普 及 課

49－S002

コンバインド・シミュレータの開発

昭 和 50 年 3 月

JIPDEC

財団法人 日本情報処理開発センター

この報告書は，日本自転車振興会から競輪収益の一部である機械工業振興資金の補助を受けて昭和49年度に実施した「コンバインド・シミュレータの開発」の成果をとりまとめたものであります。

序

当財団は情報処理に関する研究開発の一環として，各種のソフトウェアの開発をおこなっておりますが，本書は「オンライン・コンバインド・シミュレーション言語」の開発成果を報告するものであります。

一般に利用されているシミュレーション言語には，連続系シミュレーション用と離散系シミュレーション用の２つのタイプがあります。しかし，シミュレーションの対象が広がるにつれて，一方のタイプの言語だけでは扱いきれない場合も多くなってまいりました。このため，両者の機能を兼備したコンバインド・シミュレーション・システムの開発を進めております。本年度は前年度おこないましたシステムの基本設計をもとに，具体的なインプリメンテーションを実施いたしました。

本報告書がこの方面に興味を持たれる方々に広く利用され，情報処理技術の発展の一助となることを念願するものであります。

昭和50年3月

財団法人　日本情報処理開発センター
副会長　斉藤　有

ま　え　が　き

現在までに数多くの汎用シミュレーション言語が開発されているが，これらは離散系シミュレーションや連続系シミュレーションのそれぞれの系を専門に扱う目的で作られている。コンピュータ・シミュレーションの普及およびコンピュータの規模拡大や高速化に伴い，シミュレーションの対象とする領域も広がり，内容も複雑・高度化し，離散・連続の両系の要素を含んだ問題を解くことが多くなってきた。この様な対象を扱うシミュレーションに対し，従来はどちらかの言語で扱える様に簡略化したり省略したりしてモデル化していたが，モデルとしての妥当性や結果に対する信頼性が期待できない場合が生じ，満足の行かない場合も多かった。そこで離散系と連続系が同時に扱え，両系の相互関係を表現できる様なシミュレーション言語の開発が望まれるようになってきた。

今日ではバッチ用のコンバインド・シミュレータとして，ＧＬＳ，ＧＥＳＴ，ＧＡＳＰ Ⅳなどいくつかのシステムが開発され，実用化され始めてきた。中でもＧＡＳＰ ⅣはＦＯＲＴＲＡＮベースで，理解がし易く，適用範囲が大変に広く，融通性に富んでいるというような点で注目を集めている。

我々が開発したコンバインド・シミュレータＳＩＭＢＯＬ－Ⅱ（SImulation Model Builder for On-Line usage－Ⅱ）は上記の様な領域を対象とした汎用シミュレーション言語であり，離散系シミュレーション・システムＳＩＭＢＯＬと同様，タイム・シェアリング・システムの下で端末（ＣＲＴキャラクタ・ディスプレイ）からインタラクティブにシミュレーションを行う会話型のシステムである。モデルの作成や変更，試行錯誤的な実行の繰返し，簡単なグラフ表示，モデルのステータスのダイナミックなモニタリング等に会話型利用の効果を出そうと試みた。

このプロジェクトは前年度からの継続であり，前年度は既存のシミュレーション言語の調査およびＳＩＭＢＯＬ－Ⅱの基本設計を中心に行い，本年度はその基本設計をもとに具体的なインプリメンテーションを行った。本報告書は，システム仕様およびシステム設計の概要をまとめたものである。

第１章ではコンバインド・シミュレータについて概説し，第２章ではＳＩＭＢＯＬ－Ⅱシステムを使う場合の仕様について説明し，第３章ではＳＩＭＢＯＬ－Ⅱのシステム設計およびプログラム設計の概要をまとめ，第４章ではコンバインド・シミュレータとして生じる問題について述べている。

目　　次

目　　次

1. コンバインド・シミュレーター一般論

1. コンバインド・シミュレータ一般論

シミュレーションの手法は一般に，システムモデルの状態変化を時間の進め方により，離散的にとらえるか，連続的にとらえるかに大別されている。これらは現実のシステムをそのまま表わすのではなく，モデル化してそれをシミュレートするのである。現実のシステムをシミュレーション実行の対象としてモデル化する場合，離散系だけで表現できる，あるいは連続系だけで表現できるという様な，単純な構成をしているものは少なく，各々の系が複雑にからみ合って1つのシステムを構成していることが多い。従来は離散系部分と連続系部分とに分けて別々にモデル化するか，あるいは一方の系で近似的に表現してモデル化するかの方法しかなく，モデル化が困難な上，シミュレーションの結果も必要な精度が得られないということが起こっていた。そこで，より現実のシステムに則したモデルが記述でき，必要な精度で解析できるようにと，離散系・連続系両方の機能を備えたシミュレーション言語が開発されるようになり，それをコンバインド・シミュレータと呼んでいる。

