43－S006

会話モードコマンドの開発

昭 和 44 年 6 月

財団法人　日本情報処理開発センター

序に代えて

電子計算機による情報処理は，社会・経済の発展にともない，各種情報の蓄積・加工・供給を最も有機的，効果的に進める担い手として最近とくにその役割の重要性が認識されております。

また，情報処理そのものも，第三世代電子計算機の登場以来，その利用分野の拡大とともに経営の意志決定システム，電子計算機の不特定多数による共同利用といった高度化の方向が検討されつつあり，従来の事後処理的な利用からみると，現在の情報処理は大きな発展期を迎えているともいえます。

このような情勢において情報処理および情報処理産業の前途には解決を要する幾多の課題があります。すなわち電子計算機を人間の知的作業の協力手段として広く活用するための有用性が認識されるとともに，その利用は産業・社会のあらゆる分野で拡大の一途を辿っておりますが，これまでの電子計算機システムでは，いわば間接的な利用法を採らざるをえないという面倒さがあり，利用者が電子計算機を意のままに使用するという面に大きな難点もあります。そこでできるだけ利用者が電子計算機システムに直結して，電子計算機を直接操作できる電子計算機システムの確立が強く望まれております。

このような電子計算機システムを実現させるためには，遠隔情報処理技術やタイム・シエアリング技術といった各種問題の解決をはかるとともに，効果的に電子計算機を利用するための会話モードコマンドの開発が必要であります。

当財団は，電子計算機利用に関する諸問題を解決するため各種の調査・研究事業を実施しておりますが，この報告書はその一環として実施した会話モードコマンドに関する調査・研究と会話形プロセッサ開発の結果をとりまとめたものであります。

なお，この事業は日本自転車振興会の機械工業振興資金による「昭和４３年度情報処理に関する調査研究補助事業」の一部として実施したものであります。

ここに，この研究実施にご尽力ならびにご支援を賜わった上滝致孝，榊原清，矢田光治，土居範久，原田賢一の氏名ならびに関係各位に心より感謝の意を表わしますとともに，この報告書が各方面に利用され，わが国情報処理産業発展の一助として寄与できますよう念願いたす次第であります。

昭和４４年６月

財団法人　日本情報処理開発センター

会長　　難波楷吾

目　　次

まえがき

プログラミングという作業は使用するプログラミング言語の種類によって多少異なるが，電子計算機が頻繁に使用でき，さらに電子計算機にかけた結果ができるだけはやく手元に戻ることが望ましい。プログラムを組んでいる段階で，電子計算機にかけてみた結果がなかなか手元に戻らないような状態だと，非常に能率が悪い。とにかく結果が出さえすればよいという類のプログラムでも時日がたつと，案外プログラムの中味，すなわち作り方を忘れるものである。問題が複雑になれば，ますますわからなくなる。凝った作り方をすれば，何が何だか，いったいどうやっていたのか忘れてしまうことさえある。一般に現行のバッチ（一括）処理方式では，一旦だすと結果が戻ってくるまでに数時間から数日かかるのが普通である。

また，プログラムをコーディング用紙に書くとき，カードあるいは紙テープに穿孔するとき，さらに専門のプログラマに依頼したときには意思に疎通を欠くこともあるといった具合に誤りが発生しそうな場所は随所にある。

とにかくプログラムが組めるような問題を相手にしている場合にはまだバッチ処理方式でも我慢できるが，試行錯誤しながら問題を定義していくような場合には，バッチ処理ではお手上げである。プログラムができても，前の結果にもとづいてパラメータを少しずつ変えていかなければならないようなものも困る。

この他にもいろいろと欠点があるが，その源にさかのぼれば，電子計算機と人間との間が離れすぎているということになる。この間をもっとつめれば，さらに人間の知的作業の有益な道具として電子計算機を使用することができる。そこで，人間と電子計算機の共生といった概念が生まれた。共生（symbiosis）という言葉は，ここでは単に"機械と人間がお互いに害をおよぼさない"という意味ではなく，"機械と人間が一緒にいることによって人間が利益を受ける"という積極的な意味に使用している。

1. 会話形プロセッサ概説

1. 会話形プロセッサ概説

会話形プロセッサとは，人間と電子計算機との交互作業をより強力にすることを目的とし，人間が電子計算機とあたかも会話をとりかわしながらプログラムを作成し実行させることができるプロセッサである。

プロセッサをどう定義するかによって，会話形プロセッサの定義も自づと異なるが，ここでは厳密な定義はしないで，プロセッサは，いわゆる言語プロセッサを指すものとする。

使用者の立場に立つと，会話形プロセッサの言語体系はコマンドと一般のステートメントに大別される。コマンドとはシステムを制御したり，プログラムに関する種々の操作をシステムに要求したりするのに用いる命令である。一般のステートメントとはバッチ処理方式の言語プロセッサを使用したときのプログラミング言語と同じものである。ただし，会話形であるために端末から操作しやすいようになっている。システムによっては，これらステートメントを入力すると直ちに処理して折り返しその結果を端末に送ってくるようにしてあるものもある。

言語プロセッサで，プログラムを処理する際に，まずしなければならないことは，プログラムがその言語の文法に従っているかどうかを調べることである。会話形プロセッサでは，この文法チェックを行なう方式は，使用者の立場に立つと2種に大別できる。すなわち，使用者がステートメントを入力する度に文法チェックを行ない，誤りを検出すると折り返しその旨を使用者に知らせる方式と，バッチ処理の場合と同様，使用者がプログラムを作り上げるまでに何もしないで，使用者がコンパイルとか実行とかの指示をするとはじめて文法チェックを開始し，誤りを検出するとその都度端末に知らせる方式がある。前者の場合には，使用者は，システムが前のステートメントを処理し，次のステートメントの入力を許可するまで，次のステートメントを入力することができないが，入力したステートメントは直ちに処理されるので，誤りがあった場合にはすぐ訂正することができるという大きな利点がある。後者の場合には，使用者は自分に適した速度でプログラムを入力することができるが，プログラムに文法上の誤りがあるときにはバ

ッチ処理の場合と同様のエラー・メッセージを受けとることになる。ただし，すでに入力してあるプログラムはシステムに保存されているので間違っているものだけを修正しさえすればよい。後者は，システム全体の効率に重きを置いた方式であるとみなすことができる。会話形という言葉からすれば，プログラミング時には前者の方がこの言葉にピッタリしている。

実行時の処理方式とか機能は，システムによって異なるが，一般にプログラムを走らせてみてはプログラムを修正していくといったことや，プログラムで使用している変数等の現在値などを調べたり，初期値を変えてみたりすることは容易にできるようになっている。これらは，どのようなコマンドが用意されているかによって決まる。

会話形プロセッサが使用できるシステムでは，作成したプログラムとか作成しつつあるプログラム，あるいは入力データや処理結果などは，ファイルとして，システムに保存できるようになっているのが普通である。一般に，会話形プロセッサが存在する環境はタイムシエアリング・システムであると考えてよいことから，この他にも様々な機能が用意されていると考えてもさしつかえない。

それでは，会話形プロセッサとはどのようなものかをＫＥＩＯ－ＴＯＳＢＡＣタイムシエアリング・システムのKEIO システム（会話形ＦＯＲＴＲＡＮ）の使用例を用いて説明しよう。（下線部がシステムからの応答である。）

```
HELLO
user name ... N.DOI
system ... KEIO
new or old ... NEW
program name ... TABLE
ready                19:21          1.27.68
*PROGRAM
  10 ready C     TABLE (GAMMA, ERROR, LOGE)
  20 ready       DO 10 I=1, 10
  30 ready       A=FLOAT (I)/100.
  40 ready       B=GAMMA (A)
  50 ready       C=erf (A)
  60 ready       D=ALOG (A)
  70 ready    10 PRINT 100, A, BC#, C, D
  80 ready   100 FORMAT (' ', 4F20.10,)
error no.   21
  80 ready   100 FORMAT (' ', 4F20.10)
  90 ready       STOP
 100 ready       END
 110 ready *COMMAND
ready
*START                                                          ⑨
          0.0100000000          99.4325851202          0.0056416638

          0.0200000000          49.4422101636          0.0112822098

ready                                                          ⑩
*LIST                                                          ⑪
  10 C     TABLE (GAMMA, ERROR, LOGE)
```

```
   20         DO  10  I=1, 10, 1
   30         A=FLOAT (I)/100.
   40         B=GAMMA (A)
   50         C=ERF (A)
   60         D=ALOG (A)
   70      10 PRINT  100, A, B, C, D
   80   100  FORMAT  (' ',4F20.10)
   90         STOP
  100         END
end of list

ready
*ALTER  80, 80  ——————————————————————————————⑫
   80 ready   100  FORMAT (' ', 4F12.6)
   81  ready*COMMAND
ready
*START  10  ——————————————————————————————⑬
      0.010000      99.432585      0.005642     -4.605170
      0.020000      49.442210      0.011282     -3.912023
      0.030000      32.784998      0.016921     -3.506558
      0.040000      24.460955      0.022555     -3.218876
      0.050000      19.470085      0.028186     -2.995732
      0.060000      16.145727      0.033811     -2.813411
      0.070000      13.773601      0.039429     -2.659260
      0.080000      11.996566      0.045039     -2.525729
      0.090000      10.616217      0.050640     -2.407940
      0.100000       9.513508      0.056231     -2.302585

   90 STOP
```

```
ready
*DUMP ———————————————————————————————————⑭

I       =          11


A       =   0.10000000E   0

B       =   0.95135077E   1

C       =   0.56231361E  -1

D       =  -0.23025851E   1

 end of dump

ready
*GOODBYE ————————————————————————————————⑮
time          19:54              used-time        0  min.       0  sec.  960  msec.
```

説　明

① 端末を使用したいときには，まずＨＥＬＬＯと印字する。これが，システムに"これから端末で仕事を始める"という合図になる。

② 使用者の名称

③ ＫＥＩＯシステムを使用することを指示している。

④ いまから新しいプログラムを作成することを指示している。ファイルとして保存してあるものを使用する場合にはＯＬＤを印字すればよい。

⑤ プログラム名を印字すると，印字を許可する（ready）と共に時刻と日付を印字してくる。

⑥ プログラムを開始することを指示するコマンド（システムのコマンド）。このコマンドを送るとシステムから，

１０　ready

という応答がある。ここから，使用者はステートメントを送ることになる。以後，ステートメントを送るたびごとに，各行の先頭には自動的に行番号がふられていく。行番号は１０から１０おきにふられる。

この行番号は，次のような目的をもっている。

・プログラムの修正を行なうときに，その場所を指定するのに用いる。

・プログラム実行中にエラーが検出されたとき，その場所をシステムが指示するのに用いる。

・プログラムのリストをとる場所を指定するときに用いる。

・プログラムの実行を開始する場所とか，実行を中止する場所を指定するのに用いる。

⑦ 入力したステートメントに文法上の誤りがあったことを知らせてきている。このとき，次の行番号はもとのままであることに注意

⑧ プログラムが一応完成したので，実行させてみるために，コマンドが使用できる状態にするようシステムに要求している（システムのコマンド）

⑨ 実行を開始させる（システムのコマンド）

⑩ 印刷の形式がうまくないことがわかったので，端末装置にあるＡＴＴＥＮ－ＴＩＯＮキーを押して実行を中断させた。

⑪ プログラムのソース・リストの要求。

⑫ 印刷の形式を修正するために，行番号８０のＦＯＲＭＡＴステートメントを削除することを指示している（プロセッサのコマンド）。

⑬ あらためて，初めからプログラムの実行を開始させるよう指示している。

⑭ 各変数の値を調べるときに用いるコマンド（プロセッサのコマンド）。特定の変数や配列だけを指定することができる。

⑮ 端末の使用終了を告げるコマンド（システムのコマンド）。この結果，そのときの時刻（１９時５４分）と実際に中央演算処理装置を使用した時間（この例の場合には９６０msec しか使用していない）を知らせてくる。

プログラムのディバックに便利な機能の一例を次に示す。

```
ready                          17:19                2/1/69
*PROGRAM
   10 ready C      FACTORIAL TABLE
   20 ready          DO 10  I=1, 4
   30 ready          X = I
   40 ready          A = FACTOR ## (I)
   50 ready    10    PRINT  100, X, A
   60 ready          STOP  77777
   70 ready          END

undefined  statement  number  ←——————————————————— ①
100
   80 ready  *COMMAND
ready
*ALTER  50  ←————————————————————————————————— ②
   51 ready   100  FORMAT (2F20.10)
   52 ready  *COD # MMAND
ready
*TRACE  ←————————————————————————————————————— ③
ready
*START
   20 trace   I          =          1
   30 trace   X          =   0.10000000E  1  ←——— ④
   40 trace   A          =   0.10000000E  1
          1.0000000000                1.0000000000
   50 trace   I          =          2
   30 trace   X          =   0.20000000E  1
   40 trace   A          =   0.20000000E  1
```

説　明

① ステートメント番号１００が未定義であることを指示している。

② 行番号５０のステートメントのつぎにステートメントを挿入することを指示している。

③ プログラムの追跡実行を指定している。

このコマンドは，プログラム実行中に特定の変数あるいはすべての変数について，その変数の値が変わるたびに，ステートメントの行番号，変数名，および新しい値を印字することを指示するためのものである。

④ 行番号３０のステートメントを実行したことによって，変数Ｘの値が 0.1×10^1 になったことを意味する。

2. 会話形プロセッサの現状

2. 会話形プロセッサの現状

ＫＥＩＯシステムの例で，会話形プロセッサの姿，形はほぼ明らかになったと思われるが，何分，ＫＥＩＯシステムは単なる一例にすぎない。そこで，世の中の会話形プロセッサの態勢を把握するために，代表的なシステムの使用例をいくつかみることにする。

ここに取り上げたシステムは次の通りである。

・ＢＡＳＩＣ

・ＪＯＳＳ

・ＲＵＳＨ

・ＰＬＡＮＩＴ

・ＨＥＬＰ

この他にもいろいろなシステムがあるが，ＦＯＲＴＲＡＮとかＰＬ／Ｉといった既存の言語，あるいは先駆者が会話形として開発した言語の方言であるものが多い。たとえば，ＪＯＳＳの方言としては，次のようなものがある。

ＴＥＬＣＯＭＰ（ボルト，ベラネク　アンド　ニューマン社）

ＦＩＬＥＣＯＭＰ（ゼネラル　エレクトリック社）

ＣＡＬ　　（カリフォルニア大学）

ＦＩＧＡＲＯ　　（ケンブリッジ大学）

ＡＩＤ

また，タイムシエアリング・システム用に開発された特殊用途向きの言語もいろいろある。たとえば，ここで取りあげるＰＬＡＮＩＴはティーチング・マシンとしてタイムシエアリング・システムを使用するとき，そのためのプログラムを作成する言語であり，ＨＥＬＰはタイムシエアリング・システムに備えつけの質疑応答用システムである。

以下に主な会話形言語の一覧を示す。

会話形言語一覧

	名前	説明	システム	ユーザ
JOSS	JOSS (JOHNNIAC Open Shop System)	RAND社の開発したJOHNNIAC用に作られた会話形言語	JOHNNIAC PDP−6 PDP−10	
BASIC	BASIC (Beginners All Purpose Symbolic Instruction Code)	ダートマス大学で開発した会話形のTSS用語	PDP−6 PDP−10 GE−235 GE−255 GE−265 GE−420 GE−645 SDS−940 B5500 UNIVAC 1108	Applied Logic Corp CDC C−E−I−R Computer Network Computer Sharing Inc COM−SHARE Inc Data Network Corp Colla−Computer CSC DIAL−DATA Graphic Control Corp GE

	名　　前	説　　明	システム	ユ　ー　ザ
BASIC				On-Line System Inc Pillsbury Occidental Co Rapidata TYMSHARE
会話形FORTRAN	QUIKTRAN	IBM7040／44用の会話形FORTRAN言語	IBM7040／44	IBM
	RAX (Remote Access Computing System)	会話形のFORTRANとアセンブラが使用でき，IBMシステム／360モデル30以上に適用	IBMシステム／360モデル40H以上	
	CFOR (Convessat-ional FORTRAN)	UNIVACで開発した会話形FORTRAN言語	UNIVAC 1108	

	名　　前	説　　明	システム	ユ　ー　ザ
会話形FORTRAN	GE Time-Sharing FORTRAN	GEで開発した会話形のTSS用FORTRAN言語	GE－645	CDC C-E-I-R Colla-Compu-ter GE On-Line Sys tem Inc Pillsbury Occidental CO.
	SDS　会話形 FORTRAN	SDSで開発した会話形FORTRAN言語	SDS－940	Compute Sharing Inc COM-SHARE Inc Data Network Corp DIAL-DATA TYMSHARE
	RCA　会話形 FORTRAN	RCAで開発した会話形のTSS用 FORTRAN言語	SPECTRA 70／45	

	名　　前	説　　明	システム	ユ　ー　ザ
会話形ALGOL	会話形ALGOL	会話形のALGOL	GE 255 GE 265 GE 420 IBM 7044	CDC C-E-I-R GE On-Line System Inc Pillsbury Occidental CO
会話形PL/I	RUSH (Remoto Use of Shared Hardware	アレンバブコック社とIBMが共同で開発した会話形PL/I言語	IBMシステム/360モデル50	Allen- Babcock Co
会話形PL/I	CPS (Conversa- tional Programming System)	会話形PL/I言語でIBMシステム/360 モデル40H以上に適用され，RUSHの姉妹版	IBMシステム/360 モデル40H以上	
その他	AMTRAN (Automatic. Mathematical Translation)	キーボードによる計算プログラムシステムで，IBM1620などで使用	IBM 1620	Marshall Space Flight Center

	名　　前	説　　明	システム	ユ　ー　ザ
	BACCUS (Basic Calculus)	富士通で開発した会話形の計算用言語	FACOM 230－50	
そ	CAL (Conversational Algebraic Language)	会話形の計算用言語	SDS　940	Computer Sharing Inc COM－SHARE Inc Data Network Corp DIAL－DATA Inc TYMSHARE
の	DIALOG	図形入出力装置による会話形の計算用言語でＩＩＴ研究所で設計された。	UNIVAC 1105 IBM 7094	IIT Reserch Institute
	Lincoln RECKONER	ＭＩＴのリンカーン研究所で開発した計算用言語		
他	MAP (Mathematical Analysis Program	会話形の計算用言語	CTSS	

	名　　前	説　　明	システム	ユ　ー　ザ
その他	FAMOUS	LIST 1.5で書かれた数式処理プログラム	MAC	
	BRUIN (Brown University Interpreter)	ブラウン大学で開発したTSS用言語	IBMシステム／360　モデル50	

2.1　BASIC

BASIC (Beginners All-purpose Symbolie Instruction Code) は，簡単な代数言語で，1964年ダートマス大学でGE－265タイムシエアリング・システム（GE－235中央処理装置＋DN－30）に開発された。現在では，他のタイムシエアリング・システムにも開発されており，初心者が容易に学ぶことができ，そして使いやすい基本的な言語である。

特　徴

- ステートメントの種類が少なく，文法も簡単である。代入ステートメント以外のものにはLETとかFORというような必須語が必ずつく。
- FORTRANのような主プログラム，副プログラムというプログラム単位はない。実行に必要な手続きは，1つのプログラムの中に全部含んでいなければならない。
- この言語は比較的小さな問題を記述するために作られているために，ディバックの助けになるようなコマンドは用意されていない。
- ステートメントの前には使用者が必ず行番号をつけるようになっている。ステートメントの順序は，この行番号に従う。したがって使用者は，プログラムの実行順にステートメントを入力する必要はない。すでに入力したステートメントの行番号と同じ行番号をもつステートメントを入力すると，前のステートメントは破棄され，後から入力したステートメントが生きる。した

がってプログラムの修正は簡単に行なえる。

処理方式としては，ステートメントが入力された時点では何のチェックもせず，実行命令が出されてから初めてプログラム全体のチェックをするようになっているので，プログラム中の誤りはまとめて表示される。

プ ロ グ ラ ム 例

```
HELLO                                                    ①
USER NUMBER - - 020969
SYSTEM - - BASIC                                         ②
NEW OR OLD - - NEW
NEW PROBLEM NAME - - EX
READY

 10  READ N
 20  LETS ← I = 1                                        ③
 30  LET J = I ↑ 2
 40  IF N<= THEN 70
 50  I = I + 1
 60  GO TO 30
 70  PRINT J "↑2<" N "<=" I "↑ 2"
 80  GO TO 10
 90  DATA 8, 10, 36, 50
100  END
RUN                                                      ④

ILLEGAL INSTRUCTION IN 50                                ⑤
 50  LET I = I + 1
 70  LET K = I - 1                                       ⑥
 71  PRINT K "↑ 2<" N "<=" I "↑ 2"
RUN

2 ↑ 2 < 8 < = 3 ↑ 2
3 ↑ 2 < 10< = 4 ↑ 2                                      ⑦
5 ↑ 2 < 36< = 6 ↑ 2
7 ↑ 2 < 50< = 8 ↑ 2
TIME :  0 SECS.                                          ⑧
SAVE                                                     ⑨
```

（便宜上，システムからの応答には下線が引いてある。）

① 端末からタイムシエアリング・システムを使いはじめることをシステムに知らせる。

② 使用者の番号や使用するシステムの名前を問いかけてくる。ここで使用者は，自分の番号が０２０９６９であり，ＢＡＳＩＣ言語を用いてＥＸという名前のプログラムを作ることをシステムに知らせている。

③ 左むきの矢印（←）を打つことによって直前に印字した文字を１字抹消することができる。

④ プログラムを実行させる。

⑤ 行番号５０のステートメントに誤りがあったことを表示している。

⑥ ステートメント５０を訂正する。他にも誤りがあることに気がついて，ステートメント７０を削除し，そこに新たに行番号７０，７１のステートメントを挿入してから，再び実行させている。

⑦ 実行した結果

⑧ データを読みつくしてしまったのでプログラムの実行が打ち切られ，実行に要した時間が印字される。

⑨ このプログラムを後刻使用するのに備え，ファイルとしてシステムに保存しておくよう指示している。

2.2 ＪＯＳＳ

ＪＯＳＳ(JOHNNIAC Open-Shop System)はＲＡＮＤ社で開発された代数言語で，プログラミングの経験をもたない者でも簡単に使用できるような仕様になっている。

ＪＯＳＳは，文法は厳格だが，当りがよく，比較的自然であり，かつ習得しやすい。すべての計算はインタプリティブであり，各ステップの走査，翻訳，実行に要する時間は，１００ｍｓのオーダである。長い言葉の記憶はしないので，ユーザによって開発されたすべてのプログラムは，接続が切れると破壊される。

ＪＯＳＳではＦＯＲＴＲＡＮの文(Statement)に相当するものとしてステ

ップの終りには「。」がつけられる。

特　徴

・　ＪＯＳＳには直接コマンドと格納コマンドとがある。（ＪＯＳＳには，いわゆるコマンドとステートメントの区別がなく，すべてコマンドと呼ぶ）。直接コマンドはただちに処理され折り返し結果が印字される。格納コマンドはすぐ処理されないで，格納プログラムの一部としてシステムに保持される。直接か格納かの区別はコマンドの先頭に番号をつけるか，つけないかによって区別する。番号をつけると格納コマンドとなり，番号をつけないと直接コマンドとなる。格納コマンドは，入力した順序ではなく，コマンドにつけた番号の順序になっているものとして扱われる。直接コマンドは，算術式を印刷するコマンド（Type）を入力すれば，ただちに計算結果を求めることができるので卓上計算機としての役目を果たす。格納コマンドにつける番号はステップ番号（Step number）といい，ステップ番号の整数部を部分番号（Part number）という。格納プログラムはステップ番号順に実行させることも，部分番号単位に実行させることもできるので非常に融通性に富んだプログラムを作ることができる。

・　コマンドの種類は少なく，１つのコマンドでいろいろと変化に富んだ指定ができるので，憶えやすく，また融通性に富んでいる。

・　データの入力機能は簡素化されていて，実行時に端末からデータを読み込む機能はなく，直接コマンドとして代入コマンド（Set）を用い，各変数に値を与えながら実行するようになっている。

・　宣言文はない。もちろん添字つき変数を使うこともできるが，ＦＯＲＴＲＡＮのようにＤＩＭＥＮＳＩＯＮステートメントで前もって宣言しておく必要はなく，直接，式中で用いれば暗黙のうちに宣言したことになる。同一の配列名の添字式の個数は同じである必要はなく，配列として暗宣言した時点で次数も同時に暗宣言したことになる。

・　この言語は比較的簡単な数値計算を処理するのに便利なように作られているために，プログラム単位という概念はなく，独立したプログラムを呼び出して使うようなことはできない。ただし，部分番号を用いて同様のことを行

