうですが，システムプログラムはなるべくはやめにつくることが大切だということです．あとからできたシステムプログラムはよほど便利でないかぎり活用されないようです．これも人間工学の問題にかえるかもしれません．いずれにしても，よいシステムプログラムをつくるため万国のプログラマーの団結がせつにのぞまれます．

参考文献

1) IBM Reference Manual: Glossary for Information Processing (form C 20-8089)
2) 高橋 茂: 並列プログラミングと割込み，プログラム懇談会資料 1962-2-20.
3) W. Orchard-hays: The Evolution of Programming System" Proc. IRE Vol. 49, pp. 283-295 Janaury 1961