コンバインド・シミュレーションでは，モデルの状態がある時点までは離散的変化をし，それ以降では連続的変化をするという形，あるいは離散的変化に連続的変化がつけ加わった形の変化をする。この様に，コンバインド・シミュレーションの特色は離散的変化をするものと連続的変化をするものとの相互作用によって起こり，これにはつぎの様な形が考えられる。

(i) 連続変数が離散的変化を受ける場合

(ii) 離散的変化の間に連続的変化が起こる場合

(iii) 離散的変化と連続的変化がお互いに関連しながら変化する場合

すでに開発され，実用化されているコンバインド・シミュレーション言語としてはGSL，GEST，GASP IVなどがあげられる。GASP IVはFORTRANベースのバッチ用シミュレータであり，離散系シミュレーション，連続系シミュレーションおよびコンバインド・シミュレーションに用いることができる。表1－1にGASP IVを用いたシミュレーションモデルの構造を示す。

このように，コンバインド・シミュレータは今後商・工業の分野のみならず，社会，エネルギー，環境，生態等のシステムに適用されると予想されている。

また，シミュレーション言語をオン・ライン会話型システムとする事が考えられ，最近ではかなり実用化されている。モデルの作成や変更，試行錯語的な実行の繰り返し等を会話的に，またはインタラクティブに行える事は効果的であると言える。連続系シミュレータや離散系シミュレータの中にはオンライン・シミュレーション・システムとして，すでに実用化されているものがいくつかあるが，コンバインド・シミュレータにはオンライン・シミュレーション・システムとして実用化

されているものはなく，実験段階にとどまっている。計算機のハードウェア，ソフトウェア両面の進歩によって，マン・マシン・システムの形態をとったシミュレータは，問題解決のための道具としてのシミュレーションの意義をより大きいものにすると考えられる。

表1－1　GASPIVシミュレーションモデルの構成例

適用分野	シミュレーションの型	連続変数	方程式の形	離散変数	イベント
1. 電気メッキ	コンバインド	• めっきタンクと流出物の中の残留金属濃度水準	差分 微分	• すくい出し量	• めっき工程において部品を薬液に浸したりとり出したりする事 • 沈積物の除去
2. 気候	連続	• 世界の温度 • 汚染度合	差分		
3. 病院	離散			• 待ち時間 • 資源の利用度	• 患者や医者の到着と離散 • 設備の占有と解放
4. 残留金属	コンバインド	• 区画内のカドミウム量	差分	• 降雨回数と降雨量	• 降雨
5. ケーブルの製造	コンバインド	• プロセスの動作状況	微分	• 待ち時間 • 操作員の利用度	• 生産工程へのケーブルの到着
6. 森林の伐採	コンバインド	• 木の成長度	差分		• 伐採と植樹
7. 収穫方法	離散			• 生産水準 • 操作員／装置の利用度	• 機械操作と手作業
8. 交通	コンバインド	• 加速度，速度および自動車の位置	微分	• 交差点での渋滞	• 交通信号の制御
9. スケジューリング	コンバインド	• プロセス変数	微分	• 生産水準 • 機械／操作員の利用度	• 手作業 • 原料の輸送
10. 保険	離散			• 年次所得 • 保険契約者数	• 方針の開始と終了
11. 産児制限	コンバインド	• ホルモン水準	微分 差分 代数		• 産児制限用ピルの投薬
12. ワールドモデル	コンバインド	• 人口，資源，汚染および資本の水準	差分		• 天候の大異変 • 伝染病 • 飢きん • 政策 • 技術的な発見
13. 牧草	コンバインド	• 収穫高 • 温度	差分	• 機械の利用度	• 設備の輸送
14. 昆虫の成長	コンバインド	• 温度 • 昆虫の総数	差分 微分		• 昆虫の卵のふ化

2. ＳＩＭＢＯＬ－Ⅱシステムの説明

2. SIMBOL－Ⅱシステムの説明

2.1 SIMBOL－Ⅱの特徴

SIMBOL-Ⅱ システムはモデルの作成段階と実行段階との間の障壁を取り除き，効率よくシミュレーションを行える事を目的として開発された。インタラクティブな会話型シミュレーションシステムである。

このシステムの一番大きな特徴は，タイムシェアリングシステムのもとでオンライン・インタラクティブなコンバインド・シミュレーションを行うための言語であるという事ができる。つぎに，その特徴を箇条書きにする。