なうことができる。条件つき計算の他に再帰的な関数定義とか，汎関数を定義することもできる。

・　入力の誤りはできるだけ早い時点で検出され，完全な英文で適格に指摘してくれる。

・　変数名は英字1文字しか許されない

プ　ロ　グ　ラ　ム　例

```
Type  5 + 3, 4 · 3, 4 * 2, Sqrt. (2).                    ①
        5 + 3 =          8
        4 · 3 =         12
        4 * 2 =         16
     Sqrt (2) =          1.4142136                       ②

Delete all.                                              ③
Let     x =-b/(2·a).                                     ④
Let     y = Sqrt (b*2-4·a·c) / (2·a).
a = 1
b = 3
c = −10                                                  ⑤
Type    x + y, x − y.
          x + y =       2
          x − y =      -5

Delete all.
1.1     Do part 2 for x = 0 (0.1) 10.
2.1     Set  y = f (x).                                  ⑥
2.2     Type  x, y in form 1.
Form 1 :
--.-    . . . . . .
Let  f(t) = exp (-t*2/2)/Sqrt (2·3.1415927).             ⑦
Do  part  1.
  .0      3.989-01
  .1      3.970-01
  .2      3.910-01                                       ⑧
  .3      3.814-01
   .       .
   .       .
   .       .
```

（便宜上，システムからの応答には下線が引いてある。実際には，入力は"線"，出力は"黒"で印字される。）

① 直接コマンドで，5＋3，4×3，4^2，$\sqrt{2}$を計算し，その結果を印字させるコマンドである。

② ただちに上の各算術式が計算され，計算結果が印字される。

③ システムに保持させておいた情報を，すべて抹消することを指示する。

④ 右辺の式をxという名前で呼ぶことを定義している。

⑤ 各変数に値を与えて，2次方程式の根を求めている。

⑥ 格納コマンドで，あとからまとめて実行するときのために，システムにしまっておく。

⑦ ⑥のプログラムで未定義であった関数fを定義し，部分番号1の部分を実行させる。

⑧ 実行した結果

2.3 RUSH

RUSH（Remote Users of Shared Hardware）は，アレンバブロック社とIBMが共同で開発した会話形PL／I言語である。RUSHは科学技術計算と事務計算の両方の利用を目的としている。ユーザはIBM2741タイプライタ，IBM1050またはテレタイプのいずれかを用いる。システムとしてはIBMシステム360／モデル50Hが使用され，プロセッサ約120のモジュールからなっている。RUSH言語は，IBM PL／I F－レベルと互換性をもっている。またRUSHターミナルからIBMシステム／360の制御下にあるFORTRAN Ⅳ，COBOL，PL／I，アセンブラ，その他のものも，REMOTE JOB ENTRYシステム（RJE）を用いることにより使用できる。

特　徴

JOSSと同様，行番号の有無によって格納ステートメント（RUSHではcollect statementと呼ぶ）と直接ステートメントとがある。ただし，直接ステートメントの頭には"%"を付ける必要がある。

言語仕様は，ほとんどＰＬ／Ｉの部分集合になっている。といっても，ブロックの概念も構造の概念もない（最新版ではブロックの概念が付加されているとのことである）。さらに，変数等の属性は浮動小数点に，そして精度は１４桁に限られている。その他の点に関しても，かなり制約されている。

プ　ロ　グ　ラ　ム　例

```
/*     GENERATION  OF  PRIME  NUMBER  PAIRS     */

200    START:  PRIME, PAIR = 7;
210            DEL = 2;
220    NEWC:    IF  DEL = 2  THEN  DEL = 4;  ELSE  DEL = 2;
230             PRIME  =  PRIME  +  DEL;
240    TEST:    DO  D = 3  BY  2  TO  FLOOR(SQRT(PRIME));
250             IF  MOD(PRIME, D) = 0  THEN  GO  TO  NEWC;
260             END  TEST;
270             IF  PAIR = PRIME-2  THEN  PUT  IMAGE (PAIR, PRIME) (PIX);
280    PIX ; IMAGE;
FOUND ANOTHER PRIME PAIR FIRST IS -----, SECOND IS -----

290             PAIR = PRIME;
300             GO  TO  NEWC;
3100            STOP;

%XEQ  START  THRU  ...;
```

プログラム例の説明

システムは"—"を印字することによって，使用者がステートメントを打ち込める状態になったことを示す。

プログラム中行番号２８０のＩＭＡＧＥの機能はＪＯＳＳのFormと同じで

ある。

2.4 PLANIT

ＳＤＣ社のＱ－３２タイムシエアリング・システムに開発された言語システム（Programming LANguage for Interaction and Teaching の略）である。

システムは，

・教程作成

・編　集

・実　行

・計　算

という４種のモードをもち，教師はこれらすべてのモードを使用できるが，学生は実行と計算のモードしか使用できない。

作成可能な問題としては，学生に特定の解答を求めるもの，いくつかの答を提示しておき学生に正解を選ばせるものなど各種のものがある。

特　徴

学生の解答を採点する仕方も，正解が一つしかない場合の照合はもちろんのこと，次のような採点の仕方を随時採用することができる。

・綴りが間違っていても，発音上正しければよい。

例　ＰＬＡＩＴおよびＰＨＯＮＥＴＩＣが正解のとき，ＰＬＡＮＥＴやＦＯＮＥＴＩＣも合格にすることができる。

・正解が文章中に含まれていればよい。

例　ＧＥＯＲＧＥ　ＷＡＳＨＩＮＧＴＯＮが正解のとき，ＩＴ　ＷＡＳ　ＧＥＯＲＧＥ　ＷＡＳＨＩＮＧＴＯＮも合格にすることができる。

・数式が答の場合には，数学的に等値であるものはすべて合格にする。

例　（５／９）＊（Ｆ－３２）が正解のとき，５／９＊（Ｆ－３２），（５＊Ｆ－１６０）／９，（－３２＊（５）＋５＊Ｆ／９などをみな合格にすることができる。

この他，学生の解答に従って任意の教程を組むことができるし，ある時点

までの学生の正解率とか所要時間とか誤答率に従って，すなわち，この時点までの学生の履歴に従って，その先の教程を決めることもできる。

ただし，１文字で機能を指定することからプログラミングの仕方には難点がある。

プ　ロ　グ　ラ　ム　例

```
FRAME  1.00  LABEL=*HIST  ←――――――――――――――――――①
2.   SQ.
*?  ←――――――――――――――――――――――――――――――②
2.   SPECIFY  QUESTION.
*WHO  INVENTED  THE  ELECTRIC  LIGHT?  ←――――――――――③
*  ←――――――――――――――――――――――――――――――④

3.   SA.  ←――――――――――――――――――――――――――――⑤
*A+THOMAS  EDISON
*B  ALEXANDER  BELL
*

4.   SAT.  ←――――――――――――――――――――――――――――⑥
*A  F:  THATS  VERY  GOOD  B:3
*B  R: HE  INVENTED  THE  TELEPHONE, TRY  AGAIN...
*-R:
*-C:
*
```

（下線部がシステムからの応答である。）

プログラム例の説明

① この部分（フレーム）に名前をつける。

② 使用者が＂ＳＱ＂の意味がわからないので，完全な文を要求している。

③ 出題すべき問題を作成している。

④ 問題ができてしまったので，空白を１字打ってこの項の終りを告げている。

⑤ 正解（次の行のＡの後の＂＋＂で指示）および予想される解答を用意する項

で，ＳＰＥＣＩＦＹ　ＡＮＳＷＥＲの略

⑥　学生の解答に従って次にとるべき動作を指定する項で，ＳＰＥＣＩＦＹ　ＡＣＴＩＯＮＳ　to be　ＴＡＫＥＮの略

"＊"に続くＡ，Ｂは上の解答の先頭に付けたもので，いわばレーベルである。解答と同じレーベルのところに指示した動作をとる。このとき，同じレーベルに対する動作指定をいくつも書いたときには，上から順に使いすてられる。"＊"に続く"－"は，解答の項で与えておいたもの以外の解答に対する動作指定であることを示す。

Ｆ：，Ｂ：，Ｒ：，Ｃ：はコマンドである。"Ｆ："は先に進むことを指示する。"Ｂ："は分岐命令。"Ｒ："はこの問題に対する解を持つことを，そして"Ｃ："は"ＴＨＥ　ＣＯＲＲＥＣＴ　ＡＮＳＷＥＲ　ＩＳ　＊＊＊＊　と印刷することを指示する（ただし，＊＊＊＊　のところは正解を印字する）。

このプログラムを実行させたときの動きは次の様である。

解答が正解（Ａ）の場合には，"ＴＨＡＴＳ　ＶＥＲＹ　ＧＯＯＤ"と応答し，フレーム３に進む。解答がＢの場合は，"ＨＥ　ＩＮＶＥＮＴＥＤ　ＴＨＥ　ＴＥＬＥＰＨＯＮＥ，ＴＲＹ　ＡＧＡＩＮ．．．"と応答し，別の解答を待つ。解答の項で与えておいたもの以外の解答の場合には，最初は，"ＷＲＯＮＧ，ＴＲＹ　ＡＧＡＩＮ"（Ｒ：の後の応答すべきコメントが指定されていない場合の応答）と応答し，別の解答を待つ。次のものもこの種の解答だったときには，"ＣＯＲＲＥＣＴ　ＡＮＳＷＥＲ　ＩＳ　ＴＨＯＭＡＳ　ＥＤＩＳＯＮ"と応答することになる。

非常に魅力のある言語なので，もう少し複雑な応答をするように組んだプログラム例を次に示す。解答に対する応答が多少変化することになる。変化の具合は省略する。ただし，解答を用意するところで"＊ｏ　ＦＯＲＭＵＬＡＳＯＮ"とあるのは，前述の特徴のところの第３項に述べた機能を働らかすように指示するものである。

プログラム例

FRAME 2.00 LABEL = *MATH

2. SQ.

*LETS SEE WHAT YOU REMEMBER ABOUT TEMPERATURE. USING F FOR DEGREES

*FAHRENHEIT AND C FOR DEGREES CENTIGRADE WRITE THE FORMULA FOR

*CONVERTING FROM DEGREES FAHRENHEIT TO DEGREES CENTIGRADE.

*HINT: F=9*C/5+32 CONVERTS FROM CENTIGRADE TO FAHRENHEIT.

*

3. SA.

*0 FORMULAS ON

*A + C = (5/9) * (F - 32)

*B F = 9*C/5 + 32

*C C = (5/9) * F - 32

*

4. SAT.

*A F: B : 7

*B R: YOUR ANSWER IS THE SAME AS THE ONE I GAVE YOU, TRY AGAIN...

*A F: NOW YOU'VE GOT IT. B : 15

*B R: YOU'RE STILL CONVERTING FROM CENTIGRADE TO FAHRENHEIT, TRY AGAIN.

*BC F : NOTE THE DIFFERENCE. C: B: OUT

*-R:

*-C:

2.5 HELP

HELPはSDS940タイムシエアリング・システムに用意されている質疑応答用システムで，端末の使用者が，システムの使い方等について質問をすると，それに答えたり，指導したりしてくれる。

特　徴

- HELPは質問・応答形式をとり，日常の言語で質問を受けつけて，それに対する適切な答をデータ・ベースからとり出して使用者に応答する。
- HELPによって，SDS940タイムシエアリングの使用者はシステムに関する予備知識をもっていなくても使える。使用者が何か質問すると，それに対して親切な応答をしてくれるので，端末に指導員がいる必要はない。
- HELPは質問中の必須語によって質問の意味を解釈するので，質問の構造や文法に制約はない。また，多少のつづりの誤りがあっても適当に解釈してくれる。
- FORTRANやALGOLなどのサブシステムを使っている際中でも，質問したいことがあればHELPを呼び出すことができる。このときHELPは，使用者が使っていたサブシステムが何であるかを考慮して，正しい受け答えができるようになっている。
- HELPのデータ・ベースに使用者が，随時，データを追加することができる。

プ　ロ　グ　ラ　ム　例

```
*HELP.                                                              ①
?WHAT IS HELP?                                                      ②
HELP IS THE ON-LINE INFORMATION SUBSYSTEM OF THE SDS 940 TIME SHARING SYSTEM.
TYPE ANY SPECIFIC QUESTION ABOUT QED AND HELP WILL
ATTEMPT TO BE HELPFUL!  USE ONLY LETTERS, NUMBERS,                  ③
AND SPACES.  END EACH REQUEST WITH "." OR "?"
IF HELP "MISUNDERSTANDS", INTERRUPT IT WITH A RUBOUT
REPHRASE YOUR QUESTION.
```

プログラム例の説明

① この例は使用者がテキスト編集プログラム（ＱＥＤ）を使っていたときのものである。使用者が打った部分は下線を引いて示す。Ｈを打つとシステムが残りの文字を印字してくれ，使用者はピリオッドを打つことによってそれが正しいことを確認する。

② ＨＥＬＰは質問を受け入れられる状態にあることを示すために＂？＂を印字し，質問を待つ。ここで使用者はＨＥＬＰとは何かという質問を出している。

③ その質問に対して次のような答が返ってきている。

ＨＥＬＰはオンライン情報システムでＳＤＳ９４０タイムシエアリング・システムのサブシステムである。ＱＥＤに関する質問なら何でもタイプすればＨＥＬＰが何とかお答えします。使える文字は英字と数字とそれに空白で，各質問は＂．＂か＂？＂で終っていなければなりません。

ＨＥＬＰが質問を誤解したときには，ＲＵＢＯＵＴで印字を中止させて，質問の形式を変えなさい。

3. 会話形プロセッサの基本理念

3. 会話形プロセッサ

会話形プロセッサは，前述の通り，電子計算機との交互作用（interaction）によってプログラミングをできる限り効率よく行なえるようにしようという哲学にもとづいて開発されたものであり，現在も依然として開発され続けている。現在では，当然のことながら，会話形プロセッサを開発しようとするシステム環境は会話形タイムシェアリングシステムあるいは大型汎用タイムシェアリングシステムということになる（残念なことに，大型汎用タイムシェアリングシステムは多大な努力 — もちろん，経費も — がなされているにもかかわらず挫折している状態にある）。現在さかんに会話型プロセッサが開発され続けている理由は，電子計算機システムの発展の一過程であり，遠隔システムおよび会話形タイムシェアリングシステムまでの発展過程に依存している。事実上，会話形プロセッサは会話形タイムシェアリングシステムと一体であるとみなすことができる。そこで，まず会話形タイムシェアリングシステムに至るまでの電子計算機システムの発展過程を簡単にながめてみよう。

電子計算機の性能が高まるにつれて，主要な問題はその経済性であった。初期の頃には，使用者は電子計算機の中央操作卓の前に腰を据え，直接電子計算機を操作しながら問題を解いていた。プログラムを部分的に実行させては，その結果を調べたり，ときには1命令ずつ実行させながらその都度電子計算機の各種レジスタを調べたりしてプログラムを作り上げていった。しかしながら，これでは人間の動作時間あるいは思考に要する時間と電子計算機の作動時間が極端に違うことから，あまりにも不経済であった。そこで，人間はだんだんと電子計算機から隔離されるようになってきた。すなわち，電子計算機を無駄なく利用するためには，できる限り人間の仲介を最少にする必要があった。そこで，まず電子計算機の前に腰を据えて直接電子計算機をいじりながらディバッグすることは止めにして，プログラムを次々に処理していくように改められた。こうすると，1つのプログラムの処理が終った時点で，次のプログラムを用意するときにだけ人間が介入しさえすればよい。ところが，電子計算機の処理装置自体をみてみると，機械的動

作を伴う入出力を行なっている間は大方遊んでいる。そこで，中央処理装置はデータの処理に専念させ，入出力装置の制御は廉価なハードウエアで処理するような工夫がなされ，１９５０年代の終り頃にデータ処理と入出力操作が非同期に行なえるような機構が開発された。

並行処理が可能となると，１つのプログラムの演算部分と入出力部分とを分割し，できる限り，プログラム全体に要する時間を最少にするように組合せようとする努力がなされたが，たとえ最適な組み方をとっても電子計算機システムの構成要素をすべて十分に使用することは到底無理なことである。そこで１９６０年頃には入出力制約のプログラムと計算制約のプログラムを組み合わせることにより，総体的な計算時間だけでなく，システムの構成要素の使用効率を一層最適化するようになった。こうして多重プログラミングが始まった。ひき続き，この概念を発展させ，多数の使用者のプログラム間で電子計算機を分割して使用するようになった。

ところが，電子計算機の利用面が次第に開発されて行くにしたがって使用者の数が増し，電子計算機室に依頼してから結果が手元に戻ってくるまでの時間，すなわち，応答時間（ turn-around time ）はのびる一方で数時間から数日かかるようになってきた。こうして，人間と電子計算機との密接さが薄くなってくるとプログラムの作成面に悪影響を与えるようになってきた。何日か隔ってからやっと戻ってきた結果が正解の場合はまだしも，プログラミングの誤りがあったときにはもはや悲劇である。ところがプログラムはこの悲劇を繰り返すことによって作成されるものであるから，プログラマの作業に与える打撃はひどいものであった。なるほど，ハードウエアの進歩により演算速度は速くなったが，その結果，計算に要するコストがさがったのと相まって電子計算機使用人口が急激に増加してきたことから，速度がはやくなっても応答時間は短かくならず，密接さの欠乏からかえって使用回数が増加したので，人間を機械から隔離することによって得ることができた利益をかえって計算に要するコストで食いつぶしてしまうことになった。さらに，事務計算など一定の処理プログラムを作成しておき，データだけをその都度変更して使用するような場合はさておき，大学とか企業の研究所など各種の研究を行なう機関では，問題を解くというよりは，むしろ，その時その

時に問題を定義するような場合がその大部分であるような所では，次にとるべき行動を決めるのには，その前のステップでの結果を必要とすることから，この種の問題には電子計算機はむかないものと思われていた。

これにかわる方法となると，再び直接電子計算機を操作しなければならないようなことになる。こうなると折角大型化，高速化された電子計算機をまったく無駄に使用することになる。しかも，前にもまして経済的にも莫大な浪費をすることになる。また，各種のディバッグ用ルーチンが用意されているとはいえ，何とも機能的ではない。すなわち，効果的な処理を行なうことができない。したがって，人間と電子計算機との連絡を助ける複雑ではあるが多方面にわたり面倒をみるシステムプログラムをつくり，経済的にも無駄な浪費をすることもなく，かつ交互作用が機能的にできるようにすることが必要となったのである。

さらに，交互作用の度合を高めるだけでなく，このような研究機関では，必要とするときに，いつでも容易に電子計算機を利用できることが望ましい。しかも，電子計算機が普段使用している鉛筆とかノート，計算尺などのように研究のための有益な道具として使用できることが望ましい。

この様な必然性，必要性と，この時までに開発されていたハードウエアとソフトウエアの各種技法とが相まって登場したのが会話形タイムシェアリングシステムである。

3.1　会話形のための必要条件

まず，タイムシェアリングシステムも含め，遠隔情報処理システムを設計する際に考慮しなければならない事柄には次のようなものがある。

1　遠隔使用者にはプログラミングの際に協力してくれたり，助言を与えてくれるような専門家が近くにいるとは限らない。

さらに使用者が問題を問題向き言語を使って解いているときには，入力する要求事項とか，出力されるデータなどの表示形式は，そのプログラミング言語でしたいだろう。

2　遠隔使用者の仕事は，完全に人間が介在することなしに処理されることから，使用者は自分の希望を機械操作員に連絡することができない。このこと

から，次のようなことに関して考慮する必要がある。

a．システムを統制するのに用いるコマンドは，採用するプログラミング言語と形式，内容共に調和しているものにすること。

b．遠隔使用者は強力なコマンド組織を要求する。たとえば，実行時間，仕事の状態，エラーが生じたときの処理手続き，あるいは出力の形式といったような事柄が記述できること。

c．会話形システムの遠隔使用者には操作卓上にあるボタンとかライトとかスイッチといった機械操作員が使用できるいろいろな機能が使えるようにすること。さらに，たとえば端末にライトを用意し，これを定期的に明滅させるというような具合で，"万事うまくいっている"ということを知らせて，使用者を安心させてやるような工夫も必要であろう。

d．いついかなるときでも，どこにも影響を与えることなく"機械"を止めることができるようにしておくべきである。これは，遠隔使用者が人間であるという理由からと，こうしておくと，印刷用紙を取り換えたり，入力カードを読取装置に追加したり，あるいは，仕事を中断したりするようなちょっとしたことができるようになる。

3　遠隔使用者は入出力の量に非常に気をつかうものである。そこで，人力データを全部伝送し直さなくても，すでに入力済みのデータを修正したり，厖大な出力ファイルを，随時，部分的に調べたり，リストをとったりすることができなければならない。このようなことから，大容量乱呼び出し記憶装置に種々のデータが保存でき，修正とか検査とか処理などが迅速にしかも手軽にできるようにすべきである。

4　遠隔使用者には，使用者は自分しかいないで，すべて自分が管理しているんだという印象を与えるようにすべきである。さらに，タイムシェアリングシステムのもとでは，他人の破壊的な行為の恐れがまったくないようにすべきである。理想的には，端末での計算および応答の速度は，他の使用者達の要求のいかんにかかわらず，まったく変わらないようにすべきである。

5　最後に，使用者は電子計算機本体から遠く離れたところにいることから，心理的な面には十分配慮しておく必要がある。

ところで，会話形タイムシェアリングシステムというものは，システムへの入出力データの量が少なく，データ処理の仕事自体もそれ程多くないものに向いており，まとまった仕事を端末で実行するためには遠隔バッチ処理システムの方が適している。このことは会話形プロセッサについても云えることで，プログラムを作り上げ，ちょっとしたデータを用いてディバッグを行なうところまでが会話形プロセッサの守備範囲であろうと思われる。そこで，会話形プロセッサの基本精神というものが必然的に定まることになる。

会話形プロセッサの基本精神としては，次のようなものが考えられる。ただし，これらの諸事項は，当然のことながら，そのプロセッサを開発する電子計算機システムに完全に依存する。

1　使用者から入力があった時点で，できる限りの診断を行ない，誤りがあったときには折り返えしその旨を知らせること。

2　プログラムの修正が簡単にできること。

3　処理時間はできるかぎり短くすること。

4　プログラムの実行を途中で止めて，プログラムの一部を修正したり，任意のところから実行を再開したりすることができるなど，融通性のあるオペレーションができること。

5　プログラミングに必要な情報は即座にとり出せること。

6　少なくともプログラミング言語と同じレベルの言語でディバッグができること。

7　印刷の量は，使用者の必要に応じて，値を落すことなく，随時，自由にかえることができること。

8　言語体系は，少なくともFORTRAN 言語位の程度で容易に学ぶことができること。

9　プロセッサを開発する電子計算機システムの精神とソフトウエアの精神とが合致していること。

ここで，次の3種類の問題が生じる。

○　開発しようとしているシステムの精神にもとづくもの。

例　会話形プロセッサでの処理時間を苦労して短縮したところで，システムのハードウエアとかオペレーションシステムの設計に端を発するオーバ

— ヘッドが，問題としている処理時間などとは桁違いに大きいような場合。

○　言語仕様にもとづくもの。

例　会話形プロセッサでは，常に宣言文のあり方が問題になる。（この点に関しては＂会話形プロセッサのあり方＂のところで詳述する）。

○　作成者の趣味にもとづくもの。

この結果，使用者に対しては，ある程度の制約が生じることになるかもしれないが，これは止むを得ないことであろう。

3.2　ハードウエアおよびＯＳとの関係

会話形プロセッサの効率および機能は，まったくのところ，その親に当るシステムのハードウエアとＯＳに依存している。ハードウエアは電子計算機本体およびその周辺装置と，端末装置に大きく別けることができる。さらに，電子計算機本体およびその周辺装置の方はＯＳと一体になっていることから，ここではＯＳに含めて扱う。いずれにせよ，電子計算機システムのハードウエアはソフトウエアの精神と合致しなければならないということは当然のことである。

会話形プロセッサで一番問題となることは，システムの使い易さと応答時間である。使用者が生身の人間であるところから心理学的要因を十分考慮しなければならない。応答時間に関しては，以前バッチ処理方式のシステムを使用していた頃には往復所要時間が4問間位だと喜々としていた連中が，会話形システムでは数秒間で応答しないとイライラしてやめてしまうという結果が，米国での調査によって観察されている。そのために，たとえば＂ＷＡＩＴ＂というような簡単な応答でもしておくと，自分の端末は稼動状態にあって，システムとまだつながっており，しかもシステムは目下自分に対するサービスをしているんだという安心感を多くの使用者に与えることができるとのことである。ところが，慶応義塾大学での実験では，この＂ＷＡＩＴ＂も頻繁に出すと，かえって逆効果であり，応答時間を考慮した上で出すようにすべきであるという結果がでている。