① 会話型言語である。

② 離散系および連続系が複雑にからみ合ったシステムをモデル化し，それをシミュレートする，いわゆるコンバインド・シミュレーションができる。

③ プロセスタイプの言語であり，モデルの表現のし易さと記述性の豊かさを備えている。

④ モデルの作成がインタラクティブにでき，プログラムの修正が簡単である。

入力されたステートメントはその場でコンパイルされ，エラーがあると即座にその旨表示されるので，ユーザはただちにプログラムの修正をすることができる。

⑤ モデルの実行方法に融通性がある。

パラメータを変えて実行をくり返す，初期状態や積分間隔を変えて実行をくり返す，あるいはアルゴリズムを変えて実行をくり返すという様な，いろいろな実行方法を簡単に使える機能を備えている。

⑥ モデル実行中にその様子を適確につかむ事ができる。

モデルの実行中にQuit をかけて，実行を一時中断して変数の値やシステムクロックを参照したりできる。必要ならば，変数の値を変えてから実行を再開したり，場合によってはシミュレーションをそこで打ち切ったりすることなど，適確な処置を取ることができる。

⑦ トレース機能が豊かなだけでなく，その扱いが自由である。

バッチ用シミュレーション言語でトレースをとるためには，モデルの実行以前にトレースの指示をすると，途中でもはや必要ない事がわかっても，実行を終了するまで情報をとり続けてしまうという様な無駄が生じることがある。この点，SIMBOL-Ⅱ ではトレース情報を見ながら，必要ないことがわかればコマンドによって打ち切ることが可能である。また，実行途中でトレースをとる指示もできる。

⑧ キャラクタ・ディスプレイを端末としている。

プログラムの入力や結果の表示はキャラクタ・ディスプレイ端末から行っている。シミュレーションの実行結果や統計データの分布をキャラクタ・ディスプレイを使って簡単なグラフ表示ができるので，その様子を一目でとらえる事ができ，即座に，次にとるべき処置を決定できる。また，キャラクタ・ディスプレイ装置に付属しているハードコピー装置を用いて，ユーザは任意の時点で一画面分のハードコピーをとることもできる。

2.2 操作仕様

2.2.1 ステージ遷移

SIMBOL-Ⅱでは，モデルの作成，修正，実行などシミュレーションの過程をインタラクティブに行えるように，いくつかのステージとモードに分けて考えている。

ステージは，モデリング・ステージとシミュレーション・ステージに分けられる。さらに，シミュレーション・ステージはイニシャル・モード，ポーズ・モード，エンド・モードの3つのモードに分けられていて，シミュレーション実行中のモデルとインタラクションを取り易くするようにしている。

図2－1はステージおよびモードの遷移の様子を示したものである。

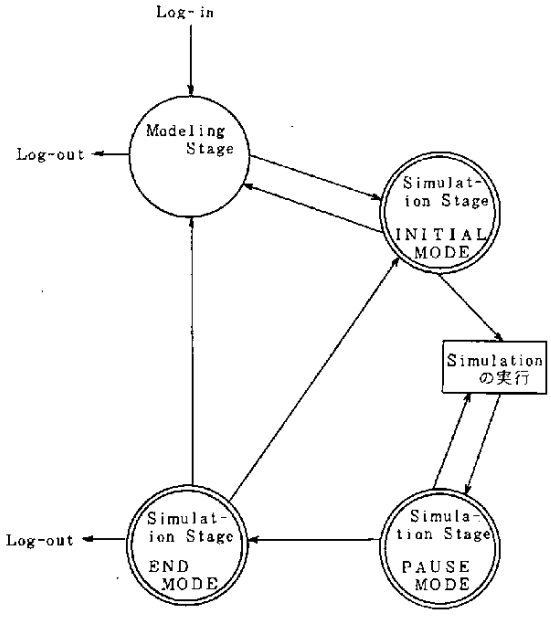


図2－1　ステージおよびモード遷移図

(1) モデリング・ステージ

ユーザがＣＲＴキャラクタ・ディスプレイ装置のキーボードからタイムシェアリング・システムに対しSIMBOL-Ⅱの起動を依頼して，Log-in が完了した状態を言う。このステージでは，すでに登録してあるモデルをロードしてその続きを作成したり，それを修正したり，あるいは新たにモデルの作成を始めたりする，モデルビルディングをインタラクティブに行うことができる。

(2) シミュレーション・ステージ

モデリング・ステージでできあがったモデルの実行を行うステージで，前述した３つのモードに分けられている。各モードごとに説明をする。

① イニシャル・モード

完成したモデルを実行するための各種の初期設定を行い，実際に実行の開始を指示するモードである。

② ポーズ・モード

シミュレーション実行の途中で，モデルの様子を見たり，パラメータの値を変えたり，トレースをかけたりはずしたりなどを行う実行中のモデルとインタラクションを取るためのモードである。