そこで問題になるのがＯＳのオーバーヘッドである。プロセッサの処理速度

を無理をして速くしたところで，ＯＳのオーバーヘッドが非常に大きいと，無理をした甲斐がない。たとえば，ＩＢＭ３６０／６７のＴＳＳ／３６０末レリーズ版（第５８版）では，８端末でアセンブラを使用したときの平均応答時間は３０秒以上もかかるとのことである。この最大の原因は，ページ操りに要するオーバーヘッドであるとのことである。応答時間がこれ程にもなると会話型としては不適当であり，ＯＳその他の点に関して改良しなければならない（大容量コア記憶装置を導入し，上の平均応答時間を０.１秒程度に改良したシステムがカーネギー・メロン大学で開発されている）のは当然であるが，ＯＳのオーバーヘッドを比べ，無視できる程度の時間をプロセッサの処理の方にしわよせする必要はないだろう。

使い易さという点に関しては，主として言語仕様や端末装置の方で問題となるが，システムコマンドの仕様やシステムからの応答に関しても，当然，熟慮しておかなければならない。

さらに，立派な会話型プロセッサが開発できるようにＯＳは機能的にも完備されていなければならないし，融通性に富んだものでなければならない。

端末装置の設計も大変むずかしい。使い易さという面からするとスイッチとかキーなどは人間工学的に無理のないように配置しておかなければならない。一旦，端末装置が設置されると，その機能からソフトウエアにいろいろな影響をおよぼすことになるから，システムに合ったものを作る必要がある。装置のもつ機能以上のことをソフトウエアに組み込み，使用者にわずらわしさを感じさせるようなことはすべきではない。したがって，作成するプロセッサも，そのシステムに合ったものにすべきであり，ある程度の制約を受けてもいたしかたないであろう。

こういった事をすべて考慮に入れると，システムを開発する際には，そのシステムの守備範囲を定めたシステムの憲法を文書化し，定めておくようにすべきである。この好例として，ＪＯＳＳ　ＲＵＢＲＩＣＳ（ＪＯＳＳの哲学）がある。これは，ＪＯＳＳシステムの"哲学"を記述したものであり，ＪＯＳＳのようなシステムを作成するとき，およびＪＯＳＳと他の類似のシステムとを比較するときには注意しなければならない事項を羅列したものである。全部で２８項目あ

る。そのうちの幾つかのものを次に紹介しておく。

○　電子計算機のハードウエアとソフトウエアは完全に"封印"されていて，使用者はそれを見ることができない。ＪＯＳＳは施行施設とは独立している。

○　ＪＯＳＳではコミュニケーションは，いつも同じソフトウエアにより同様に解釈される。

○　ＪＯＳＳのコマンドは自然語でかけるようになっており，思考用言語としての性質をもつ。

○　ＪＯＳＳは完全な入力を要求し，出力する。2通りの解釈ができるようなものはない。ＪＯＳＳのコマンドをつくる規則は，きちんとした英文と一列に数式をかくための規則からなっている。

○　ＪＯＳＳの言語は可能な限り一般的につくってある。例外をつくらないために，規則は単純になっている。

○　ＪＯＳＳの規則の適用範囲は明確で，わりと簡単に解釈できるようになっている。

○　ＪＯＳＳを説明したり，プログラム相談等のために専門家はいらない。

・　・　・　・　・　・　・　・　・

○　ＪＯＳＳ言語は上記のすべてを考慮しなければならないので，一般にはそう簡単に拡張できない。

4. 会話形プロセッサのあり方

4. 会話形プロセッサのあり方

電子計算機システムを上手に利用できるかどうかは，そのシステムに関する詳細や手続きをどれ位知っているかによる。また，それ程重要でないようなものでもオペレーションをうまく行なうためには欠くことのできないような問題が絶えず生じる。これらの問題は，端末にいる使用者にとって非常にきびしいものとなる。イライラさせられるような問題とか，取扱いに困るような問題が生じた場合には，端末にいる使用者はたちまちまごついたり，失敗したり，ときにはシステムに対して腹を立てることさえある。したがって会話形プロセッサはもちろんのこと，一般に遠隔情報処理システムを開発する場合には，使用者に必要な援助を与えることができる機能をもたせる特別な考慮を払わなければならない。ここでは，それらのうちの代表的な事柄について論じることにする。

4.1 言語仕様

遠隔端末で使用する言語は，プロセッサの言語はもちろんのことシステム全体にわたるコマンド・システムまで，バッチ処理方式のシステムのときよりも一層念を入れて設計する必要がある。会話形プロセッサの親にあたる会話形タイムシェアリング・システムの目的の１つは，プログラミングの専門家でない人達に電子計算機をどんどん使わせるようにすることである。言語の形式，構文法およびその他の特殊用語などは，電子計算機になじみのないこれらの使用者にとって理解しやすいものにすべきである。使いやすいということは，覚えやすいかどうかということに結びつく。したがって，プロセッサの開発者にとって便利だからといって，機能と縁もゆかりもない名前とか文字を採用することはすべきではない。

SDS 940 タイムシェアリング・システムに DDT という言語がある。DDT は同システムにあるアセンブラと密接な関係にある汎用のオンライン・ディバッギング・パッケージである。このシステムのうたい文句に，使用者は迅速に交互操作を行なえるとあるが，実際には，使用者は約５０個もの動詞について，

次に示すようなコードを記憶しなければならない。

! ブレイク ポイント

) パッチの挿入

;U 実 行

はたして，これで迅速な交互操作ができるだろうか。

会話形プロセッサでは，特に，わかりやすい，しかも意味内容と結びついた言語を用いるべきである。こうすると言語によっては相当ながい言葉がでてくる。初心者にとっては好都合であるが，あまり長いものは，一旦覚えてしまうとわずらわしくなる。そこで，この種の言葉に対しては略語も用意しておくことを薦める。

たとえば，PL／I には次のような略語が用意されている。

きめられた語	略語
CHARACTER	CHAR
DECLARE	DCL
DEFINED	DEF
INITIAL	INIT
PICTURE	PIC
PROCEDURE	PROC
VARYING	VAR

また，SDS 940 タイムシェアリング・システムでは，使用者はシステムを使用するにあたって自分のなじみ具合を宣言するようになっている。なじみ具合は，初心者（ beginner ），未熟者（ novice ），熟練者（ expert ）の3段階にわかれている。初心者の場合，システム・コマンドは完全な綴りで使用しなければならない。たとえば，次のように完全な形で印字する。

DEFINE FILE

未熟者や熟練者の場合には， DEF と印字するとシステムがそれを認知して残りを印字してくる。

DEFINE FILE

システムが残りの文字を印字しているときには，未熟者の使用者からの入力は

無視される。この場合，使用者が熟練者であれば，システムからの印字中に入力した情報はバッファに保存され，コマンドを印字したあとで処理すべき入力とみなされる。

次に，大部分のステートメントは，JOSS 流にいうと，直接（direct）ステートメント，格納（storedまたはindirect．RUSHではcollect）ステートメントの両方に使用できるようにすべきである。直接ステートメントとは，入力と同時に実行されるステートメントであり，格納ステートメントとは，普通一般のものと同じで，プログラムとして後刻の実行に備え貯蔵されるにすぎないステートメントである。JOSS では，ステートメント（正確には，JOSS ではコマンドという）の前に番号をつけるか否かで区別している。

例
```
1.1  Type x.
1.2  Stop.
1.3  Type x*3.
Do part 1.
Error at step 1.1 : x ???
x = 2
Go.
                x =           2
Stopped by step 1.2.
```

ここで，番号が付いているものが格納ステートメントであり，番号が付いていないものが直接ステートメントである。便宜上，システムからの出力には下線が引いてある。

このように，直接ステートメントと格納ステートメントを設けると，非常に融通性に豊んだ言語ができ上がる。最も素朴な会話型プロセッサであるデスク・カルキュレータと呼ばれているものは，いわば，この直接ステートメントだけから成るシステムである。初期値の変更とかプログラムの実行途中で変数の値をいろいろと変更したりすることなどはいとも容易にできることはもちろん，その場限りの計算に関しては非常に便利になる。

さらに，JOSS のように部分番号（part number）とステップ番号（step

number ）とを採用すると，増殖型コンパイラ（ incremental compiler）としての機能が加わり一層機能的にも融通性が増す。部分番号とステップ番号の機能を示す簡単な例を次に示す。ステートメントの前に付けられた番号がステップ番号であり，そのうち小数点の左側が部分番号である。

例
```
1.1  Type "step 1.1".
1.2  Do part 2.
1.3  Type "step 1.3".
2.1  Type "step 2.1".
2.2  Type "step 2.2".
Do part 1.
step 1.1
step 2.1
step 2.2
step 1.3
Do step 2.1.
step 2.1
Do part 2.
step 2.1
step 2.2
```

ステートメントに番号をつけると，別にプログラムを順序だって段階的に作成する必要がなくなる。また，挿入や削除が楽にできる。このためだけに番号付けを採用したものとしてはBASIC や RUSH がある。

実行開始を指定するステートメント（一般にはコマンドである）は，いろいろなものを取り揃えておくと使用者に対して親切なものができる。しかし，同じような機能をもったものがいろいろとあるとかえって煩雑になり，使いにくい面もでてくる。この実行開始を指定するステートメントは例外として考えるべき種類のものである。たとえば，KEIOTOSBAC TSS の KEIO システム（会話形FORTRAN ）では実行開始を指定するものとしては，START というコマンドだけである。ただし，単に START と指定すると最初はプログラムの先頭

から実行開始することを意味し，中途で実行を中断したときには，その時点から実行を再開することを意味する。START に続けて行番号で始点と終点の両方あるいは一方だけを指定することもできる。BASIC では RUN というコマンドだけである。JOSS では，DO と GO とがあり，続行する場合に GO を用い，その他 KEIO と同じ様なことは DOで行なう。ただし，JOSS の DO は単に実行の開始を指定するだけではなく，実行の仕方も指定することができる。

さらに，この他にディバッグ用のものと本番用のものを用意すべきであろう。UNIVAC 1108 のタイムシェアリング・システムにある CFOR （会話形FORTRAN）には，本番用のものとして BEGIN，ディバッグ用のものとして EX と2様のものが用意されている。実行開始を指定するステートメントで区別しないで2様の実行モードを用意することもできる。どちらにせよ，このように区別しておくと，直接と格納の2種のステートメントを採用した場合とか，BASIC 様の DATA ステートメントを用いる場合には，特に，その便利さを発揮する。

直接と格納の2種のステートメントを設けた場合，その場限りの計算を行なうのならば別だが，後で再度そのプログラムを利用するような場合を考えると，プログラミングの段階で随時実行してみて完成品に近ずけるのが普通である。このとき，プロセッサの作り具合，すなわち面倒をどの程度までみるかによって異なるが，たとえば，実行時に未だ値が代入されていない変数が参照されたときは，その変数名を示し一担止めるのが会話形プロセッサとしては普通である。このとき，使用者は直接ステートメント（代入ステートメント）を用いてこの値を定義し，実行を続けさせ最後まで無事に行ったとする。そして，新ためて本番の開始を指示したとき，この変数はどのように処理されるだうか？面倒見のよくないプロセッサだと，そのままの姿で実行してしまう。結果は，当然，正しいとは思えない。実行のモードを本番とディバッグに区別しておくと，この様なことは避けられる。"ついうっかりして忘れた"ということは普通の人間であれば日常茶飯事あたりまえのことである。この程度のことは，会話形プロセッサと呼ぶ限りは面倒を見るべきではなかろうか。

BASIC の DATA ステートメントを用いた場合は，別の意味でもっと切実に

問題が生じる。この DATA ステートメントにかかれたデータは，READ ステートメントが実行されるとなくなってしまうのでディバッグを行なう度毎に，再度，DATA ステートメントで定義し直さなければならないという事態が生じる。こうなると，ヘタにこの種のステートメントは使うことができなくなる。そこで，前述の場合よりもさらに融通性をもたせた2様の実行開始指定ステートメントを用意すべきである。これは会話 であるが故に考慮しなければならない事柄の一例である。

さらに，会話 であるが故に，宣言ステートメントは実行ステートメントにすべきであるということを提唱したい。現実に，既存の言語を会話型にしようとすると必ず問題になるのが宣言ステートメントである。理想的には，会話型プロセッサでは宣言は動的にできるようにすべきである。たとえば，JOSS では配列は宣言しないでも，添字を付けて使用すると動的に配列として処理される。1つのプログラムで同じ名前が2次元の配列になったり，3次元になったり，自由に変えることができる（10次元までの配列が許される。）

例

```
Set a(251) = 3.
Index value must be integer and 1 index 1≦250.
Set a(1,2,3,4,5,6,7,8,9,8,7)=5.
Please limit number of indices to 10.
Set a(2,1)=5.
Set a(3,4)=10.
Type a(3,4).
                a(3,4)=          10
Type a.
                a(2,1)=           5
                a(3,4)=          10
Set a(3,2,1)=7.
Type a.
                a(3,2,1)=         7
a(1)=1
```

```
Type a.
                a(1)=          1
a = 3
Type a.
                a =            3
```

（注） 代入文は Set ……．という形式であるが直接コマンドとして用いるときに限り Set と最後のピリオドはなくてもよい。

PL／I のように属性がたくさんあると，こう簡単にすることができなくなる。省略時解釈を存分に用いたところで，やはり大変である。通常，FORTRAN 等での2重宣言の処置は，会話形では踏襲する方が自然であろう。ところが，踏襲するようにしたとき，具体的にどのように処理すべきなのか？バッチ処理方式のシステムとの互換性を考えたときの処理の方法はどうすべきなのか？ 等々いろいろと問題が生じる。そもそも会話形である BASIC でさえこのわずらわしさから逃げようとする例がある。バージニア大学のB 5500 の BASIC では，配列宣言は同時に幾つもできないというような制約を設けている。しかし，まだ問題が残る。たとえば，次のような例を考える。

```
100   DIM   X(15)
200   LET   X(1) = 0
100   DIM   X(15, 2)
```

このとき行番号 200 のステートメントは明らかに誤りである。これはどうすべきか？

宣言ステートメントを実行ステートメントにすると，この種の問題はすべて解決するが，文法上正しいかどうかの検査は直ちに行なうことができない。文字列として正しいかどうかの検査しかできない。ただし，この種の誤りの検出は実行時に行なうことができる。

さらに，バッチ処理方式のシステムとの互換性を考えると，バッチ処理方式では宣言は記述順序に従って宣言すればよい（JIS FORTRAN では，まずはじめに宣言しておかなければならない）が，宣言ステートメントを実行ステートメントにした場合には多少趣を異にする。記述順序ではなく，実行順序に従う

ことになるからである。この点に関しては，宣言ステートメントは，すべてJIS FORTRANのように前に置くようにしさえすればよい。行番号が使用者の制御下にありさえすれば，これは容易にできるから問題ない。しかも，宣言ステートメントを直接ステートメントとしても使用できるようにしておくと一層使いやすくなる。ここで強調しておかなければならないことは，あくまで会話形という点に重点を置いているということである。もちろん，バッチ処理方式のシステムとの互換性を考えると，バッチ処理のシステムにかけるときには，そのように修正する必要がある。

その他，一般的な要求事項としては次のものが考えられる。

- ある言語の会話形であると名乗るときには，たとえば PL／I の場合には動的な部分は OS に依存するから，致し方ないものもあるが，少くともブロック構造とか，IFステートメントで DO グループの使用を許すとか，あるいは構造を処理するとかいったいわゆる言語のもつ特性は生かしてもらいたいものである。

  たとえば，RUSH などはおこがましくも PL／I のサブセットであると名乗っているが，サブセットには違いないが pL／I と名乗れる程のものではない。FORTRAN と何ら変るところがない。

- JOSS の Form に相当するステートメントは会話形プロセッサでは採用すべきであろう（RUSH では IMAGE として採用している）。

例　Form 1:

```
x = 12.356
Type x,x,x in form 1.
  12     12.4     1.2 01
Type x,x,x,x in form 1.
I have too many values for the form.
```

すなわち，出力の仕方を絵で表現できるというところに意味がある。

この他，会話形プロセッサでは，絵で表現できるところはできる限り絵を採用すべきである。絵は，眼で見ただけで直ちに様子がわかるという人間に

とってこの上ない手段であり道具だからである。

会話形だからといって自由度が失なわれているような場合が多くみうけられるが，会話形だからこそ，かえって自由度を増すようにすべきである。

たとえば，RUSH では1行に1ステートメントしか書くことができない。言語がPL／I（といっても，前述の通りRUSHは一見FORTRAN風言語ではあるが）なのだから，言語の性質上，1行に複数個のステートメントを書くこともできれば，他行にわたるステートメントも書けるようにすべきであろう。この点に関しては，QUIKTRAN は端末にカード読取装置を置くこともできることから，1行だけ継続行を許している。

ハードウェアに依存している例を1つ最後に上げておこう。端末の使用者としては，結果の印刷の仕方が自由にできればできる程有難い。そこで，頁がえとか行がえとかが随意に指定できれば便利である。ただし，この程度のことは，使用者にちょっとした制約を加えれば，ソフトウェアで処理できないこともないが。

4.2 システムの応答

遠隔使用者に対するシステムからの応答の仕方は非常に重要な問題である。一般のバッチ処理方式のプロセッサでも，この点に関して問題はあるのだが，使用者は何かわからなければ，手近に居る専門家に相談すれば済むので，あまり重要視されていないようである。

バッチ処理方式の FORTRAN を例にとると，たとえば，COMMON ステートメント中のエラーに対し，〝この COMMON ステートメントには文法上の誤りがある〟といった類のエラー・メッセージを出すものが多い。一見もっともらしいが，こんなエラー・メッセージを出す位なら，エラー・メッセージは〝このステートメントは文法上正しくない〟，ただ1つだけで済ませた方がかえってすっきりする。また，バッチ処理方式では，以前のステートメントでのエラーが後々まで影響し，エラーがエラーを呼んでまさにエラーの花盛りといった現象がしょっちゅう起る。どこか1カ所訂正すると，やま程あったエラー・メ

ッセージは姿を現わさなくなる。このような後遺症的エラー・メッセージは，慣れていないとひどい目にあう。ところが幸いにして，バッチ処理では専門家が近くに居るから，何とか解決されている。

遠隔端末の使用者に対してはこうはいかない。システムの応答の仕方は，会話形システムでは最重要視しなければならないものの１つなのである。というのは，会話形システムにおいては，前にも述べたが，使用者の多くはしろうとであると考えなければならない。もちろん，この点に関してはシステムを開発する環境によって異るが，一般的にはこう考えるべきである。といって，専門家の面倒もみなければならない。一般に，遠隔使用者はひとりぼっちであるから，理想的なシステムとしては，これらピンからキリまでの使用者をそれぞれ満足させるべきである。

初めて誤りを犯したときには，システムが詳細に誤りを説明し訂正の仕方を示唆するようにすると使用者は感動する。しかし，同じながったらしいメッセージを何度も出されると使用者はイライラし始める。

理想的には，応答の量およびそのスタイルは使用者のその時の要求と一致させることが望ましい。しかしながら，実際には，使用者の経験の程度をシステム自身で決めることはむずかしい。そこで，システムによっては，システムの応答の冗長の程度をいくつか設けておいて，使用者に自分でそのいずれのものが自分に適当かということを指定させることによって，この問題を解決しようとしているものもある。前述の SDS 940 の初心者，未熟者，熟練者という段階にわけたのもこの例である。

その他の例としては，Massachusetts General Hospital の実験用システムがある。このシステムには，電子計算機からの質問，すなわち，使用者へデータを要求するとき，〃長文モード〃と〃短文モード〃の２つのモードが用意してある。たとえば，患者名を要求するとき，長文モードの時には〃PATIENT NAME〃と印字するが，短文モードの時には単に〃PN〃とだけ印字する。そして，使用者はいつでも随意にモードの切りかえができる。

同様に，処方プログラム（the Medication Order Program）では，〃教育モード〃のときには，〃処方した薬は肝臓の病気には使用してはいけな

い〃といったことを使用者に伝える。使用者が専門家で，この種のコメントはいらないと云うときには，このような警告は除くことができる。

また，教育用言語 PLANIT では，学生の正解率とか誤答率にもとづいて，次に進むべき教程を決めることができる。

こういったことを考慮に入れた上で，たとえば，エラー・メッセージなどはいろいろと工夫する必要がある。

会話形プロセッサの利点の1つに，前述のエラーがエラーを呼びまさにエラーの花盛りといった現象を排除することができるということがある。会話形プロセッサでは，使用者の行動を厳重に監視して，使用者の手続き等に誤りがあったときには，できるだけ早く，その旨を使用者に知らせるようにすべきである。使用者に対する応答は，完全でしかも解り易いものにすべきである。できることなら訂正の仕方も指示するようにすることが望ましい。バッチ処理の場合のように〃このステートメントは文法上正しくない〃といった類のメッセージは，会話形プロセッサでは適当でない。システムによっては，会話形でありながら〃?〃とだけ応答するものもあるが，これは避けるべきである。誤りの検出および診断に関しては，現在のところ，JOSS が最も親切のようである。

JOSS では，

1　印字の仕方が正しくない

2　JOSS の語彙にない語を使用している

3　数学的に書き方が正しくない

4　関数名，公式名，あるいは配列名と右カッコとの間に空白がない

5　〃1〃の代りに〃ℓ〃を，〃ゼロ〃の代りに〃o〃を，あるいは〃ドット〃の代りに〃ピリオド〃を用いている

6　Do 以外の動詞に for 句を用いている

7　伝送中にエラーがあった

といったものに対しては〃Eh?〃と応答し，それ以外の場合については，いままでの JOSSの例でも示したが，ちゃんとエラーの内容を指摘するようなメッセージで応答している。

例

```
1.234567876543  Set x=3.
Please limit step labels to 9 significant digits.
Type 1234.56789876.
Please limit numbers to 9 significant digits.
Set a(2345)=-17.
Index value must be integer and 1≤index≤250.
Do part 10*77.
Part number must be integer and 1≤part<10*9.
1.1  Type x.
1.2  Stop.
1.3  Type 2*5.
Do part 1.
Error at step 1.1: x ???
x=3
Go.
                x=          3
Stopped by step 1.2.
Go.
              x*5=         15
Go.
I have nothing to do.
Let f(a,b,c,d,e,f,g,h,i,j,k,l,m)=(a*b-c)*d+e.
Please limit number of parameter to 10.
```

（注）　下線の部分が JOSS からの応答

KEIO－TOSBAC TSS. の初期では，英字の小文字はシステムからのメッセージだけに用い，使用者は用いることができなかった。使用者が小文字を使ったときには，〝小文字を使ってはいけませんよ〟という意味の英文のメッセージが印字される（日本人として残念なことは，ある程度の英語を知らないと電

子計算機が使えないということである。ローマ字を使用すると，かえって始末が悪い。しかも，さらに悪いことには，システムによっては，何と，まともな英語を知っているだけでは何が何だかさっぱりわからないものがある）。ところが，毎度この長ったらしいメッセージを出していたら，使用者から苦情がでたので，初回だけはちゃんと出すが，2度目以降は〝Eh？〟とだけ印字する様にしてある。

遠隔情報処理システムでは，使用者が途中で何かわからないことが生じたら，随時，システムに相談できるようにしておくと非常に便利である。相談する相手が専門家であってもよい。理想的には，この両方を用意しておくべきである。前者の例としては，SDS 940 タイムシェアリング・システムにある HELP がある。

4.3 心理学的要因

いままでに述べた事柄も，その大部分のものは心理学的なものにもとづいている。何分，処理しているところを実際に見ていないので，一寸でも応答が遅れると心配になるものである。そこで，〝WAIT〟を出して，多少なりとも，安心させているシステムが多い。システムによっては，〝確かに万事うまくやっております〟と次の出力まで，端末で低い音を出させるものもある。また，実際に，その使用者の仕事を処理しているときだけ，その端末にあるライトを点灯させるものもある。General Motors Research Laboratory の DAC－1 システムでは，使用者の仕事とか要求を処理していることを低い音を出して知らせると共に，システムで現在処理しているステートメントのステートメント番号をディスプレイ装置に表示することまでしている。

〝WAIT〟の状態を解除して使用者に入力が可能になったことを知らせる際のメッセージとしては，通常，〝READY〟という言葉が使われている。このような頻繁に使用される言葉は，端末のほとんどはテレタイプとか電動タイプといった出力速度が遅いものなので，できるだけ簡単にすることが望ましい。マサチューセッツ工科大学の MAC 計画では，初期の頃はたとえば〝WAIT〟とか〝READY〟とか〝TIME〟と全綴りを印字していたが，途中から，〝W〟〝R〟，

〃T〃としか印字しないように改められた。RUSHでは〃_〃，PLANITでは〃*〃で，入力が可能になったことを示す。また，JOSSでは〃緑色のライト〃を点灯して，この状態を表示する。

とにかく，端末に居るのは生身の人間だから，いつ気が変わらないとも限らないし，本当に何をしでかすか予想することができない。そこで，ハードウェアおよび伝送施設に完全に依存することであるが，会話形システムでは，いついかなるときでも使用者が主導権を握れるようにしておくべきである。

端末で仕事中，ちょっとだけ離れる積りで席を立ち，そのまましばらくかえって来ない場合の使用者の処置なども問題になる。ダートマス大学のBASICシステムでは，２０分たっても使用者からの応答がなければ，システムでその回線を切ってしまう。

また，会話形システムの作成者は，ありとあらゆる点に気を配る必要がある。会話形システムの開発ともなると，必然的に人海作戦ということになるが，プロセッサを実際に作成する段階では，よほど文書化をしっかりしておき連絡を密にしないと，〃舟頭多くして舟沈む〃ということになりかねない。しかし，設計段階では，できるだけ多くの人の意見を聞くように努めるべきである。また，実際に作成にかかってからもいろいろと改良されるだろうが，これだけではまだ十分とはいえない。というのは，作成者はシステムの中味とか装置のクセなどは，すべて承知しているので，ディバッグ時にできる限りのことをしてみた積りでも，ある程度の常識の上に立って行なっているので，〃思いもよらない〃ことはまさしく思いもよらないからである。

4.4 処理方式

その場限りの計算を対象にするものは別として，一般に，会話形プロセッサはプログラムを作り上げ，ちょっとしたデータを用いてディバッグを行なうところまでが守備範囲であろうと思われる。実際の実行は，バックグラウンドとして存在する同一言語のバッチ処理方式のシステムで行なうのである。もちろん，出力量が少い場合は端末で実行させる場合もある。現在のバッチ処理では，何でもかでも，とにかく出力してしまうという傾向があるが，実際に必要とす

るものは案外少いものである。端末で実行させる場合もコマンドを用いてバックグラウンドで実行するよう指示できることが望ましい（端末は無人運転ということになる場合もある）。コンパイルに要する時間と通訳等に要する時間の関係によって，正確にはどちらで実行させた方がよいかという〝判断〟が必要だが，かなりの面倒をみる会話形のプロセッサと効率のよいバッチ形のプロセッサとを，いずれにせよ用意するべきであろう。

通常，会話形プロセッサとひとくちにいっても，処理方式は千差万別である。会話形プロセッサを作成するにあたって，まず問題になるのが処理方式である。実行方式を取り上げてみても，ソース・イメージを直接通訳実行するもの（例：RUSH）から，機械語のオブジェクト・プログラムを実行するもの（例：BASIC）まで様々である。そして，ソース・イメージの文法チェックも含め，綜合的な処理方式となるとますます数がふえる。

たとえば，BASIC の場合は会話形とはいうものの，ソース・イメージを入力している段階では何の処理も行なわず，実行を開始させて初めてコンパイルが始まり，文法チェックが行なわれると共に機械語のオブジェクト・プログラムが生成される。したがって，バッチ処理方式のプロセッサと会話形の編集用ルーチン（エディター）が用意されているものと思えばよい。

JOSS で格納コマンドを用いてプログラムを作成している場合も BASIC 同様単に格納されていくにすぎない。実行を指示して初めて文法チェックが行なわれるのだが，JOSS では機械語に落されることはなく，通訳実行という形をとる。

QUIKTRAN や KEIO では，ソース・イメージは入力と同時に文法チェックが行なわれ，中間言語に変換される。この段階で文法的な誤りを検出したときには，折り返しその旨を使用者に知らせ，正しい入力を待つ。実行を指示すると，この中間言語を通訳実行するという形をとっている。

この他にも，入力と同時にソース・イメージの文法チェックは行なうが中間語は生成せず，ソース・イメージをそのまま保存しておき，実行に際して機械語に変換するものとか，同様にソース・イメージを保存しておき，実行はソース・イメージを通訳実行するものとかいろいろなものがある。

会話形プロセッサでは，当然，ソース・プログラムを修正したり，ソース・プログラムのリストをとったりするので，プログラムの保存の仕方も重要である。BASIC や JOSS や RUSH では，ソース・イメージをそのまま保存しているが，QUIKTRAN や KEIO は中間言語だけを保存しておき，ソース・イメージは中間言語生成と同時に破棄してしまう。使用者からソース・プログラムのリストの要求があると，中間言語からソース・イメージを再生するという方法をとっている。この方法では定数の形がソース・イメージとは異なる場合があるし，Do の増分が省略されているようなときにもソース・イメージとは異なってしまうという欠点がある。しかし，補助記憶装置に制約があるときなどには，非常に有効である。さらに，KEIO は主記憶装置の容量が小さい（16K 語）こと，およびオーバーヘッドをできるだけ小さくしようとする意図から，プロセッサは純手続（pure-procedure または re-entrant）にしてある。

さらに，ステートメントの修正等の場合に関しては，応答をできるだけ早く行なう必要があることから，ソース・プログラムの保持の仕方はよほど工夫しておかなければならない。BASIC とか RUSH のようにソース・イメージをそのまま保存しておく場合には，たとえばソートなど何らかの方法を採用することもあるだろう。もちろん，QUIKTRAN とか KEIO のようにリスト構造を用いるのが普通である。リスト構造を用いた場合には，リストを手繰ったり，何らかの意味での〝ゴミ集め〟を行なう必要がある。応答速度を一定の範囲内に収めようとすると，これらのことが迅速に行なえるような手法を導入しなければならない（KEIOではこのためにいろいろな手立てを編み込んでいる）。

応答速度に焦点を合すと，使用者の側にある程度の制約を加える必要が生れてくる。会話形 FORTRAN を例にとると，この点に関してプロセッサ作成に影響をおよぼすものはきまっている。DIMENSION，FUNCTION および SUBROUTINE ステートメント，ステートメント関数定義式および Do 関係がこれである。すべて，応答速度を考慮した上でのことであり，プロセッサでどこまで面倒をみるかという問題である。

たとえば，DIMENSION ステートメントが削除されたり挿入されたりしたときを考えてみよう。まず，削除されたときにはどんな問題が生じるだろうか。

その配列名がまだ1度もプログラムで使用されていなければ問題はない。1回でもその配列名が使用されていたときには，その名前の処置が問題になる。その時点でプログラム全体を見渡せば，その名前は関数名に変じたことになったり，場合によっては変数名に変じたことになったりする。だからといって，そのどちらか一方に，その名前のもつ特性を変更するのは大仕事である上に，そもそも適切であるとは思えない。大方の場合，配列宣言し直すと判断する方が適切であろう。次に，挿入された場合にはどんな問題が生じるだろうか。この場合も，未だ1度もプログラムで使用されていないものが宣言されたのなら何も問題はない。以前に配列として宣言されていたもので，一担その宣言ステートメントが削除されて未定義になっていたときには，記憶場所の割り付け等に問題が生じる。さらに，次元数まで変更されたらどうすべきか。また，他の特性をもつものとして使用されているものが配列として宣言されたときにはどうすべきか。といった具合に，どこで，どの程度まで面倒をみるべきかという大問題が生じる。こういった問題を解決するために宣言ステートメントは，すべて会話形プロセッサの場合に限り，実行ステートメントにしてしまえということに関しては前に述べた。

以上，会話形プロセッサとはどういったものなのか，また作成するときにはどういった点に注意しなければならないか，あるいは，どういった事が問題になるのかという事柄について述べたが，この他にもまだまだ数多くの問題がある。さらに，ここでは主として使用者の立場から見た会話形プロセッサ観を述べたにすぎない。プロセッサ作成者の立場から見た，いわばミクロ的な会話形プロセッサ観も，そろそろ話題として取り上げてもよい頃かとも思う。しかし，会話形プロセッサでは外科的な問題よりも，むしろ精神科的あるいは内科的な問題の方がより重要なのである。

もっとも，姿，形は内部処理と無関係というわけにはいかない。具体的に実現可能なのかということがいつでもついてまわるからである。しかも，現実には，様々な制約のもとで実現させなければならないことから，ある意味では，外科的な問題の方がむしろ重要であるといえないことはない。

しかし，常に人間を相手に，しかも面と向い合って面倒を見なければならな

いのだから，会話形プロセッサでは，ちょっとしたことが大問題となることが多い。

たとえば，使用者が1字打ち損った場合，そのままにしておいて，わかりきったエラーをさそいだすか，場合によっては後からそのステートメントを削除すればよい。コマンドとか直接ステートメントだと，強引に実行させエラーをさそいだす以外にない。これは無駄である。そこで，1ステートメントを無効にすることができるような機能がどうしても必要になる。ところが，これだと長いステートメントの終りの方で打ち損ったら悲劇である。人によったら，1日がかりで1ステートメントやっと入力できたといったことにもなりかねない。そこで，1字抹消という機能はどうしてもほしい。ところで，1ステートメント無効とか1字抹消はどの段階で処理すべきか，こういった当り前のことが，誇張して云えばかなりの問題になりかねないのである。まず，編集用ルーチンで準備するようなところでは問題はないのだが……。方法論となるとさらに大問題。どういったやり方が一番いいのかわからないからである。1字抹消の例をとると，BASIC では"←"，TOSBAC—KEIO TSS では"#"を"伏字"にしておき，直前の1文字抹消という特殊機能を与えている。何文字か消したい場合には，その文字数だけこの伏字を印字する。ある程度打ってから，中程に打ち損いがあることがわかったときには，結局そこまで抹消して，新ためてそこから打ち直すか，場合によっては1ステートメント抹消しなければならない。JOSS では，この点を重要視して，ステートメント中任意の場所まで戻り"かさね打ち"によって，随時，修正ができるようになっている。しかし，かさね打ちをすると本当に正しく打ったのかどうか印字された文字をみて判読することがむずかしい。

以上のように，一見つまらないようなこと，簡単に思えるようなことが，何となく問題になるのが会話形プロセッサのもつ宿命とでもいうことができるだろう。こういったことは，これから，数多くの経験を積むに従って，段々と改善していかなければならない。経験を積むに従って，また，各種の技法が開発され多くの問題は解決されることは間違いないが，会話形プロセッサ故に特に問題となり，絶対に何とかしなければならない大問題がある。それは，"我々は日本人であるから「日本語」で使用することができるようにしなければならない"ということである。

```
HELLO
USER NAME ... N.DOI
SYSTEM ... KEIO
NEW OR OLD ... NEW
PROGRAM NAME ... TABLE
READY          19:21          1.27.61
%PROGRAM
  10 READY C   TABLE(GAMMA,ERROR,LOGE)
  20 READY     DO 10 I#1,10
  30 READY     A#FLOAT(I)/100.
  40 READY     B#GAMMA(A)
  50 READY     C#ERF(A)
  60 READY     D#ALOG(A)
  70 READY   10 PRINT 100,A,BC#,C,D
  80 READY  100 FORMAT(@ @,4F20.10,)
 ERROR NO.  21
  80 READY  100 FORMAT(@ @,4F20.10)
  90 READY     STOP
 100 READY     END
 110 READY %COMMAND
READY
%START
        0.0100000000        99.4325851202        0.0056416638
        0.0200000000        49.4422101636        0.0112822098
READY
%LIST
  10 C   TABLE(GAMMA,ERROR,LOGE)
  20     DO 10 I#1,10,1
  30     A#FLOAT(I)/100.
  40     B#GAMMA(A)
  50     C#ERF(A)
  60     D#ALOG(A)
  70   10 PRINT 100,A,B,C,D
  80  100 FORMAT (@ @,4F20.10)
  90      STOP
 100      END
END OF LIST
READY
%ALTER 80,80
  80 READY  100 FORMAT(@ @,4F12.6)
  81 READY%COMMAND
READY
%START 10
    0.010000    99.432585    0.005642   -4.605170
    0.020000    49.442210    0.011282   -3.912023
    0.030000    32.784998    0.016921   -3.506558
    0.040000    24.460955    0.022555   -3.218876
    0.050000    19.470085    0.028186   -2.995732
    0.060000    16.145727    0.033811   -2.813411
    0.070000    13.773601    0.039429   -2.659260
    0.080000    11.996566    0.045039   -2.525729
    0.090000    10.616217    0.050640   -2.407940
    0.100000     9.513508    0.056231   -2.302585
  90 STOP
READY
%DUMP
I     #          11
A     #   0.10000000E  0
B     #   0.95135077E  1
C     #   0.56231361E -1
D     #  -0.23025851E  1
 END OF DUMP
READY
%GOODBYE
TIME        19 54          USED-TIME      0 MIN.    0 SEC. 960 MSEC.
```

```
HELLO
USER NUMBER--020969
SYSTEM--BASIC
NEW OR OLD--NEW
NEW PROBLEM NAME--EX
READY
10  READ N
20  LET S=I#1
30  LET J#I↑2
40  IF N #J THEN 70
50  I#I+1
60  GO TO 30
70  PRINT J"↑2<" N"<#"I"↑2"
80  GO TO 10
90  DATA 8,10,36,50
100  END
RUN
ILLEGAL INSTRUCTION  IN 50
50  LET I#I+1
70  LET K#I-1
71  PRINT K"↑2<"N"<#"I"↑2
RUN
2↑2<8<# 3↑2
3↑2<10<# 4↑2
5↑2<36<# 6↑2
7↑2<50<# 8↑2
TIME:  0 SECS.
SAVE
```

```
TYPE 5+3,4.3,4*2,SQRT(2).
       5+3#      8
       4.3#     12
       4*2#     16
    SQRT(2)#    1.4142136
DELETE ALL.
LET  X#-B/(2.A).
LET  Y#SQRT(B*2-4.A.C)/(2.A).
A#1
B#3
C#-10
TYPE  X+Y,X-Y.
       X+Y#      2
       -X-Y#    -5
DELETE ALL.
1.1 DO PART 2 FOR X#0(0.1)10.
2.1 SET Y#F(X).
2.2 TYPE X,Y IN FORM 1.
FORM 1:
--.-  ........
LET  F(T)#EXP(-T*2/2)/SQRT(2.3,1415927).
DO PART 1.
  .0    3.989-01
  .1    3.970-01
  .2    3.910-01
  .3    3.814-01
   ...    ..
```

```
/*   GENERATION OF PRIME NUMBER PAIRS   */
200  START: PRIME,PAIR#7;
210         DEL#2;
220  NEWC:  IF DEL#2 THEN DEL#4; ELSE DEL#2;
230         PRIME # PRIME + DEL;
240  TEST:  DO D#3 BY 2 TO FLOOR(SQRT(PRIME));
250         IF MOD(PRIME,D)#0 THEN GO TO NEWC;
260         END TEST;
270         IF PAIR#PRIME-2 THEN PUT IMAGE (PAIR,PRIME)(PIX);
280  PIX: IMAGE;
FOUND ANOTHER PRIME PAIR FIRST IS -----, SECOND IS -----
290           PAIR#PRIME;
300           GO TO NEWC;
3100          STOP;
%XEQ START THRU ... ;
```

```
FRAME 1.00 LABEL#*HIST
2.  SQ.
*?
2.  SPECIFY QUESTION.
*WHO INVENTED THE ELECTRIC LIGHT?
*
3.  SA.
*A+THOMAS EDISON
*B ALEXANDER BELL
*
4.  SAT.
*A F: THATS VERY GOOD B:3
*B R:HE INVENTED THE TELEPHONE,TRY AGAIN...
*-R:
*-C:
*
```

```
FRAME 2.00 LABEL#*MATH
2.  SQ.
* LETS SEE WHAT YOU REMEMBER ABOUT TEMPERATURE. USING F FOR DEGRESS
*FAHRENHEIT AND C FOR DEGREES CENTIGRADE, WRITE THE FORMULA FOR
*CONVERTING FROM DEGREES FAHRENHEIT TO DEGREES CENTIGRADE.
*HINT:  F#9*C/5+32 CONVERTS FROU CENTIGRADE TO FAHRENHEIT.
*
3.  SA.
*O FORMULAS ON
*A+C#(5/9)*(F-32)
*B F#9*C/5+32
*C C#(5/9)*F-32
*
4.  SAT.
*A F:  B:7
*B R:YOUR ANSWER IS THE SAME AS THE ONE I GAVE YOU, TRY AGAIN...
*A F: NOW YOU'VE GOT IT.  B:15
*B R:YOU'RE STILL CONVERTING FROM CENTIGRADE TO FAHRENHEIT, TRY AGAIN...
*BC F:NOTE THE DIF FERENCE. C:B:OUT
*-R:
*-C:
*
```

```
 READY            17:19        2/1/69
PROGRAM
  10 READY C    FACTORIAL TABLE
  20 READY        DO 10 I#1,4
  30 READY       X#I
  40 READY       A#FACTOR##(I)
  50 READY    10 PRINT 100,X,A
  60 READY       STOP 77777
  70 READY       END
UNDEFINED STATEMENT NUMBER
100
  80 READY *COMMAND
READY
*ALTER 50
  51 READY  100 FORMAT(2F20.10)
  52 READY *COD#MM#ND
READY:
*TRACE
READY
*START
  20 TRACE  I       #        1
  30 TRACE  X       #  0.10000000E  1
  40 TRACE  A       #  0.10000000E  1
       1.0000000000          1.0000000000
  50 TRACE  I       #        2
  30 TRACE  X       #  0.20000000E  1
```

5. 会話形ＰＬ／Ｉの開発

5. 会話形ＰＬ／Ｉの開発

現在米国の商業用ＴＳＳで使用されている会話形言語のうち，５８％はFORTRAN，３２％はBASIC,残り１０％が２０もの違った言語にわけられている。

ＴＳＳの利用範囲は技術計算的なものが多いことと，バッチ（batch）用との互換性という意味でFORTRANが圧倒的に多い。

会話形言語はその目的からして，言語仕様のあまり複雑な機能は要求されないし，またビジネス用言語，例えばCOBOLなどは，会話形には不向きな言語であるというのが一般的な通説である。しかし，時代の趨勢からすると，事務計算や一般のデータ処理も，やはりＴＳＳ的に処理したいという要求は当然強くなるであろうし，またバッチ処理と同じように，FORTRANかCOBOLか，どちらか一方では不便だという仕事を，ＴＳＳのもとで行なうことも当然でてくると思われるので，この際，思いきってＰＬ／Ｉの会話形プロセッサを開発することに踏み切ったわけである。

会話形のＰＬ／Ｉとしては，現在Allen-BabcookのRUSHが有名である。そのほかＩＢＭのＣＰＳや，Stanford大学のＡＣＭＥコンパイラなどもある。いずれもＰＬ／Ｉとしては，かなり言語仕様をしぼったサブセット（Subset）であり，とくに技術計算用の機能に主体を置いている傾向がある。

このＣＰＬも勿論サブセットであるが，目的からいって技術用，事務用どちらの仕事も出来るようなサブセットをえらび，特にＰＬ／Ｉの大きな特長の一つとされているビット処理の機能は完全に含ませた。

6. 設　計　目　標

6. プロセッサの設計目標

会話形言語への一般的な要求としては次の様な項目があげられている。

(1) プログラムの作成，変更，ディバッグ，1部の演算実行などが随時，任意にきりかえられること。

(2) 仕事を中断しても，そこまでのファイルが保管されており，後刻その先が続けられること。

(3) 言語仕様，コマンド仕様とも初心者にも容易に使用でき，ガイダンスの機能もあること。

(4) また経験者や熟練者にも満足のゆくシステムである事が望ましいので，初心者用には，ある限られた範囲内の仕様で充分仕事ができ，それ以上のもっと凝った複雑な機能も使える様になっている事が望ましい。

(5) エラーメッセージやシステムとの応答は，バッチにくらべてかなりインタラクティブである事が必要である。エラーメッセージも初めのうちは，かなり丁寧に出て来て欲しいが，熟練者相手の場合は省略形でもよい。彼等にとっては，丁寧に長々とメッセージが打ち出されるより，相互のやりとりの迅速性をより尊ぶであろうから，できればメッセージのタイプを使用者がセレクトできるとよい。

(5) ダイレクトステートメント，ディバッグステートメントなど，会話形の特長を生かしたものが利用できるとよい。

(6) 現在会話形に使われている端末の多くは，タイプライタ程度のものであるから，かなりディフォールト（default）の機能を生かして，最少限の入力でもよい様にしておきたい。

(7) 端末操作の不なれからくる人間のミスが処理に影響しない様なシステムでなければならない。

(8) 入力データの入れかた，出力プリントのフォーマットの指定のしかたなど，端末の入出力機能をよくわきまえて設計すべきである。またデータの入力ステートメントが幾つかある場合，タイプインを要求するデータはどのステートメ

ントに対応するものであるかという事は知らせてくれないと困る。

これらの要求以外にも，実は会話形プロセッサのみに関する問題ではなくて，本質的には，オペレーティングシステム，あるいはそのタイムシェアリングシステム全般に対する要求，例えばハード，ソフトを含めたシステム全体の安定性，応答の迅速性などが密接にからんで来る。

さて以上の様な一般的要求の他に，このＣＰＬシステムで留意した諸点は次の様なものである。

(1) バッチ用ＰＬ／Ｉとの互換性をもたせた。

前述のように，このＣＰＬは，ＦＡＣＯＭ２３０－６０のＭＯＮＩＴＯＲ－Ｖシステムの下で働かせることになっている。そこで，バッチ用のＰＬ／Ｉコンパイラと互換性を持たせ，フォアグランド（ forground)で作成したＰＬ／Ｉのソースプログラムファイル（ Source program file)を，そのままバックグランド（ back ground ）のバッチ用ＰＬ／Ｉコンパイラに引き渡して，コンパイルさせる事もできるようにしてある。

会話形プロセッサはやはりインタプリタ処理が比較的多いので，実行時間がやゝかゝること，また会話形では大きなファイルはあまり扱い易くないので，比較的大きなプログラムや，データの多いものは，プログラムディバッグやテストランをフォアグランドで行い，ディバッグ済みのプログラムは，バックグラウンドのバッチ用コンパイラでリコンパイル（ recompile）し，完全なオブジェクトの形に作り直して本番の実行をさせる方が能率がよい。

(2) コマンドやエラーメッセージに日本語を採用

会話形というからには，できれば日常語とほゞ同じニュアンスで会話をしたいわけだが，残念ながらコンピュータ語の習慣から，今までは英語系の言葉がもっぱら使われて来た。

基本的には日本人向きの会話系言語はどうあるべきかという事がもっと検討されねばならないのだが，従来の言語との互換性，漢字を含めた日本語の扱いの困難性などの理由から簡単にはいかない。

このシステムではコマンドは英語と日本語の両方が使えるようにした。

エラーメッセージは，名詞や慣用的になっている言葉は英文を使うが，テニオハや文体は日本語に統一した。

(3) 演算スピードの向上

一般に会話形プロセッサはインタプリタ形式で演算実行するものが多いが，概してかなり時間がかゝる。かといってバッチ用のプロセッサのように完全にコンパイルしてオブジェクトを作ってしまうと，会話形の特長であるプログラムの変更，修正，中断，継続という様なかなり不規則な要求に対し，臨機応変に処理するのがむづかしくなる。

さきに開発したＦＯＲＴＲＡＮの会話形プロセッサは殆んどインタプリタ形式をとったが，かなり実行速度がおそいので，このシステムでは，コンパイル形式とインタプリタ形式の中間をねらう事とした。ステートメントによっては入力の段階で，かなりオブジェクトに近い形に落してしまうものもある。プログラムの変更や修正が，デクラレーションステートメントやブロック構造に関係のある場合は，チェインをたどって関連する部分を１つ１つあたってゆくよりは，思いきってリコンパイル（ recompile ）する方が早いと思われるので，そういった点で，バッチ処理のコンパイル方式のよさを充分活用することとした。

(4) ＰＬ／ＩをＰＬ／Ｉで書く試み

このＣＰＬの約７０％はＰＬＤ（ＰＬ／Ｉ for System Description）言語で書いてある。ＰＬＤはＦＡＣＯＭ２３０－６０用のシステム記述用ＰＬ／Ｉサブセットであり，このＣＰＬの開発に先だって，まずバッチ用のＰＬＤコンパイラを作り，その言語で会話形のＣＰＬを書くという２段構えとした。勿論このＰＬＤは今後，他のシステムの作成にも有効なものと考えている。

(5) 汎用の目的

ＰＬ／Ｉの言語のねらいは，もともと技術用，事務用をとわず，何でもできるという事であるが，このＣＰＬでもその精神は充分生かした。その上，ビットオペレーションをはじめ機械語的機能を生かし，システム記述用の目的にも使えるように考えている。

7. プロセッサの概要

7. プロセッサの概要

7.1 プロセッサの構成

会話形言語プロセッサＣＰＬは，マクロコマンド，コンパイラ，エグゼキュータ，オブジェクトプログラム，インタプリタ，ダイレクトステートメント処理，割込み処理より成る。

マクロコマンドは計算機利用者の資格検査やジョブ，ジョブステップに必要な情報を計算機利用者との応答によって，作成し（各情報は１つ１つ応答せずパラメータ形式で一括して与えることも出来る。），ジョブオープナ，ジョブステップオープナ，ジョブステップクローザ，ジョブクローザ等によってＣＰＬプロセッサの実行開始，終了を制御するものである。

コンパイラは計算機利用者が端末装置よりインプットするソースプログラムを翻訳し，オブジェクトプログラムを作成するものである。ＣＰＬでは，オブジェクトプログラムの実行を速めるため出来る限り機械語のオブジェクトを作成している。またｌｉｎｅ Ｎｏ．を計算機利用者がインプットすることによって，ステートメントの挿入，削除，交換等が自由に出来るがコンパイラはこれらに関する処置もしている。ダイレクトステートメントがインプットされた場合は，そのステートメントの翻訳が終了すると直ちにエグゼキュータに制御を移し，そのステートメントを実行する。

エグゼキュータはコンパイラによって作成されたオブジェクトの実行を制御するものである。これは前述のようにオブジェクトプログラムは出来る限り機械語に変換されてはいるが，直接オブジェクトプログラムに制御を移すことが出来ないからである。

また，ＣＰＬではｌｉｎｅ Ｎｏ．を指定することによりある特定のステートメントだけを実行させたり，ある範囲を実行させたりすることが出来るのでｌｉｎｅ Ｎｏ．の監視が必要である。その他，次に実行すべきステートメントがない場合や未定義のラベル，コンパイル時にエラーのあったステートメントを実行しようとした場合などにはその旨メッセージを出し，利用者の指示を待

つ。

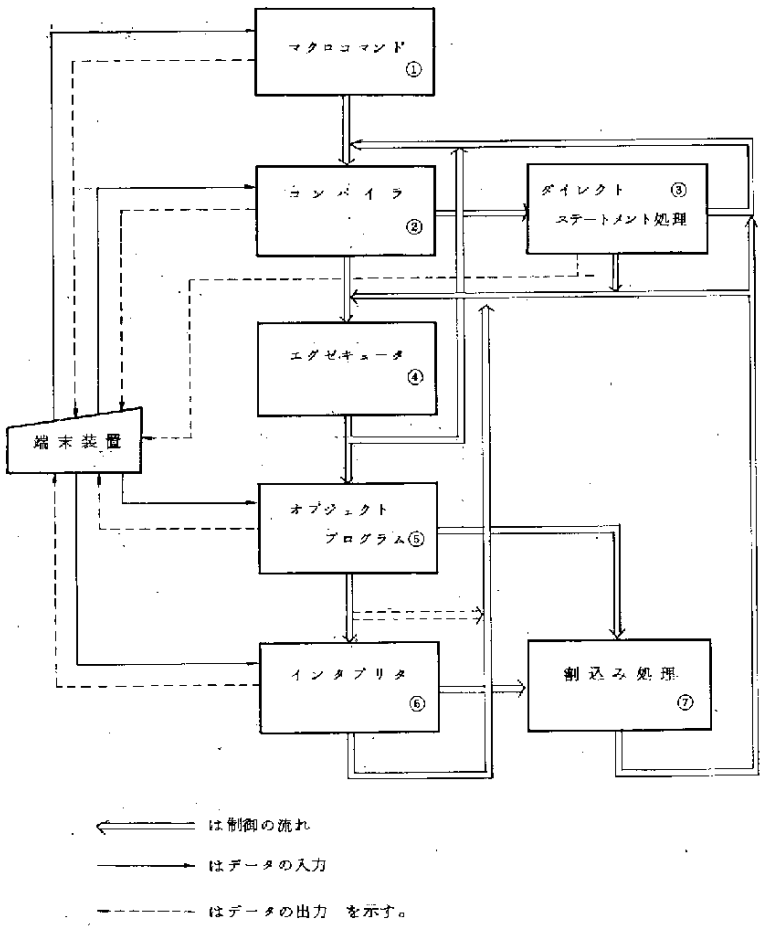
オブジェクトプログラムはコンパイラによってソースプログラムが翻訳されて作成されたものである。

インタプリタは入出力関係，あるいは代入ステートメントの１部など完全にコンパイルしオブジェクトをつくりにくいステートメントを中間言語におとしておきそれを実行時に翻訳実行するものである。

ダイレクトステートメント（direct state ment）処理は％ＲＵＮ，％ＬＩＳＴ，％ＳＡＶＥ等のダイレクトステートメントを実行するものである。

割込み処理はオブジェクトプログラム及びインタプリタ実行中に（演算結果等により）割込み条件が発生した場合その処置をするものである。

これらの関係を図示すると次のようになる。



① マクロコマンドは端末タイプライタからのマクロコマンド呼び出し指令によって，マクロコマンドライブラリから呼び出され，ここへ制御が移る。ジョブステップの開設に必要な情報を端末利用者との応答によって作成したあとジョブステップオープナを動作させる。また，ジョブステップクローザ（ＣＰＬの場合は，ＳＴＯＰステートメントを実行することによってジョブステップクローザを動作させる）の動作によって，このジョブステップオープナを動作させたステートメントの次へ制御が戻る。

② マクロコマンドのジョブステップオープナの動作によって，ここに制御が移る。端末タイプライタからインプットされたものがダイレクトステートメントにのみ指定できるステートメントであれば，ダイレクトステートメント処理へ制御を移し，そうでなければオブジェクトプログラムを作成する。ダイレクトステートメントであれば，すぐエグゼキュータへ制御を移すが，そうでなければ入力待ち状態となる。なお，インプットされたソースプログラムに誤りがあれば，直ちにその旨メッセージを出し入力待ち状態となる。

③ コンパイラで処理出来ないダイレクトステートメントが入力された時，ここに制御が移る。そのステートメントを実行後入力待ち状態となる。なお，ステートメントに誤りがあれば，直ちにメッセージを出し入力待ち状態となる。

④ コンパイラ，あるいはダイレクトステートメント処理からここへ制御が移る。指定された範囲の実行が終ればコンパイラへ制御を移す（入力待ち状態となる）。そうでなければ次のオブジェクトプログラムを実行する。次に実行するステートメントがない場合や，エラーステートメントを実行しようとした場合はメッセージを出し入力待ち状態になる。

⑤ オブジェクトプログラムは間接的にエグゼキュータに呼ばれる。このオブジェクトプログラムを実行中演算による割込みが発生すれば，割込み処理へ制御が移り，インタプリタが必要であればインタプリタへ制御が移る。１ステートメントの実行が終ればエグゼキュータへ制御を戻す。

⑥ オブジェクトプログラムの実行によってインタプリタが必要な場合，ここに制御が移る。インタプリタ実行中割込み条件が発生すれば割込処理へ制御を移す。また１ステートメントの実行が終ればエグゼキュータへ制御を戻す。

エラーステートメントを実行しようとした場合はメッセージを出し,コンパイラへ制御を戻し入力待ち状態となる。

⑦ オブジェクトプログラムまたはインタプリタよりここに制御が移る。ソースプログラムにONユニット指定があれば，そこへ一担制御を移すためにエグゼキュータへ制御を戻す。指定がなければコンパイラへ制御を戻し入力待ち状態とする。

7.1.1 オペレーティングシステムとの関連

(1) マクロコマンド

FACOM 230-60 MONITOR-V システムの下でデマンドジョブを行なうには，デマンドジョブの開始，終了を宣言するマクロコマンドが必要である。

マクロコマンドはジョブ制御定義文，直接作用文で作成され，マクロコマンドライブラリに登録される。

ＣＰＬにはマクロコマンド"ＣＰＬ"と"ＢＹＥ"の二つがある。詳細は外部仕様書（コマンドの種類と機能），内部仕様書（コマンド処理）を参照。

(2) システム制御コマンド

MONITOR-V に直接作用し，MONITOR-V の状態の変更，表示等を行なうシステム制御コマンドがある。これはＣＰＬに限らずすべてのデマンドジョブに共通なコマンドである。（ＴＳＳコマンド外部仕様書参照）

(3) 端末装置との通信

端末装置（タイプライタ）との通信にはデマンド用サービスマクロ

㋐ ＲＥＱＴＩＤ

㋑ ＷＴＵ

㋒ ＷＴＵＲ

㋓ ＲＦＵ

等を使用している。（デマンド用サービスマクロ参照）

(4) 作業領域と記憶保護

ＣＰＬは，リエントラント（Reenterable）であることを前提としてい

る。

そのため作業用領域はＣＰＬの実行時に必要に応じて，ＭＯＮＩＴＯＲ－Ｖに要求している。

作業用領域の要求，返却はマクロ命令

ＧＥＴＤＭ(get domain)

ＲＥＴＤＭ(return domain)

を使用する。

ＧＥＴＤＭマクロ命令は，主記憶上に作業領域（ブロックの整数倍）を開設する。（１ブロックは１０２４語）

制御プログラム（ＭＯＮＩＴＯＲ－Ｖ）は指定された大きさの領域を主記憶上にとり，指定されたベースレジスタ，Ｇレジスタに開設した領域の先頭番地，記憶保護のための領域番号等をセットする。

ＲＥＴＤＭマクロ命令は，主記憶上の指定した領域（ＧＥＴＤＭマクロ命令で開設されたもの）を解放する。解放された領域は，空領域となり，必要に応じて他の目的に使用される。

ＦＡＣＯＭ－２３０－６０の記憶保護はＧレジスタで行なわれる。ＧレジスタはＨＣＭ（高速磁心記憶）上ではG_1，G_2，G_3 レジスタの３個が使用可能である。

G_1，G_2 レジスタに設定されている領域は，それだけでは読み書き可能（即ち，その領域に記憶されたものを命令として実行することは出来ない。），G_3 レジスタに設定されている領域はそれだけでは読み実行可能（即ち，その領域に書き込むことは出来ない。）である。G_1，G_2 レジスタに設定された領域と G_3 レジスタに設定されている領域が同じであれば読み書き実行可能となる。

ＣＰＬでは上記条件によって，実行時に要求する作業用領域（読み書き可能）を２個開設することとした。（領域１，２と呼ぶことにする。）

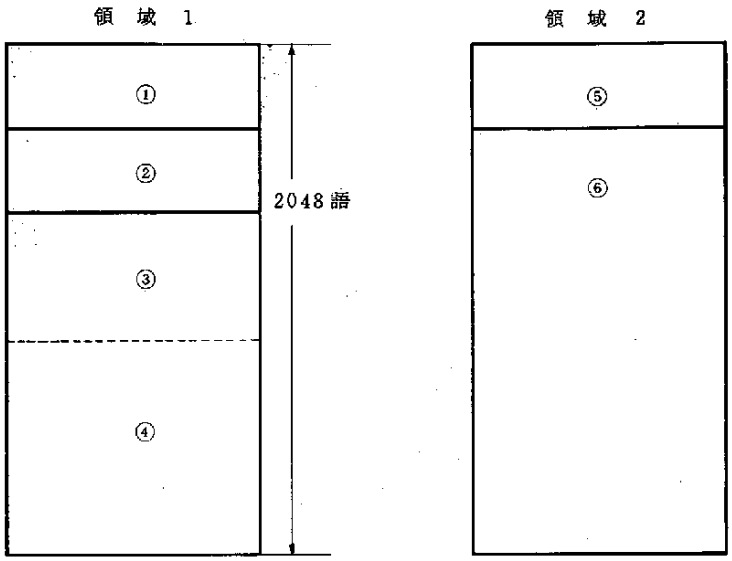
その使用目的は，

① Sourceプログラムと Objectプログラムを記憶するためのもの。

② Compiler，Interpreterに必要な作業エリア，Compile するた

めに必要な情報（テーブル）やSourceプログラムの変数名に対するエリア等を含めたもの。

とする。



領域１：

① 各種レジスタの一時保存や情報転送の際のパラメータ交換部

② 作業用エリア

③ 各種テーブル及びＳＴＡＴＩＣストレジ割り付けエリア

④ ＡＵＴＯＭＡＴＩＣ ストレジ割り付けエリア

領域２：

⑤ ①と全く同じ

⑥ Sourceプログラム，Objectプログラム用エリア

なお，③の領域が不足すれば④の部分をずらす，また領域１，２とも最初は２Block(２０４８語)であるが,実行中にエリアが不足すれば新しい領域を開設し，旧領域の内容を転写し，その后旧領域を返却する。これは前述のごとくある時点ではＧレジスタが３個のためＣＰＬ側から参照出来る領域は最高２個迄である。それ以上参照するには，ベースレジスタやG_1，G_2 レジスタを絶えず交換しなければならない。これではObject Programを出来るだけ機械語

に変換する意味がなくなるし，実行時間も遅くなる。

また，AUTOMATIC 属性をもつ変数は，それを含むブロックが始動された時にストレジが割り付けられるので関接アドレス指定を用いる。

領域1の参照には必ずベースレジスタ1を用いなければならない。割込み等が発生した場合は，SBORG，XCHB マクロ命令等によってベースレジスタを管理する必要がある。

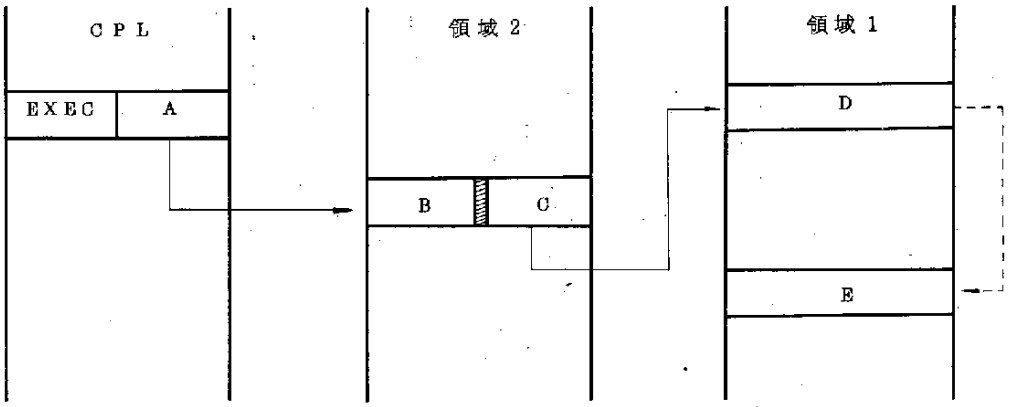
(5) Object Program の実行

CPLではObject Program の実行時間をなるべく速くするために，出来るだけ機械語のObjectを作成するようにしている。

しかし作業用領域の項で述べたように，Objectの出力エリアはRW領域であるのでObjectを直接実行することは出来ない。そのためRX領域（CPL側）にあるExecute 命令で実行させるようにしている。

機械語のObjectを出力することがむずかしいものについてはInterpreterへのJump 命令とパラメータを出力することにした。

これらの関係を図示すると



CPL側のEXEC命令のアドレス部によって領域2の機械語命令が指定され，その命令がさもそこにあったように（EXEC命令の所に）実行される。この時領域2の機械語命令のアドレス部によって領域1の"変数に対して割り当てられた番地"が参照される。また領域2の機械語命令のアドレス部が関接

指定であればアドレス部の示すアドレスの内容が実効アドレスとなる。

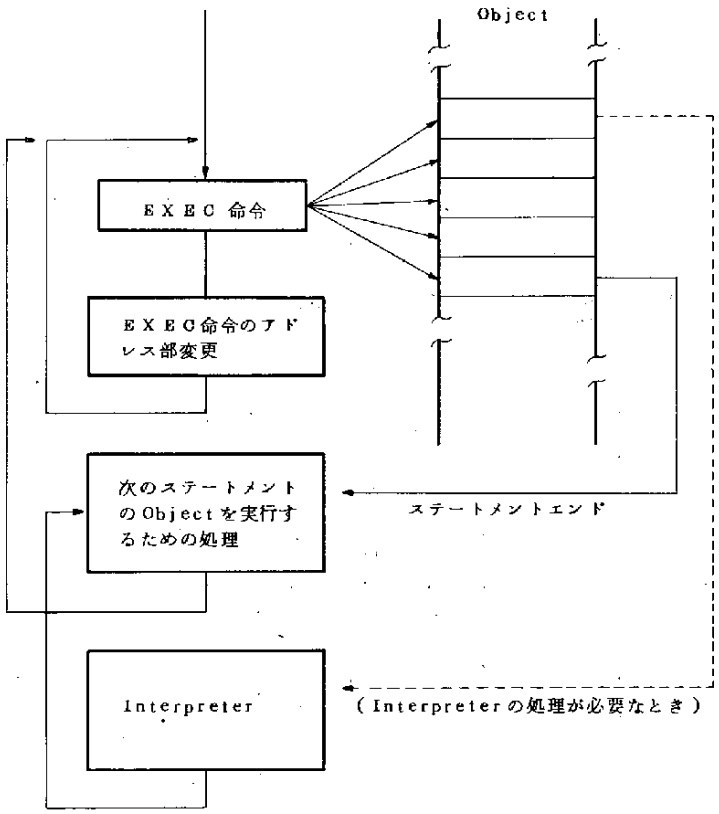
AはＥＸＥＣ命令のアドレス部で実行すべき命令の入っているアドレスを示す。

Bは，ＥＸＥＣ命令によって実行される命令部である。

Cは，参照すべきアドレスを示す。

Dは，Cに関接指定のない場合の実効アドレスである。

Eは，Cが関接アドレス指定の場合の実効アドレスであり，その場合は，CがDを指し，DがEを指すことになる。



アドレス部の変更はＣＰＬがリエントラントであるため直接ＥＸＥＣ命令のアドレス部へ書き込むことが出来ないのでIndex の内容を変更して行なう。

通常は１加えるだけである。

7.1.2 コンパイラ

会話形のコンパイラと普通の一括処理（batch）のコンパイラと比較すると会話形では次の点が特徴とされている。

1. コンパイル，実行が随時交互に切換えられること
2. プログラムのエラーが即時に検出できること
3. プログラムの修正および編集が任意に行えること
4. 任意のステートメントを実行させることが自由にできること
5. ソース・ステートメントをタイプ・インすると同時に実行する直接文が使用できること
6. プログラムを自動的に保存する（ＳＡＶＥ）ことができ，いつでも，呼び出して使用できること
   （LOAD）
7. あるプログラムから他のプログラム（保存されているプログラムであっても）を呼ぶことができること

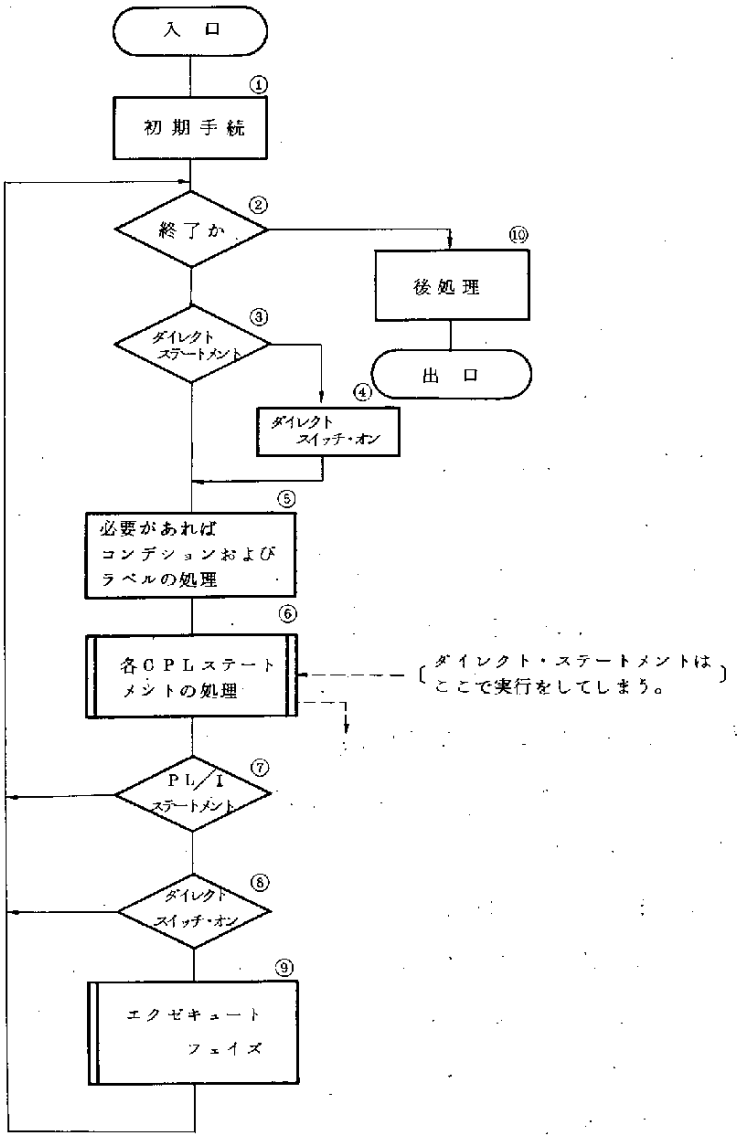
これらの性質を考慮し，このコンパイラでは次のような処理方針をとった。

1. ソース・プログラムの一部の修正，変更がオブジェクト・プログラム全体に及ぼさないようなオブジェクト・プログラムにする。
2. プログラムが完成しないうちに入力されているプログラムがいつでも実行できなければならないので逐次翻訳をする。
3. オブジェクト・プログラムの実行時における実行時間を短かくするためにはインタプリタで処理していては駄目である。かといって，ＰＬ／Ｉでは言語体系上，完全な機械語のオブジェクトを作成することは無理である。
   だから，出来るだけ機械語のオブジェクトを出すようにして，実行速度を上げるようにしている。
4. ソース・プログラムのリストの要求に応じられるようにソース・イメージもそのまま保存しておく
5. 文法的なエラーは出来る限り検出し，わかりやすい適切なメッセージを

出す。

ＰＬ／Ｉプログラムのエラーには，記述の誤りのようなそのステートメント自身を翻訳中に検出できるものと，ブロックが閉じないと（ＥＮＤステートメントが出現した時）検出できないものと，実行中にしか検出できないものとがある。

実行中のエラーについてはエグゼキュート・フェイズで解説する。



コンパイルの流れの説明

① コンパイラが呼ばれると，ＣＰＬステートメントを処理するのに必要なポインタおよびカウンタ類をセット，およびリセットする。

② もし，すべてのブロックが閉じていればそのプログラムは完成したものであるからコンパイルを終了する。

③ ステートメントの最初が　％であれば，ダイレクト・ステートメントであり　％以外であればインダイレクト・ステートメントである。

④ もしダイレクト・ステートメントであればダイレクト・ステートメントであることを示すスイッチをオンにする。

これは，ＰＬ／Ｉステートメントのダイレクト・ステートメントがコンパイルした後，ただちに実行しなければならないので，その解の処理に用いる。

⑤ 必要があればコンディションおよびラベルの処理をする。必要があればという意味は，コンディション接頭語およびラベル接頭語が用いられていればということである。

ダイレクト・ステートメントにはラベル接頭語を付けても意味がない。また，ライン番号についても同様である。

⑥ ここでは各ＣＰＬステートメントの処理を行う。

ＣＰＬステートメントにはインダイレクト・ステートメントとダイレクト・ステートメントがある。

⑦－⑨ ＰＬ／Ｉステートメントであって，ダイレクト・ステートメントであれば直ちに実行させなければならない。

実行させるにはエグゼキュータを呼んで部分実行させればよい。

ここで，部分実行とは１つのステートメントか複数ステートメントを部分的に実行させることをいう。

⑩ 外部手続きが閉じないと，処理することのできないラベルの未定義などの誤りの検出に関する処理また，実行するために必要な処理等を行う。

また，ダイレクト・ステートメントのＰＬ／Ｉステートメントを実行した場合はそのオブジェクト・プログラムは実行後不要なのでオブジェクト・プログラムの中からとりさってしまう。

この作業はポインタを下げるだけで良いと思われる。

各種テーブル類に登録された情報はそのままにしておくものとする。

7.1.3 エグゼキュータ

ここでは，コンパイラによって作成されたオブジェクト・プログラムを実行する。

1つのソース・ステートメントはインデックス，オブジェクト・プログラムおよびソース・プログラムに分解されているのでこのインデックスをたぐりながらオブジェクト・プログラムをExecute 命令によって実行させる。

オブジェクト・プログラムには完全な機械語になっているものと，インタプリタで実行をする通訳語から成っているものとがある。

機械語はそのままExecute 命令で実行できるが通訳語はそのまま実行できないのでインタープリートすることにより実行させる。

通訳語は次の形式をとる。

J　インタプリタ処理系番号

パラメータ　1

パラメータ　2

・

・

・

パラメータ　n

ここで

インタプリタ処理系番号とは，エグゼキュート・フェイスで

インタプリートすることにより，処理しなければならない処理系の通し番号で，この番号により，しかるべき処理系に制御を渡してやるためのものである。

パラメータにはアドレスかまたは，それ以外の情報が記憶される。

アドレスにはＩＤ table のその情報の入っている場所，Constantの情報の入っている場所，およびWorking storage の情

報の入っている場所を示すもの等がある。

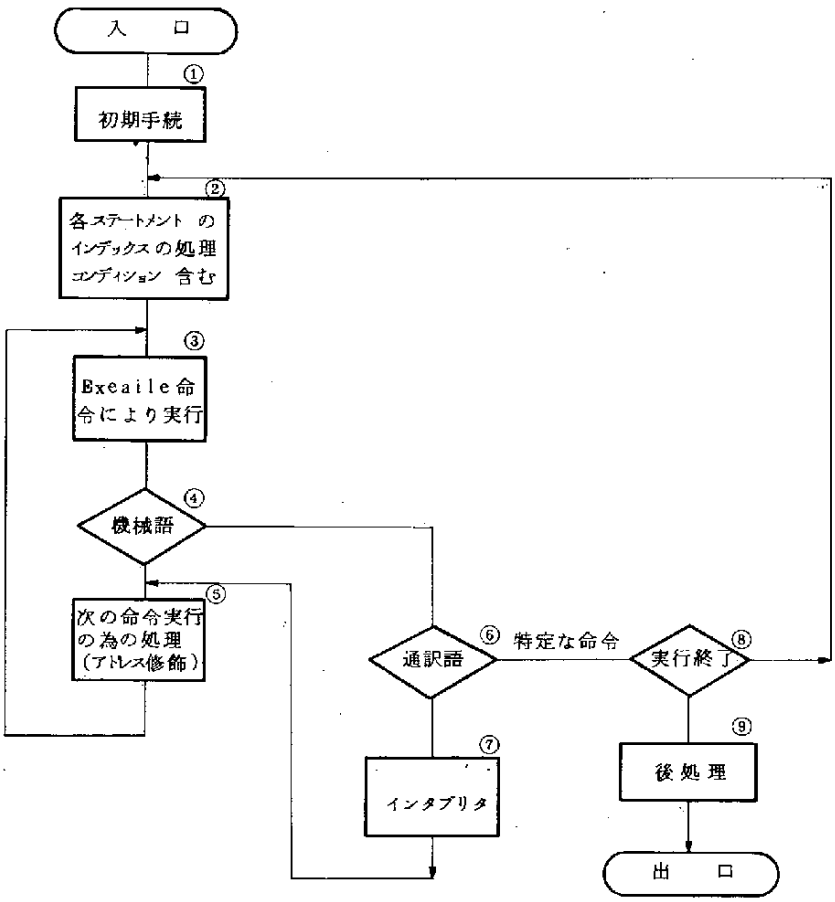
アドレス以外の情報とは例えば，パラメータの個数，属性等である。

ＣＰＬのステートメントの中には完全に機械語になってしまうもの，すべてが通訳語の形をとるもの，または両方ミックスした形になるものの３通りがあるが，いづれにせよ，各ステートメントのオブジェクト・プログラムの最後には，特定の命令が出力され，エグゼキュータに戻る様になっている。

通訳語の形をとるステートメントは，

属性が混合した場合の代入ステートメントとＧＥＴ，ＰＵＴ，ＲＥＡＤ，ＷＲＩＴＥなどの入出力ステートメントである。

エグゼキュータの流れ



エグゼキュータの流れの説明

① 各種初期手続きを行う。

② 次に実行するステートメントをインデックスによりたぐる。またこのときコンディション接頭語が指定されていれば，それに関する処理を行う。

次に実行すべきステートメントがPROCEDURE ステートメントであるときは，そのPROCEDURE ブロックの次のステートメントを実行する。

③ オブジェクト・プログラムをExecute 命令により実行する。

ここでExecute 命令とはHardware で持っている命令でアドレス部で示す命令があたかもExecute 命令のところにあったと同じように実行することができる命令である。

④－⑤ もし機械語であれば，その命令を実行した後次の命令の実行に関するアドレス修飾などの処理後，③へ行き，くり返す。

⑥－⑦ もし通訳語であれば，その通訳語のインタプリタ処理系番号により，それぞれのインタプリタに制御を渡し，インタプリートした後⑤へ行く。

⑧－⑨ オブジェクト・プログラムの最後に特定な命令があり，それを実行した場合⑧へくる。

もし指定されたステートメントをすべて実行していれば後処理をして実行を終了する。

また指定されたステートメントで未実行のステートメントがあれば②へ行き，くり返す。

エグゼキュータは，ダイレクト・ステートメントでPL／Iステートメントを指定した場合と　%RUNで実行を指定した場合に呼ばれる。

前者はそのステートメントのみ実行すればコンパイラに制御が渡される。

後者は実行指定されたすべてのステートメントが実行された後，前の状態に戻る。

エグゼキュータ設計上，次のような問題点がある。

(1) PROCEDURE ブロックを呼ぶときはPROCEDURE ブロックから戻ってきたときに実行するオブジェクト・プログラムに関する情報をStackしておく必要がある。

ＣＡＬＬ ステートメントでサブルーチン呼び出しの場合は次のステートメントだが式中から関数呼び出しの場合は，今実行中のインデックスとその相対アドレスが必要となる。

すなわち，いずれもインデックスとオブジェクト・プログラムの次に実行すべき命令の相対アドレスをＳｔａｃｋ すればよい。

但し，サーブルーチン呼び出しのときは相対アドレスはｚｅｒｏ となる。

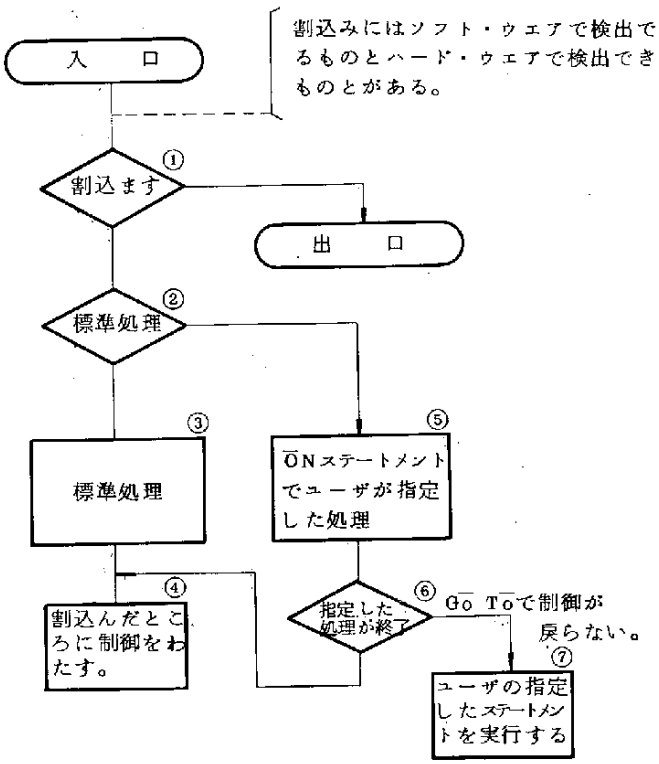
(2) 各種割込み処理

ＣＰＬで制御する割込みとその他の割込みに分けることができる。

ＣＰＬで制御する割込みとはＣＨＥＣＫ，ＳＩＺＥ，ＣＯＮＶＥＲＳＩＯＮ，ＳＵＢＳＣＲＩＰＴＲＡＮＧＥ，ＦＩＸＥＤＯＶＥＲＦＬＯＷ，ＵＮＤＥＲＦＬＯＷ，ＯＶＥＲＦＬＯＷ およびＺＥＲＯＤＩＶＩＤＥである。

その他の割込みとはこの８つ以外のもので，Ｈａｒｄｗａｒｅで発生するメモリーのパリティエラー等と各種スイッチ等ユーザが発生させたものとがある。割込み処理の流れを次に示す。

割込み処理



説　明

① ここでは割込みをおこすかどうかをしらべる。

SIZE, SUBSCRIPTRANGE および CHECK についてはユーザが割込みを指定しなければ割込まない。またその他のコンディションも前にNoを付けたコンディション接頭語を指定すれば割込みが起っても，何の処理も行なわない。

②—③ ユーザが割込みの処理を指定していなければ，標準の処理を行う。

たとえば，OVERFLOW であれば最大の値を代入して続行する。このとき，この主旨を示すメッセージは出力する。

④ 割込んだところに制御を戻す。

すなわち，割込みの原因になった命令は割込み終了後実行されないで，その次の命令から実行される。

⑤—⑦　ONステートメントでユーザが割込み処理を指定していればそれを実行する。

もし，Go To ステートメントで制御が戻らない場合は割込んだステートメントの実行は中断されっぱなしになるのでそのステートメントの実行結果は信頼することができない。

制御が戻る場合は④へ行く。

割込みで注意しなければならないのは，ユーザのプログラムで割込んだものと，処理系で割込んだものと区別しなければならないことである。

処理系で割込んだものは無視してやらなければならない。

(3)　Go To ステートメントで未定義ラベルの実行処理

CPLではプログラムが完成していなくても，また，プログラム中にまだエラーがあってもプログラムを実行させることができる。

だからGo To ステートメントのラベルが未定義のまま実行されてしまう場合がある。

そこで，ラベルの定義すべきところに特定な相対アドレスかまたは番号を入れて，未定義ラベルを実行したときの処理をする処理系に制御を渡すようにする。

(4)　続行不可能な状態の処理

演算の続行不可能な場合は，適当なメッセージを出し，ユーザの入力を待つ。例えば次のような場合がある。

○　次に実行すべきステートメントがまだ入力されていない。

○　不適当なアーギュメントとパラメータの受渡し

アーギュメントとパラメータの個数および属性などが正しく対応しているかをチェックし，もし誤りがあればメッセージを出す。

○　関数呼びこみで関数値が設定されなかった。

式中より関数手続きを呼びこみ，その関数の関数値が与えられなかったときメッセージを出す。

○　ポインタ修飾したポインタ変数の値がNull であった。

スカラポインタ変数—≫基底付き変数でスカラポインタ変数にポインタ

値が与えられていない場合

○　データ・リストとそれと対応する書式リストが適当でない。

書式リストとそれと対応するデータが適当でない。

データ・リストとそれと対応するデータが適当でない。

等，入出力に関する誤り。

7.2　プログラムファイルの形式

このコンパイラではステートメント単位に挿入（Insert），削除（Delete）および置換え（Replace）が出来る機能をもっているので，ステートメント単位にChained Sequencial にfileしている。

各ステートメントにはChain の情報とそのステートメントに関する情報とをもっている。

前者をINDEX（指標）と呼ぶことにする。

後者はSource Program と Object Program から成っている。

INDEX は次に示す5つの情報から成っている。

1. 次のステートメントのINDEX の場所
2. 前のステートメントのINDEX の場所
3. このステートメントのソースプログラムの場所
4. ステートメント シーケンス番号（各ステートメント毎に付いている番号）
5. コンディションに関する情報

1，2はPL／Iステートメントのオブジェクトが Chained Sequencial になっている為に次のステートメントのオブジェクトの場所，および前のステートメントのオブジェクトの場所を示すためのものである。

1はエグゼキュート・フェイズでステートメントを実行するときに必要である。

1および2は修正など編集をするときに必要である。

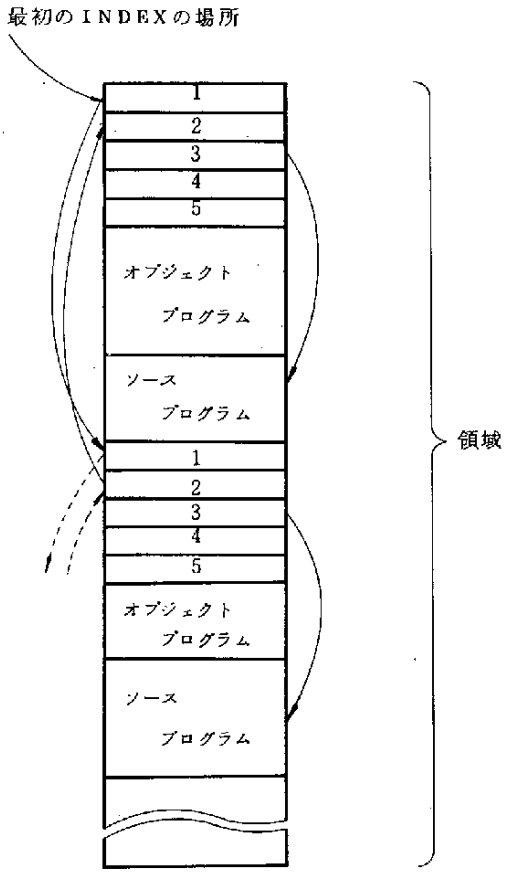
3は，このステートメントのソースプログラムの入っている場所を示すために必要である。

オブジェクト・プログラムは INDEXの直後から入り，その場所は固定しているので場所に関する情報は必要ない。

4は，ユーザが各ステートメント毎にタイプインする番号で0～999.99まで使用することが出来る。

5は，コンディション接頭語を処理するための情報である。

これらを図示すると次のように成る。



コンパイラの出力形式

1つのステートメントのソース・プログラムおよびオブジェクト・プログラムの量は可変である。

1つの領域にオブジェクト・プログラムとソース・プログラムが混って入って良いならば語単位に出力することができるが，オブジェクト・プログラムは機械語になっている部分が多いために全部まとまっている必要がある。

したがって一般に記憶する情報の少い方を他方が全部出力されるまでためておき，他方の出力がすべて終った時に，ためておいた情報を出力する。

PL／Iではソース・プログラムよりそのオブジェクト・プログラムの方がしめる語数が大であるから，ソース・プログラムをためておいてオブジェクト・プログラムが全て出力された後，ソース・プログラムをその後に出力する。

ただし，これは各ステートメント単位である。

また，配列式や構造体式などのように1つの式がいくつかのステートメントに分解されるようなものは，それぞれの処理系で説明するものとする。

7.3 モジュール構造

この処理系はモジュール構造を成している。

ここでモジュール構造を次のように定義しておこう。

モジュール構造とは，処理系に対してある種の変更があっても全体に影響をおよぼさないようにくふうされ，一部を変更すればよいようになっているサブ・プログラムを云う。

この種の処理系（システム・プログラム）を製作すると応々にして初期の設計思想から，いろいろと変更しなければならないことがしばしば生じる。

また，完成してからでも機能の追加，削除等と変更しなければならない場合がある。

このようなときに処理系全体に影響をおよぼし，全べてをコーディングしなおすのでは手間がかかって大変である。従って，このようなときに最小限にとどめるように考えておく必要がある。

たとえばName table の情報の記憶形式を変更するようなことが生じてもName table に情報を登録するルーチンや参照するルーチン等をまとめておけば，そのモジュールだけを変更すれば良い。

これは大規模な処理系を作成するときに大切なことである。

7.4 リエントラント・プログラム

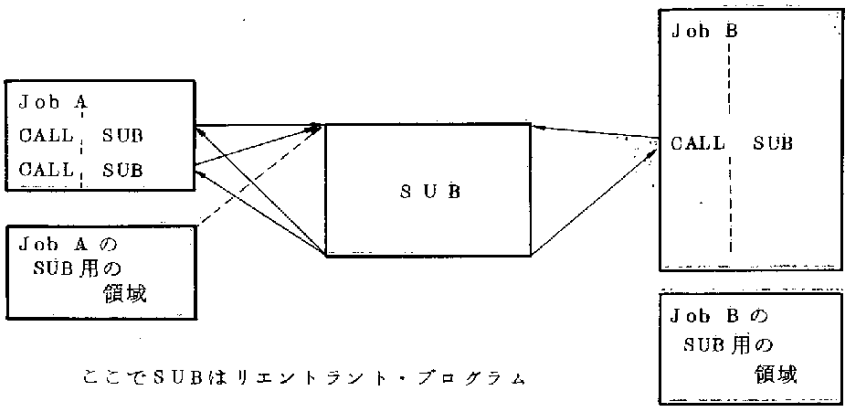
リエントラントなプログラムとは，2つ以上のプログラムから同時に利用することが可能なプログラムを云う。

たとえば，同時に複数個のプログラムから呼ばれたときに，その数だけのプログラムをロードィングするのは時間的にもメモリーの関係からも無駄が多くあまり好ましくない。これを避けるためにリエントラントなプログラムにするわけであるが，リエントラントなプログラムとは，そのプログラム自身を書き換えてはいけない。

すなわち，RX（読み，実行）でなければならない。だから，リエントラントのプログラムを呼んだプログラム側のプログラム領域に作業領域をとらなければならない。このようなリエントラントのプログラムは使用頻度が高いプロ

グラムに対して特に有効である。

この様子を図示すると次の様になる。



ここでSUBはリエントラント・プログラム

プログラムAとBからＳＵＢを呼んでいることを示している。

プログラムAもBも共に自身の領域にＳＵＢの為の作業領域を確保している。

7.5 編　集

ＣＰＬシステムではステートメントごとの挿入，削除および置換えができる。

編集はライン番号により行われる。

もし，以前にタイプインしたものと同じライン番号を後からタイプインすれば，前のステートメントのかわりに後のステートメントが置換え（Replace）られる。

また，以前にタイプインしたライン番号でないものを指定すれば挿入（Insert）される。

また，以前にタイプインしたステートメントを削除したければ

%ERASE(line no.〔THRU line no.〕)；

で削除してもよいし，1ステートメントならnullステートメントで置換えてしまってもよいであろう。

例　挿入　Insert

```
1     GET LIST(X, Y);
```

```
 2      Z = X + Y ;
1.1     PUT  LIST(X, Y);
結果      ↓Insert
 1      GET  LIST(X, Y);
1.1     PUT  LIST(X, Y);
 2      Z = X + Y ;
```

これは，ライン番号1と2の間にライン番号1.1のPUT ステートメントを挿入した様子を示している。

削除 Delete

```
1.  GET  LIST(X, Y);
2.  PUT  LIST(X, Y);
3.  Z = X + Y ;
%ERASE(2);
結果      ↓Delete
1.  GET  LIST(X, Y);
3.  Z = X + Y ;
```

これは，ライン番号2のPUT ステートメントを削除した様子を示している。

置換え Replace

```
1.  GET  LIST(X, Y);
2.  Z = A + B ;
3.  PUT  LIST(Z);
2   Z = X + Y ;
結果      ↓Replace
1   GET  LIST(X, Y);
2   Z = X + Y ;
3   PUT  LIST(Z);
```

これは，ライン番号2の代入文Z = A + BをZ = X + Yに置換えた様子を示している。

オブジェクト・プログラムをChained Sequencial にした訳は挿入，削除および置換えを簡単にするためである。

以下，それぞれの様子を図示する。

各ステートメントのオブジェクト・プログラムのＩＮＤＥＸ には前後のステートメントのオブジェクト・プログラムがどこにあるかの情報（ Address key ） を持っている。

このＩＮＤＥＸ によりステートメントが順に連続して結ばれていることになる。

この特徴は連続した一連の領域を必要とせず，またステートメントの挿入，削除および置換えのときもその前のステートメントのＩＮＤＥＸ の内容を変更するだけで，他の位置にオブジェクト・プログラムを移動する必要がない。

ステートメントの削除，挿入は次ページの如く行う。

ステートメントの置換えの場合は削除した後挿入すればよいので説明を省略する。

記号の説明

ST_x ： ステートメントの順番を示す。

I_x ： そのステートメントのＩＮＤＥＸ を示す。

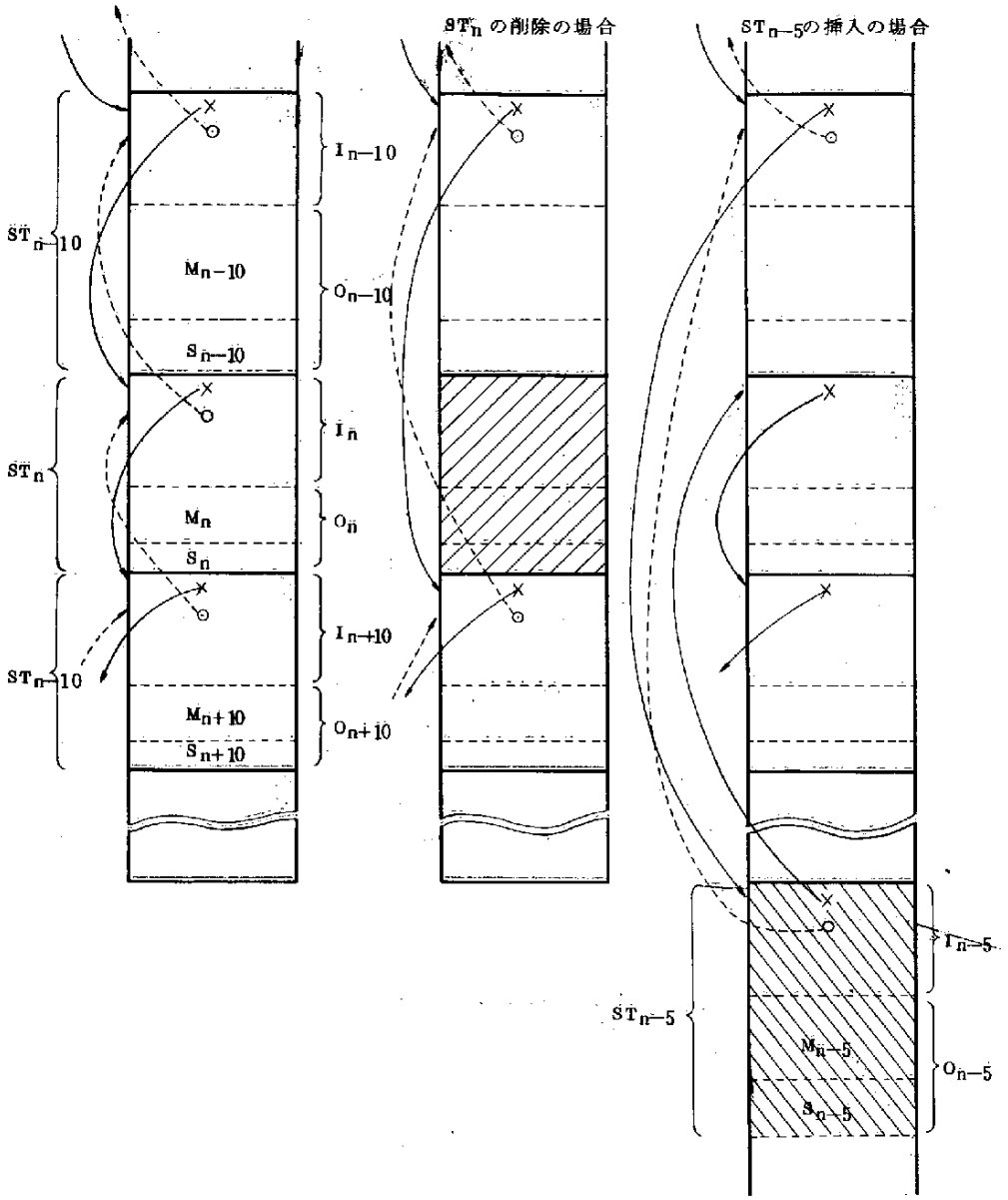
O_x ： 〃 Object を示す。

M_x ： 〃 Machine Languageを示す

S_x ： 〃 Source Statement 〃

x ： 次に続くステートメントのＩＮＤＥＸのアドレスを示す

○ ： 前の 〃


STn の削除の場合
STn−5の挿入の場合
In−10
Mn−10
On−10
Sn−10
STn−10
In
Mn
On
Sn
STn
In+10
Mn+10
On+10
Sn+10
STn+10
In−5
Mn−5
On−5
Sn−5
STn−5

8. 使用機器構成と記憶装置のわりあて

8. 使用機器構成と記憶装置の割り当て

- 使用機器構成

  図1に示す。

- 記憶装置の割り当て

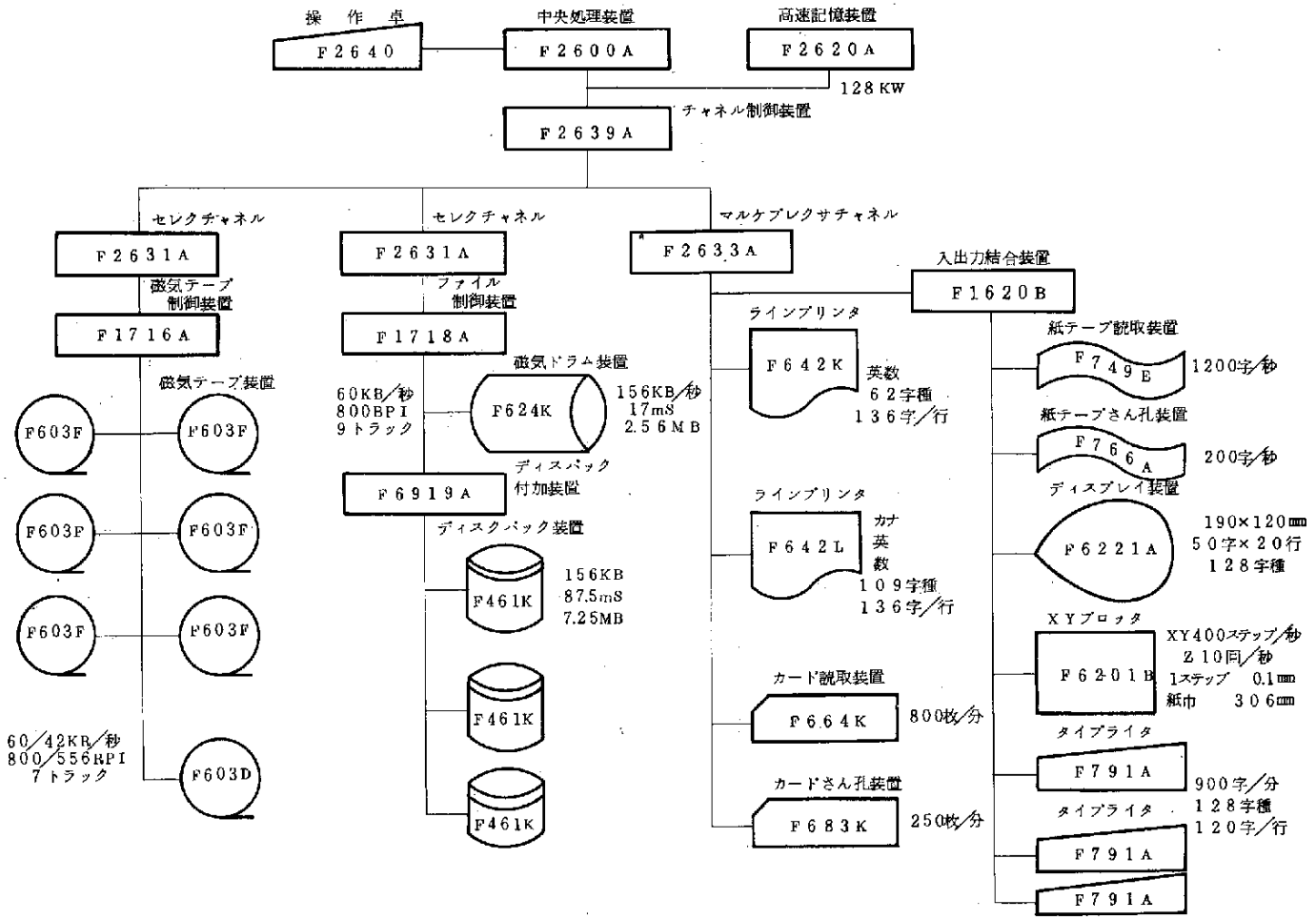
  主記憶装置

MONITOR—V	80K語
CPL	20K語

  外部記憶装置

ディスクパック	MONITOR—V専用
ディスクパック	CPLプロセッサ（基本関数，SAVEによるプログラムの保存）
ディスクパック	使用者ファイル用
磁気テープ	使用者ファイル用
磁気ドラム	CPLプロセッサおよび使用者ファイル用

図1　FACOM230－60機器構成



9. コマンドの種類と機能

9. コマンドの種類と機能

9.1 マクロコマンド

ＣＰＬによる作業を開始または終了するためには端末装置をコマンド待ち状態（端末タイプライタに"¥"を印字することによって解る）にして，ジョブの開始，終了を宣言するマクロコマンドを呼び出さなければならない。

マクロコマンドはジョブ制御定義文，直接作用文等で作成され，マクロコマンドライブラリに登録されており，それを呼び出すにはコマンド待ち状態でマクロコマンドに付けられた名前を打鍵すればよい。

マクロコマンドはマクロコマンド会話形トランスレータ（ＭＣＣＴ）によってインタプリティブに実行され，計算機利用者の資格検査やジョブ及びジョブステップに必要な情報を計算機利用者との会話によって作成する。それゆえ，計算機利用者はＭＣＣＴの要求に答えなければならない。

ＣＰＬには次の二つのマクロコマンドがあるのでそれらについて説明する。

① ＣＰＬ△〔利用者名，登録番号，ジョブ名〕〔，端末番号〕〔，ＦＩＬＥ＝ｎ〕 ㊐

② ＢＹＥ△〔ＡＣＣＯＵＮＴ〕 ㊐

利用者名：利用者の名前で，英字で始まる２６文字以内の文字列。

登録番号：利用者が登録した番号で，４文字からなる文字列。

ジョブ名：ジョブの名前で，１２文字以内の文字列。

ＭＯＮＩＴＯＲ－Ｖ の関係上同時に同じ番号をもつジョブがあってはならないので，ジョブ番号は，端末名＋利用者の整理番号というような形式が望ましい。

ｎ： 端末タイプライタ以外に入出力ファイルを扱う場合，そのファイルの個数を１０進整数で指定する。

端末番号： 各端末につけられた固有の番号

ACCOUNT： 計算機の使用時間をプリントする。

①が打鍵された場合マクロコマンドとの応答は次のようになる。（下線の部分はＴＳＳモニタまたはＭＣＣＴの出力である）

¥ＣＰＬ△〔利用者名，登録番号，ジョブ番号〕〔，端末名〕〔，ＦＩＬＥ＝ｎ〕 ㊞

ＹＯＵＲ　ＮＡＭＥ　＝　×××……×　㊞ ＴＯＲＯＫＵ—ＢＡＮＧＯ　＝　×××……×　㊞	利用者名，登録番号が省略されたとき。
ＵＮＲＥＧＩＳＴＥＲＥＤ　ＮＡＭＥ ¥	利用者名と登録番号が一致しなかった場合。
ＹＯＵＲ　ＪＯＢＮＡＭＥ　＝　×××……×　㊞	ジョブ番号を省略したとき。
ＦＤ　ＳＴＡＴＥＭＥＮＴ　１　＝　¥　ＦＤ　×××㊞ ・ ・ ・ ・ ・ ＦＤ　ＳＴＡＴＥＭＥＮＴ　ｎ　＝¥ ＦＤ ××××㊞	ＦＩＬＥ＝ｎの指定があったときのＦＤ文の作成。 ＦＤ文中のファイル定義名はＥＮＶＩＲＯＮＭＥＮＴ属性のオプションリスト中のＦＤ　ＮＡＭＥ＝ファイル定義名のファイル定義名と一致しなければならない。

ＪＯＢ　×××…×　ＯＰＥＮＥＤ　ＡＴ　ｙｙｙｙ.ｍｍ.ｄｄ　ｈｈ.ｍｍ.ｓｓ

ソースプログラムインプット可能状態

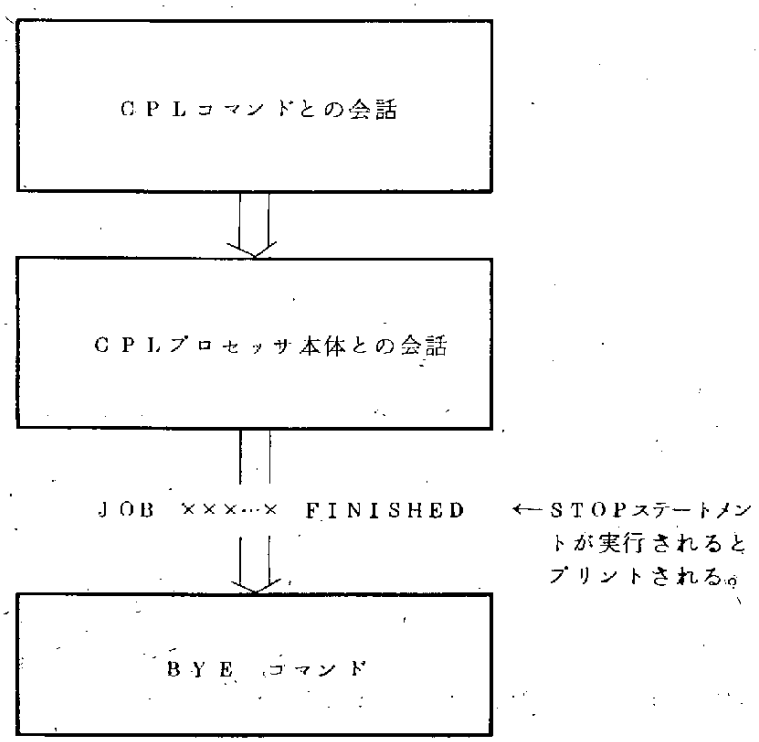
（プロセッサとの会話）

②が打鍵されたとき

¥BYE △〔ACCOUNT〕 ㊮

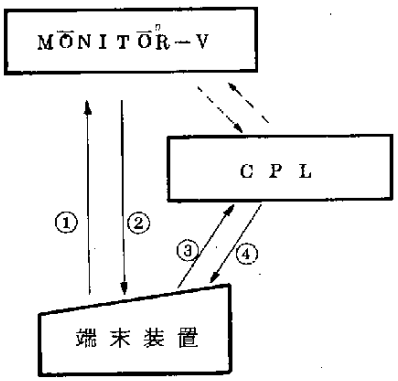
〔CPU USED mm.ss 〕 ACCOUNTが指定されたとき。

CPLコマンドとBYEコマンド迄の遷移は，



9.2 システム制御コマンド

システム制御コマンドとはMONITOR—V に直接作用するものである。システム制御コマンドと4.1で述べたCPLに対するコマンドとの関係を図で示すと次のようになるであろう。



①：システム制御コマンド

②：システム制御コマンドに対する返答

③：ＣＰＬに対するコマンド

④：ＣＰＬからの返答あるいは問合せ

図からも解るようにシステム制御コマンドとはある意味ではＣＰＬと関係のないものと言える。それはＣＰＬを経由しないで直接MONITOR—V と交信する訳けであるが，ＣＰＬの使用者であってもそれを使用することが出来るし，使用者にとっても有効な機能をもっている。なおここでいうシステムとはMONITOR—V を指す。システム制御コマンドについて補足すれば，それはコマンド待ち状態（¥が印字された状態）で実行され，システムの状態を変更したり表示したりするものである。なおシステム制御コマンドと呼ばれるものには次のようなものがある。（詳細はＦＡＣＯＭ２３０—６０　ＴＳＳコマンド外部仕様書参照）

① COMMENT △ 文字列 ㊎ （省略形 C△ 文字列 ㊎）

文字列をタイプライタに印字するだけで，他に何ら影響を与えない。

② DISPLAY △ T ㊎ （省略形 D△T ㊎）

打鍵した時の計算機内部の日付けと時刻を次のような形式で印刷する。

yyyy.mm.dd　　hh.mm.ss

年　月　日　　時　分　秒

③ KILL ㊎

現在実行中のデマンドジョブステップとマクロコマンドを強制的に終了させる。その後，このシステム制御コマンドを発した端末に対するシステムの状態はコマンド待ち状態となる。このままジョブを終了するにはジョブ終結コ

マンド（ＣＰＬの場合は＂ＢＹＥ＂）を打鍵する。

④　ＯＦＦ　㊉

この端末に対するシステムの状態をオフライン状態にする。オフラインの状態とはその端末がＭＯＮＩＴＯＲ—Ｖ と切り離されることで，利用者が再びその端末を使用するときは呼び出しボタンを押してＭＯＮＩＴＯＲ—Ｖ と接続してからでなければならない。

⑤　ＲＥＱＵＥＳＴ　㊉　（省略形　ＲＱ　㊉）

実行中のデマンドジョブステップに割り込むためのコマンドである。

このシステム制御コマンドは，この端末に対するシステムの状態がプロブレムプログラム実行中であるときにのみ使用出来る。

ＣＰＬの場合，このＲＥＱＵＥＳＴ が打鍵されると実行中のステートメントの処理は中断されステートメント（ダイレクトステートメントも含む）のインプット可能状態となる。

⑥　△　㊉

割込み釦を間違って押した場合に，直前の状態に戻すためのコマンドである。

以上①～⑥は特にＣＰＬと関係のあるコマンドであり，その他にも使用出来るコマンドがあるがそれらは前述のようにＭＯＮＩＴＯＲ—Ｖ　ＴＳＳ　コマンド外部仕様書を参照されたい。

また，コマンドの打鍵法やＣＰＬの使用法等は上記外部仕様書及びＣＰＬの使用法（第５章）を参照されたい。

10. PL／Iサブセット

10. PL／Iサブセット

10.1 サブセットの基本目標

すでに発表されているPL／Iサブセットには，例えば，ECMAサブセット，IBMのサブセット，電子協のMINIMUMサブセット などがある。

本来，PL／I 言語の大きな特長は次の4点にあるといわれる。

(1) 技術用，事務用を問わぬ汎用言語である。

(2) ビットやキャラクタのハンドリングが自由にでき，機械向き言語と同じ様な能力を持つ。

(3) マルチタスクのコントロール機能がある。

(4) プリコンパイル（precompile）でソースステートメントを一部アレンジする事ができる。

上記のいくつかのサブセットは，いづれもPL／I 本来の言語の特長をなるべく生かすよう工夫しているようだが，(3)(4)の機能は，大ていのサブセットでは除いている。また少し方向をかえたものとしては(2)の機械向き言語的な機能を強調し，特にシステムプログラム記述用のサブセットを作る試みも各所で行われている。

当センターではすでにFORTRAN, COBOL の両会話形プロセッサが使用可能であるが，このCPL ではこの両者の機能を統一したものとしてのPL／I機能と，一方システムプログラム作製の機会がかなり多いので，その目的のためにもこれを利用しようと考えている。

ただ残念なことに会話形プロセッサとしては，少し機能を欲張り過ぎているという批判もたしかにある。

このあたりは実用試験の結果，徐々に改めてゆきたいと思っている。

会話形プロセッサは，実際にはオペレーティングシステムの影響をモロに受ける。このシステムは既製のTSS モニタの下で働かせるため，(3)の機能，すなわち言語本来のマルチタスク処理は不可能である。また(4)のプリコンパイラに代るものとしてダイレクトステートメントを許した。

ＰＬ／Ｉの言語仕様から除いた，または制限した機能は次のようなものである。

1. 多重タスク
2. コンパイル時の機能の代りにダイレクトステートメント（direct statement）を許し，コンパイル時に実行させる。
3. 構造体の配列
4. 配列の断面
5. 添字つきの修飾された名前
6. 虚数定数
7. 属性としてはデータ属性，次元属性，ＥＮＴＲＹ属性，ストレジクラス属性，ファイル属性，リスト処理属性のみ許す
8. ＢＥＧＩＮ ブロック
9. 再帰的手続
10. ＡＢＮＯＲＭＡＬ 性と ＩＲＲＥＤＵＣＩＢＬＥ 性
11. ポインタなしのＣＯＮＴＲＯＬＬＥＤ変数

ＣＰＬの言語仕様は別冊「ＣＰＬの言語仕様」を参照されたい。なお次節に各ステートメントの一覧表をのせる。

10.2　ステートメントの種類とその制限事項の概要

ステートメント	ステートメントの一般形	制限事項
ALLOCATE ステートメント	ALLOCATE　基底つき変数名標　SET（ポインタ変数） 〔，基底つき変数名標　SET（ポインタ変数）〕…；	
(%)代入ステートメント （assignment）	オプション1（スカラ代入） {スカラ変数 / 擬変数} 〔，スカラ変数 / ，擬変数〕…＝スカラ式； オプション2（配列代入） 配列　〔，配列〕…＝ {配列式； / スカラ式；} オプション3（構造体代入） 構造体〔，構造体〕…＝構造体式； オプション4（ステートメントラベル代入） スカララベル変数〔，スカララベル変数〕…＝ {ラベル定数； / スカララベル変数；} オプション5（ポインタ代入） ポインタ数〔，ポインタ変数〕…＝ポインタ式；	
CALLステートメント	CALL　入口名〔（アーギュメント〔，アーギュメント〕…）〕	

ステートメント	ステートメントの一般形	制限事項
CLOSE ステートメント	CLOSE　オプション群〔，オプション群〕…； "オプション群"の形式　FILE（ファイル名）	
DECLARE ステートメント	DECLARE〔レベル〕名前〔属性〕… 〔，〔レベル〕名前〔属性〕…〕…；	
(%)DISPLAY ステートメント	オプション1　DISPLAY（スカラ式）； オプション2　DISPLAY（スカラ式）REPLY（文字変数）；	
DO　ステートメント	オプション1　DO； DO　WHILE（スカラ式）； オプション3　DO　変数＝指定； 指定の形式　式$_1$〔TO式$_2$〔BY式$_3$〕〕 〔WHILE（式$_4$）〕	
END ステートメント	END〔ラベル〕；	
FORMAT ステートメント	ラベル：〔ラベル：〕…FORMAT　書式リスト；	
FREE　ステートメント	FREE　〔ポインタ変数－＞〕基底つき変数名標 〔，〔ポインタ変数－＞〕基底つき変数名標〕…；	
(%)GET　ステートメント	GET　〔FILE（ファイル名）｜STRING（文字ストリング名）〕データ指定	

	データ指定の形式は次のいずれかである LIST データリスト DATA データリスト EDIT データリスト 書式リスト 〔データリスト 書式リスト〕…	
GO TO ステートメント	{GO TO / GO TO} {ラベル定数 / スカララベル変数} ;	
IF ステートメント	IF スカラ式 THEN ユニット1〔ELSE ユニット2〕 各々の"ユニット"は単純ステートメントでありセミコロンで終る	各々のユニットはラベルを持つことができない。
空(null) ステートメント	〔ラベル：〕…；	
ON ステートメント	オプション1 ON コンディションON ユニット オプション2 ON コンディションSYSTEM；	
OPEN ステートメント	OPEN オプション群〔，オプション群〕…； "オプション群"の形式は次の通り FILE(ファイル名)〔INPUT|OUTPUT〕〔STREAM|RECORD〕〔SEQUENTIAL〕〔BUFFERED〕〔PRINT〕〔LINESIZE(式)〕〔PAGESIZE(式)〕	

ステートメント	ステートメントの一般形	制限事項
PROCEDURE ステートメント	入口名：〔入口名：〕…PROCEDURE〔(パラメータ〔,パラメータ〕…)〕 〔OPTIONS(オプションリスト)〕 〔データ属性〕；	
(%)PUT ステートメント	PUT〔FILE(ファイル名)｜STRING(文字ストリング名)〕 〔データ指定〕〔PAGE〕〔SKIP〔(式)〕〕；	
(%)READステートメント	READ FILE(ファイル名) [INTO (変数)； SET (ポインタ変数)；]	
RETURN ステートメント	オプション1 RETURN； オプション2 RETURN(式)；	
SIGNAL ステートメント	SIGNAL コンディション；	
STOP ステートメント	STOP；	
(%)WRITEステートメント	WRITE FILE(ファイル名)FROM(変数)；	

備考 ステートメントに(%)の書かれたものはダイレクト ステートメントを示す。

10.3 ダイレクト ステートメント

ステートメントにはダイレクト ステートメント(direct statement)と，インダイレクト ステートメント(indirect statement, または collect statement)の2通りがあり，ダイレクト ステートメントは，それが入力されると直ちに実行され，オブジェクト プログラムの形であとには残らない。それに対し，インダイレクト ステートメントは，いわゆる普通のステートメントで，コンパイラによって，オブジェクト プログラムが作られ，%RUNの実行指令により，エグゼキュータで実行されるものである。

ダイレクト ステートメントの一覧表を次に示す。この一般形は各ステートメントの頭に%をつける。このうち，代入，DISPLAY, GET, PUT, READ, WRITE の各ステートメントは，ステートメント自身の働きは，普通のインダイレクト ステートメントと全く同じである。

使用目的としては，初期値の設定，データの臨時の入力，あるいはディバッグ用の出力などに便利である。

それ以外の%RUN, %SAVEなどは，ダイレクト ステートメント特有のものでジョブ制御ステートメントとも呼び，それぞれの目的でシステムを制御する働きをする。

ダイレクト ステートメントの処理は，そのための特別な処理ルーチンを特つわけではなく，コンパイル後，ただちにエグゼキュータに渡して実行させ，実行後は一時的にできたオブジェクトを消している。具体的には，ポインタの指示位置を変えるだけである。

ステートメント	ステートメントの一般形	機　　能	備　　考
% RUNステートメント	〔Ln〕%RUN〔Ln_1〔,Ln_2〕〕; （ジッコウ）	インプットしたプログラムの実行開始を指令する	
% LIST ステートメント	〔Ln〕%LIST;（リスト）	インプットされたソースプログラムのリストをとる	
% ERASE ステートメント	〔Ln〕%ERASE Ln_1〔,Ln_2〕; （ショウキョ）	インプットしたプログラムをLine NO によって削除する	宣言ステートメントが削除されるとその時点でインプットされているプログラムはRecompile される
% RESEQ ステートメント	〔Ln〕%RESEQ; （ギョウバンゴウツケナオシ）	Statement Sequence NO（Line NO）のつけ直しをする。Line NO は10.00 から１０きざみでつけ直しされる	端末装置がタイプライターの場合はその結果が印刷される
% SAVE ステートメント	〔Ln〕%SAVE name〔,KEY〕; （ホゾン）	インプットされたプログラム（ソースプログラム及びオブジェクトプログラム）を大記憶に転写する	SAVE されたプログラムは% LOAD あるいはCALL ステートメントで再使用可能である
% LOAD ステートメント	〔Ln〕%LOAD name〔,KEY〕; （トリダシ）	% SAVEステートメントによって大記憶に保存されているプログラムを主記憶へロードする	このステートメント以外のステートメントがインプットされた後でインプットされてはならない
% PURGE ステートメント	〔Ln〕%PURGE name〔,KEY〕; （トリケシ）	% SAVE ステートメントにより大記憶に保存されているプログラムを登録簿から取り消す	

%NAME LIST ステートメント	[Ln] %NAME LIST [PUBLIC / LIBRARY]; （ホゾンリスト）	USERがSAVEしているProgram name の一覧表をプリントする optionでPUBLICと指定すればpublicなprogram nameの一覧表をLIBRARYと指定すればlibrary programのnameの一覧表をプリントする	
%ENDCPL ステートメント	[Ln] %ENDCPL; （オワリ）	このステートメントが読まれるとただちにコマンド待ちの状態となる。 （STOP ステートメントの実行を待たずにコマンド待ちにして BYE をすることが出来る）	
%代入ステートメント %DISPLAY ステートメント %GETステートメント %PUTステートメント %READ ステートメント %WRITE ステートメント	これらのステートメントについては，10.2 ステートメントの種類とその制限事項を参照のこと ステートメントの一般形機能その他全く同じである。		

注 1. 上図で Ln, Ln_1 , Ln_2 は Line number を name は program name を示す。
注 2. ステートメントの一般形の欄にカッコでかこまれたカナ文字はダイレクトステートメント名標として英文字のかわりに使うことが許されることを示す。

11. プロセッサの使用法

11. プロセッサーの使用法

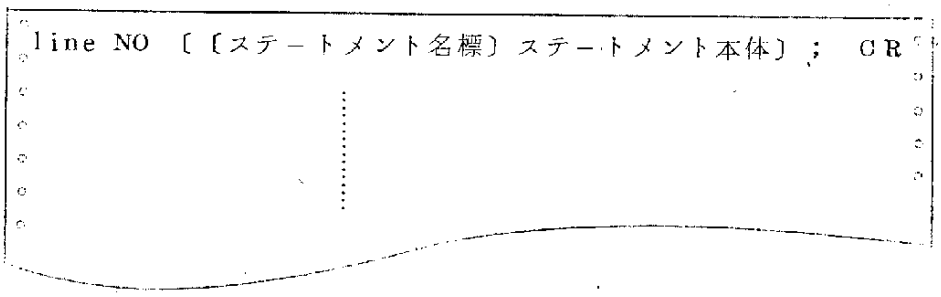
11.1 プロセッサのローディングとオペレーション

(1) ローディング

計算機利用者がこのＣＰＬプロセッサを使用する場合，最初に一連の端末操作（次の11.2節で説明される）を行なうことによりシステムはコマンド待ち状態である文字¥を印字する。その時，計算機利用者がＣＰＬと打鍵し，それに必要な情報をシステムに送ることによりＣＰＬプロセッサがローディングされ計算機利用者とＣＰＬプロセッサとの会話が可能な状態（システムより　ＪＯＢ　×××…×　ＯＰＥＮＥＤ　ＡＴ　yyyy. mm. dd　hh. mm. ss と印刷される。）となり，このことを入力待ち状態という。

(2) オペレーション

ＣＰＬプロセッサがローディングされ入力待ち状態にあるとき計算機利用者はＣＰＬ言語のソースプログラムを入力できる。line NO とソースプログラムの入力は次のような形式で打鍵する。



（注）　line NO の形式は××××〔・××〕△（ここで×は数字0～9であり，△はブランクである。）であり1.0から９９９９．９９まで表現できる。またline NO はステートメント単位につけるものとし，1つのステートメントが複数行になる場合，2行目以降はプロセッサよりline NO の部分にブランクが送られる。ダイレクトステートメントやコメントにつけられたline NO は無視される。

このような方法でプログラムを打鍵してゆき，計算機利用者が今まで入力したプログラムを実行する場合には％ＲＵＮのダイレクトステートメントを打鍵すればよい。そしてそのプログラムの実行中データを要求するステート

メントに来たときには＊INPUT DATA LINE NO ××××〔·××〕変数名Nが印字されるので必要なデータを打鍵しなければならない。またSTOP ステートメントが実行されるとJOB JOBNUMBER FINISHEDと印字される。

その他プログラムに誤りがあれば直ちにその旨メッセージが印字されるがその方法や形式についてはエラーメッセージの項を参照のこと。

次にＣＰＬプロセッサとの会話によるプログラム例を示す。

```
¥CPL
YOUR NAME = K.Y
TOROKU-BANGO = 1005
YOUR JOBNAME = MAX
JOB MAX OPENED AT 1969.03.20  14.30.10
      10 EX1:   PROCEDURE OPTIONS (MAIN);
      20        DECLARE D DECIMAL FIXED (5, 1),
                          X DECIMAL FIXED (5, 1),
                          Y DECIMAL FLOAT (8),
                          SMAX DECIMAL FLOAT (8);
      30        X = 0.0;
      40        SMAX = -1.0;
      50        GET LIST (D);
      60 L50 :  Y = SIN (X);
      70        IF(Y-SMAX) < =0  THEN GOTO L10;
      80 L20 :  SMAX = Y;
      90 L10 :  IF (X-5.0)  >= 0  THEN GOTO L40;
     100 L30 :  X = X + D;
     110        GOTO L50;
     120 L40 :  PUT LIST (X, SMAX);
     130        STOP;
     140        END EX1;
     150 %RUN
*INPUT DATA LINE NO 50 D
0.1
3.1     0.9995736E+00
JOB MAX FINISHED
¥BYE ACCOUNT
CPU USED 1.05
```

説　明

1. ターミナルはＣＰＬからＢＹＥまでの間がアクティブである。
2. アンダーラインのある語はプロセッサによるプリントであり，特に二本線のものはユーザの何等かの入力または答を要求するものである。
3. プログラムエラーは各ステートメントの直後にメッセージがプリントされる。
4. ＢＹＥの直後にオプションのACCOUNTを打鍵するとＣＰＬからＢＹＥまでの計算機使用時間をプリントする。

11.2　ターミナルの使用法

11.2.1　端末操作（各端末で仕事を要求する手順）

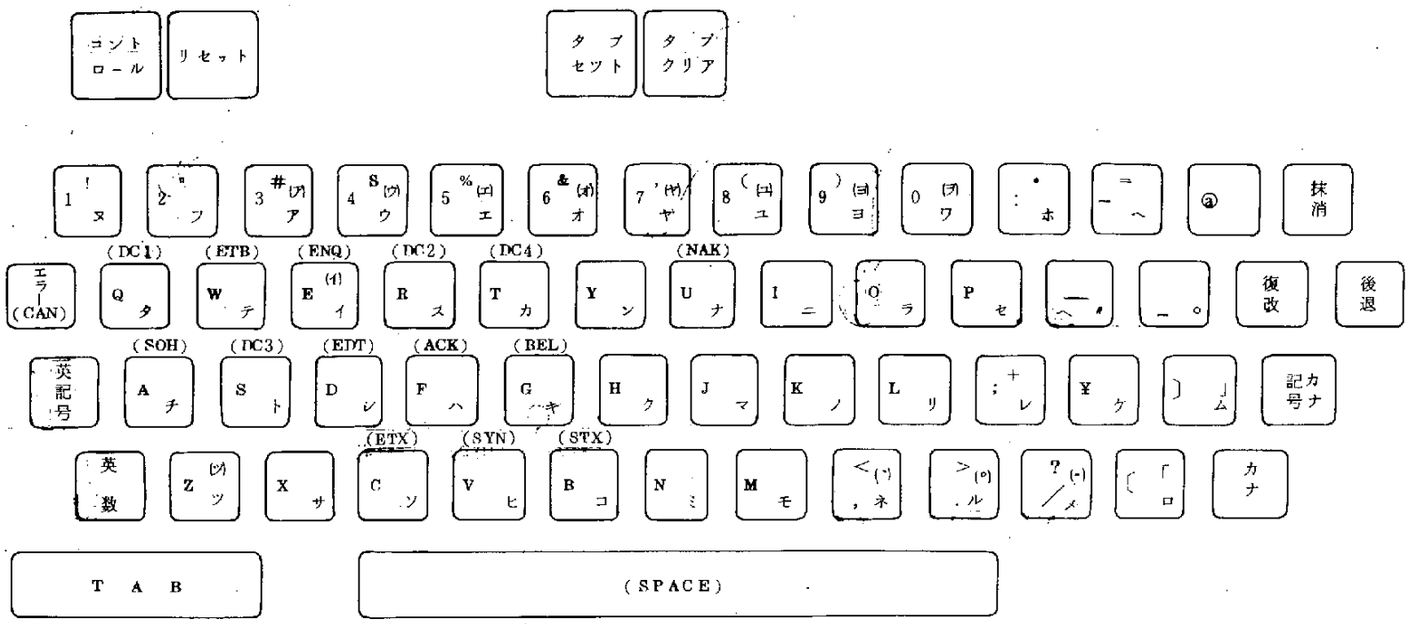
1. タイプライターの操作卓のPOWER釦を押す。（POWERランプが点灯）
2. CLEAR READY釦を押す。（CLEAR READYランプとMANUALランプ点灯）
3. AUTO MANUAL釦を押す。（MANUALからAUTOランプに点灯が移る）
4. REQUEST釦を押す。（システムから¥が印字されコマンド待ち状態となる）
5. ENTER ランプとけん盤可ランプが点灯したら。（ＣＰＬと打鍵）

ここで説明した各釦やランプについては図１を参照のこと。

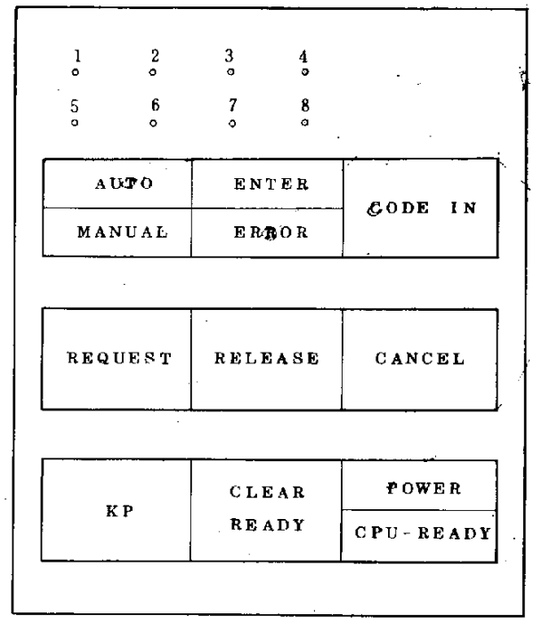
11.2.2　コマンドの打鍵法

(1)　システムの呼出しと割込み

計算機利用者が端末経由計算機を利用する場合には，まず端末装置に電源を入れる。この時この端末に対する計算機システムの状態はオフライン状態にある。この状態からコマンド待ち状態にシステムの状態を移すためには端末上のREQUEST釦（呼出し）を押す。システムがこの端末に対してコマンド待ち状態になったことを示すため，システムから端末に文字¥が送られ印刷される。この状態の時，計算機利用者はマクロコマンド呼



端末タイプライター操作ボタン



(図　　1)

出し指令またはシステム制御コマンドを打鍵することができる。

また端末に対するシステムの状態がオフライン以外の状態にあるとき，システムをコマンド待ち状態にするためにはやはり端末上のREQUEST釦（割込み……呼出しと兼用）を押す。

マクロコマンド（CPLまたはBYE）実行中に新らしいマクロコマンドを打鍵することはできない。あとから打鍵されたマクロコマンドは無視される。しかし，この状態の時システム制御コマンド（REQUEST, OFF, KILL 等）を割込ませることは可能である。この時，直前の状態はスタックされ，このシステム制御コマンドの実行が完了すると，中断された点に戻る。システム制御コマンドのKILLとOFFはマクロコマンドを強制終了状態とし，従ってスタックされた状態も消去される。その他システム制御コマンド実行中にシステム制御コマンドを割込ませることはできない。これは割込み信号が作用しないからである。

(2) コマンドの入力

コマンド待ち状態におけるコマンドの入力はこの状態にシステムがあることを表示するために，システムは文字¥を印字するゆえ，計算機利用者はこの¥に続けてマクロコマンド呼出し指令又はシステム制御コマンドを打鍵し，その最後にキャリジリターン（CR）を打鍵する。ただし次の行に継続させる必要のある場合にはハイフン（－）に続けてキャリジリターン（CR）を打鍵すればよい。

（注） ハイフン（－）キャリジリターン（CR）の組合せは計算機内部では無視される。
応答文中のスペースおよびタブは記号列中を除きすべて無視される。

(3) 誤字，誤文の訂正

① 誤字の訂正

誤字の訂正はバックスペース（例では BS と略記する）で行う。ただしバックスペースを行い，その上に新しい文字を打鍵すると読みにくくなるので，エラー（例では Ⓔ と略記する）を打鍵してもよい。ただし打鍵したバックスペースの数より多く Ⓔ 記号を打鍵すると，この行中このⒺより前にある全文字が消されてしまうから注意すること。

（注） ここでエラー記号とはＦ７９１Ａの［エラー（ＣＡＮ）］なる鍵盤のことであり

ＥＢＣＤＩＣコード（０１０１０００）に対応しタイプライター上には×印が印字される。

例１：

打　鍵　ＥＸ１：　ＰＲＯＣＧ　ＢＳ　Ⓔ　ＥＤＵＲＥ；　CR

印　字　ＥＸ１：　ＰＲＯＣ×ＥＤＵＲＥ；

システム　ＥＸ１：　ＰＲＯＣＥＤＵＲＥ；　CR

例２：

打　鍵　ＰＵＴ　ＬＥ　ＢＳ　ＩＳＴ（×）；　CR

印　字　ＰＵＴ　ＬＥＳＴ（×）；

システム　ＰＵＴ　ＬＩＳＴ（×）；　CR

例３：

打　鍵　ＤＩＳ　BS　ⒺⒺ ＳＰＬＡＹ（Ａ）；　CR

印　字　ＤＩＳ×ＳＰＬＡＹ（Ａ）；

システム　ＳＰＬＡＹ（Ａ）；　CR

この例のとき　BS　の数よりⒻの打鍵の数が多いと今まで打鍵された文字はすべて消去される。

② 行の訂正

行の訂正には前に説明したⒺを使用する。

Ⓔが複数個打鍵された場合は最初のⒺによりその行が消去され２つ目以降のⒺは無視される。

例１：

打　鍵　ＡＬＯＣＡＴＥ　Ⓔ　ＡＬＬＯＣＡＴＥ　ＶＡＬＵＥ　ＳＥＴ(Ｐ)；　CR

印　字　ＡＬＯＣＡＴＥ×ＡＬＬＯＣＡＴＥ　ＶＡＬＵＥ　ＳＥＴ（Ｐ）；

システム　ＡＬＬＯＣＡＴＥ　ＶＡＬＵＥ　ＳＥＴ（Ｐ）；　CR

例２：

打　鍵　RETURN;(E) (E) RETURN(A*B); (CR)

印　字　RETURN;×× RETURN(A*B);

システム　RETURN(A*B); (CR)

例３：

打　鍵　GET (E) (BS) (BS) (BS) (E) (E) (E) (E) PUT LIST (S); (CR)

印　字　GET × PUT LIST(S);

システム　PUT LIST(S); (CR)

③　コマンドの訂正

マクロコマンド呼出し指令およびMCCT との会話中の状態でMCCTからの問合せに対する応答が２行以上にわたり，かつ最後のキャリジリターン（ＣＲ）を打鍵する前に誤りに気がついたときは @ (CR) （アッドマークとキャリジリターンとの組合せ）により，現在打鍵中のコマンドを取消すことができる。

例１：

YOUR NAME=TAROO-YAMADA @ (CR)

YOUR NAME=TAROO・YAMADA (CR)

この場合(E)を用いて訂正してもよい。

YOUR NAME=TAROO-YAMAD(E) TAROO・YAMADA (CR)

例２：

¥ BYE ACOUNT @ (CR)

BYE ACCOUNT (CR)

この例のようにマクロコマンド呼出し指令の場合，訂正しても¥は印字されない。

(4)　打鍵の時期

前記のようにコマンドの打鍵はシステムより文字¥又は問合せのメッセージが送出され，それが端末上に印刷されたら直ちに打鍵を開始してよい。またプログラムの打鍵はプロセッサより入力待ち状態（つまりENTER

ランプとけん盤可ランプが点灯しているとき）ならいつでも打鍵できる。

(5) 割込み（QUIT）信号について

計算機利用者はプログラムの打鍵中や実行中，強制的にそれらを打切りたい場合が生ずる。その時にはREQUEST 釦（割込み……呼出しと兼用）を押すことによりシステムは¥を印字しコマンド待ち状態に移る。

11.3 プロセッサ使用上の注意事項

1. F791A 端末タイプライターは鍵盤が4段シフトなので各文字の打鍵ではそのシフトのことを十分注意する必要がある。
2. line NO は必ずユーザが打鍵しなければならない。
3. すでにSAVE されているプログラムがCALL ステートメントや関数呼出し等で使用されるとそのプログラムは外部ブロックに包含されるように処理され，そのプログラムに対しては改めてline NO がプロセッサでつけ直されるので外部ブロックのEND ステートメントのline NO の変更をユーザに要求する。
4. このCPL プロセッサーでは1行に1ステートメントが原則となっている（1ステートメントが何行にもなる場合は除く）がIFステートメントに限り1行に2ステートメントになることもある。

例：

```
50   IF  A=0  THEN  GOTO  END;ELSE  D=C+E;  CR
```

またELSE が2行目に打鍵され，しかもその前にline NO がつけられたときにはそのline NO を無視するというメッセージが印字されるので，後で改めてプログラムの変更等をするときに注意する必要がある。

例：

```
60   IF  A>B  THEN  GO  TO  L1;  (CR)
70   ELSE  D=C+E;  (CR)
W××××  LINE  NO  70  オ  ムシスル
```

11.4 エラーメッセージ

Compiler phase では，エラーのあったステートメントのすぐ次の行に，次のような形式でエラーメッセージがプリントされる。

＊＊＊＊ ERROR NO　　メッセージ　　＊＊＊＊

また，プログラムの修正や変更でリコンパイルを行なった時やEND 処理を行なわないと検出できないエラー，あるいは，実行時のエラー等は，line NO も必要となるので次のような形式でプリントする。

＊＊＊＊ LINE NO ERROR　　メッセージ　＊＊＊＊

例：

＊＊＊＊ 0102 ALLOCATE STATEMENTノ キジュツ ノ アヤマリ＊＊＊＊

なお，この例に示すように，ステートメント名標，名詞，慣用語等には英語を使い，用語やテニオハは日本語（カナ文字）を使った。

以下にSTATEMENT 毎にERROR NOとそのメッセージの一覧表を示す。

ステートメント	メッセージNo.	エラーメッセージ	備考
ALLOCATE ステートメント	0100	BASED VARIABLE = CONTROLLED STORAGE CLASS ガ センゲン サレテ イナイ	
	0101	BASED VARIABLE ガ SCALAR VARIABLE, ARRAY マタハ MAJOR STRUCTURE ノ イズレデモナイ	
	0102	ALLOCATE STATEMENT ノ キジュツ ノ アヤマリ	
ASSIGN ステートメント	0200	ASSIGMENT STATEMENT ガ ナガスギル	
	0201	STRUCTURE EXPRESSION ノ ソウタイテキ コウゾウガ コトナル	
	0202	SUBSCRIPT EXPRESSION ノ キジュツ ノ アヤマリ	
	0203	ARRAY NAME ガ センゲン サレタ シゲント コトナル	
	0204	( ) ノ タイオウ ガ アワナイ	
	0205	ARRAY ノ カッコノ タイオウ ガ アワナイ	
	0206	FUNCTION ノ カッコ ノ タイオウ ガ アワナイ	
	0207	BUILT-IN FUNCTION ノ ARGUMENT ノ コスウ ガ コトナル	
	0208	センゲン サレテイナイ QUALIFIED NAME オ シヨウ シテイル	
	0209	ASSIGNMENT STATEMENT ノ サヘンニ CONSTANT オ シヨウシテイル	
	0210	SCALAR EXPRESSION ノ SYNTAX ノ アヤマリ	
	0211	ARRAY EXPRESSION ノ SYNTAX ノ アヤマリ	

	0212	STRUCTURE EXPPESSIONノ SYNTAXノ　アヤマリ	
	0213	STATEMENT LABEL ASSIGMENTノ SYNTAXノ　アヤマリ	
	0214	POINTER ASSIGNMENTノ SYNTAXノ　アヤマリ	
	0215	ASSIGMENT STATEMENTノ　オワリガ　；　デナイ	
	0216	ASSIGMENT STATEMENTノ　キジュツ　ノ　アヤマリ	
CALL	0300	ENTRY-NAMEデナイモノ　ガ　CALLサレタ	
ステートメント	0301	ARGUMENTトPARAMETERノ　タイオウ　ガ　アワナイ	
	0302	CALL STATEMENTノ　キジュツ　ノ　アヤマリ	
CLOSE	0400	FILE NAMEガ　オカシイ	
ステートメント	0401	CLOSE STATEMENTノ　キジュツ　ノ　アヤマリ	
DECLARE	0500	センゲンサレタATTRIBUTEニ　ムジュン　ガ　アル	
ステートメント	0501	MULTIPLE DECLARATIONノ　アヤマリ	
	0502	QUALIFIED NAME　ノ　ヨビダシ　ガ　アイマイデアル	
	0503	DECLARE STATEMENTノキジュツ　ノ　アヤマリ	
NULL	1400	NULL STATEMENTノキジュツ　ノ　アヤマリ	
ステートメント			
DISPLAY	0600	CHARACTER-VARIABLEノ ATTRIBUTEガ　オカシイ	
ステートメント	0601	DISPLAY STATEMENTノキジュツ　ノ　アヤマリ	

ステートメント	メッセージNo.	エラーメッセージ	備考
DO ステートメント	0700	VARIABLE ガ SCALAR VARIABLE デ ナイ	
	0701	SCAR EXPRESSION デナイモノ ガ キジュツ サレタ	
	0702	DO STATEMENT ノ キジュツ ノ アヤマリ	
END ステートメント	0800	LABEL ガ ミティギ デ アル	
	0801	END STATEMENT ノ キジュツ ノ アヤマリ	
FORMAT ステートメント	0900	FORMAT LIST ノ キジュツ ノ アヤマリ	
	0901	FORMAT STATEMENT ノ キジュツ ノ アヤマリ	
FREE ステートメント	1000	BASED VARIABLE ニ CONTROLLED STORAGE CLASS ガ センゲン サレテイナイ	
	1001	BASED VARIABLE ガ SCALAR VARIABLE, ARRAY マタハ MAJOR STRUCTURE ノイズレデモナイ	
	1002	FREE STATEMENT ノ キジュツ ノ アヤマリ	
GET ステートメント	1100	DATA LIST ノ キジュツ ノ アヤマリ	
	1101	FORMAT LIST ノ キジュツ ノ アヤマリ	
	1102	コノ GET STATEMENT ニハ DATA SPECIFICATION ガ ナイ	
	1103	FILE ガ STREAM INPUT FILE デナイ	
	1104	NUMERIC DATA デナイモノニ P FORMAT ITEM ノ シテイ ガ アル	
	1105	LABEL ガ ミティギ デ アル	

	1106	GET STATEMENTノ キジュツ ノ アヤマリ	
GO TO	1200	LABELガ ミティギ デ アル	
ステートメント	1201	GO TO STATEMENTノ キジュツ ノ アヤマリ	
IF ステートメント	1300	UNITノSTATEMENTノ キジュツ ニ アヤマリ ガ アル	
	1301	IF STATEMENTノ キジュツ ノ アヤマリ	
ON ステートメント	1500	ON UNITがSINGLE SIMPLE STATEMENTデナイ	
	1501	ON STATEMENTノ キジュツ ノ アヤマリ	
OPEN	1600	フカ サレタFILE ATTRIBUTEニ ムジュン ガ アル	
ステートメント	1601	LINESIZE OPTIONガSTREAMPRINT FILEデナイモノニ シテイサレタ	
	1602	PAGESIZE OPTIONガSTREAMPRINT FILEデナイモノニ シテイサレタ	
	1603	OPEN STATEMENT ノ キジュツ ノ アヤマリ	
PROCEDURE	1700	DATA ATTRIBUTE ニ アヤマリ ガ アル	
ステートメント	1701	PROCEDURE STATEMENTノ キジュツ ノ アヤマリ	

ステートメント	メッセージ№	エラーメッセージ	備考
PUT ステートメント	1800	DATA LISTノ キジュツ ノ アヤマリ	
	1801	FORMAT LISTノ キジュツ ノ アヤマリ	
	1802	FILEガ STREAM OUTPUT FILEデナイ	
	1803	PAGE OPTIONガ PRINT FILEデナイモノ ニ シテイ サレタ	
	1804	SKIP OPTIONガ PRINT FILEデナイモノニ シテイ サレタ	
	1805	LABELガ ミティギ デ アル	
	1806	PUT STATEMENTノギジュツ ノ アヤマリ	
READ	1900	FILEガ RECORD INPUT FILEデナイ	
ステートメント	1901	INTO(VARIABLE) OPTIONノ キジュツ ニ アヤマリガ アル	
	1902	READ STATEMENTノ キジュツ ノ アヤマリ	
RETURN	2000	( )ノ タイオウ ガ アワナイ	
ステートメント	2001	RETURN STATEMENTノ キジュツ ノ アヤマリ	
SIGNAL	2100	CONDITION NAMEガ オカシイ	
ステートメント	2101	SIGNAL STATENTノ ギジュツ ノ アヤマリ	
STOP	2200	STOP STATEMENTノキジュツ ノ アヤマリ	
ステートメント			
WRITE	2300	FROM(VARIABLE) OPTIONノ キジュツ ニ アヤマリ ガ アル	
ステートメント	2301	FILEガ RECORD OUTPUT FILEデナイ	

	2302	WRITE STATEMENTノ キジュツ ノ アヤマリ	
%RUN	2400	LINE NOガ オカシイ	
ステートメント	2401	%RUN STATEMENTノ キジュツ ノ アヤマリ	
%LIST ステートメント	2500	%LTST STATEMENTノ キジュツ ノ アヤマリ	
%ERASE	2600	LINE NOガ オカシイ	
ステートメント	2601	%ERASE STATEMENTノ キジュツ ノ アヤマリ	
%RESEQ ステートメント	2701	%RESEQ STATEMENTノ キジュツ ノ アヤマリ	
%SAVE	2800	NAMEノ キジュツ ガ オカシイ	
ステートメント	2801	KEYノ キジュツ ガ オカシイ	
	2802	%SAVE STATEMENTノ キジュツ ノ アヤマリ	
%LOAD		NAME(KEY)ガ SAVE サレテ イナイ	
ステートメント		%LOAD STATEMENTノ キジュツ ノ アヤマリ	
%PURGE		NAME(KEY)ガ SAVE サレテ イナイ	
ステートメント		%PURGE STATEMENTノ キジュツ ノ アヤマリ	
%NAME LIST		%NAME STATEMENTノ キジュツ ノ アヤマリ	
ステートメント			
%END CPL		%ENDCPL STATEMENT ノ キジュツ ノ アヤマリ	
ステートメント			

参　考　文　献

1) George O. Collins, Jr. : "Interim Technical Documentary Report, A Study of the Remote Use of Computers," TR-66-679-1, Informatics Inc., Sept., 1966.

2) George O. Collins, Jr. : "Interim Technical Documentary Report, A Study of the Remote Use of Computers," TR-66-679-2, Informatics Inc., Dec. 1966.

3) George O. Collins, Jr. : "Final Report on A Study of the Remote Use of Computers, "TR-67-679-1, Informatics Inc., Jan. 1967.

4) J. C. R. Licklider : "Man-Computer Symbiosis, "IRE Transactions on Human Factors in Electronics, Vol. 1, HFE-1, March 1960, pp. 4-11.

5) 土居範久，他 ："KEIO-TOSBAC タイムシェアリング・システム：入出力制御とスーパーバイザ，：" 情報処理，vol. 9 No.2, March 1968, pp.80-87.

6) 土居範久 ："KEIO-TOSBAC タイムシェアリング・システム，" 情報処理月例会資料 33, 1968.

7) 浦 昭二，土居範久 ："KEIO-TOSBACタイムシェアリング・システム" EDPアプリケーション ハンドブック，日刊工業新聞社，1968, pp.639-658.

8) "BASIC", Dartmouth College Computation Center, June 1965.

9) "Dartmouth Time-Sharing System, "Dartmouth College Computation Center, Oct. 1964.

10) Helen V. Braden and William A. Wulf : "The Implementation of a BASIC System in a Multiprogramming Environment," CACM, vol. 11, No. 10, Oct. 1968.

11) ファリーナ著 関根智明訳 ："タイムシェアリング プログラミング＝BASICによる。" 培風館，1969.

12) C.L. Baker : "JOSS : RUBRICS," The RAND Corp., AD649321, March, 1967.

昭和44年6月発行

発行所　財団法人　日本情報処理開発センター
東京都港区芝公園21号地1番5
機械振興会館内
TEL (434) 8211 (代表)

印刷所　三協印刷株式会社
東京都渋谷区渋谷3－11－11
TEL (407) 7316

43—S006