③ エンド・モード

モデルの実行途中の状態や実行結果をリポートするモードである。

2.2.2 コマンド

コマンドはSIMBOL-Ⅱシステムに，モデルの実行や中止を指示したり，プログラムの編集を指示したり，ステージやモードの遷移を指示したりなどするための手段である。

本システムでは，ＣＲＴキャラクタディスプレイ装置の画面上に，現在のステージやモードで使用可能なコマンドを表示し，ユーザがライトペンを用いて任意のコマンドを選択するようになっている。各コマンド特有のパラメータについては，その都度入力するよう指示があるので，それに従ってキーボードから入力すればよい。

シミュレーションステージとコマンドの関係を図に表わすと，次のようになる。

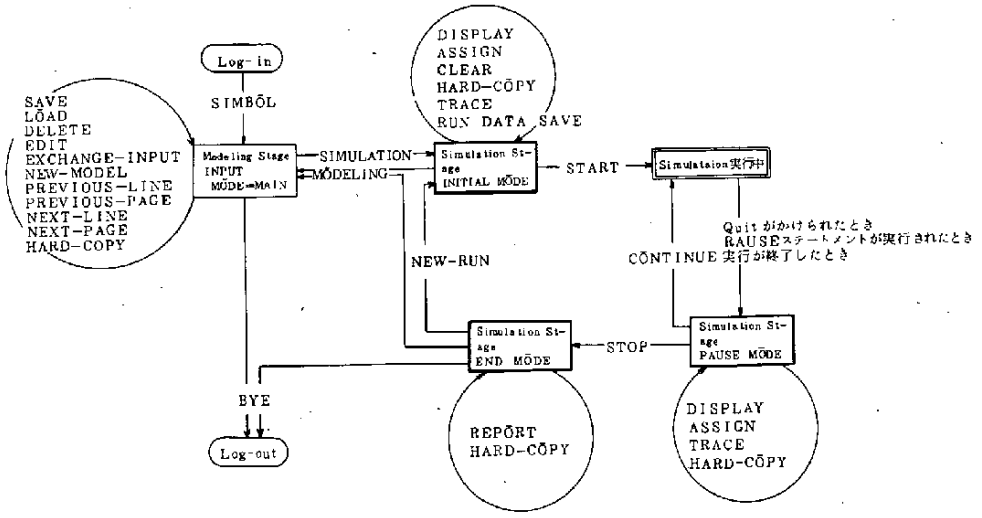


図２－２　シミュレーションステージとコマンドの関係

SIMBOL-Ⅱのコマンドは24あり，中にはサブコマンドを持つものもある。

1. ASSIGN
2. BYE
3. CLEAR
   - ALL-CLEAR
   - SELECTED-CLEAR
4. CONTINUE
5. DELETE
6. DISPLAY
7. EDIT
   - DELETE
   - EDIT-END
   - INSERT
   - LIST
   - RENUMBER
8. EXCHANGE-INPUT
9. HARD-COPY
10. LOAD
11. MODELING
12. NEW-MODEL

13. NEW-RUN
14. NEXT-LINE
15. NEXT-PAGE
16. PREVIOUS-LINE
17. PREVIOUS-PAGE
18. REPORT
    - FREQUENCY-TABLE
    - HISTOGRAM
    - PRINT
19. RUN DATE-SAVE
20. SAVE
21. SIMULATION
22. START
23. STOP
24. TRACE
    - START
    - STOP
    - ON
    - OFF
    - VIEW

以下に，各コマンドについて機能と使い方を簡単に説明し，表２－１にコマンド一覧表を示す。

(1) ASSIGN コマンド

機　能

グローバル変数に値を代入する。

使い方

シミュレーション・ステージ，イニシャル・モードまたはポーズ・モードにおいて"ASSIGN"をピックする。変数名とその値（定数）の入力を要求してくるので，ルールに従ってキーボードより入力する。

(2) BYEコマンド

機　能

ジョブの終了を指示する。

使い方

モデリング・ステージおよびシミュレーション・ステージ，エンド・モードのときに

"BYE"をピックする。

(3) CLEAR コマンド

機　能

グローバル変数，統計用テーブルをゼロに設定する。

使い方

シミュレーション・ステージ，イニシャル・モードのときに"CLEAR"をピックする。CLEAR サブパネルに移行し，サブコマンドが表示されるのでその中から任意のものをピックする。

① ALL-CLEAR サブコマンド

機　能

グローバル変数，統計用テーブルをすべてゼロに設定する。

使い方

CLEAR サブパネルのときに" ALL-CLEAR"をピックする。

② SELECTED-CLEAR サブコマンド

機　能

指定のグローバル変数，あるいは統計用テーブルをゼロに設定する。

使い方

CLEAR サブパネルのとき"S-CLEAR"をピックする。名前の入力を要求してくるので，グローバル変数名あるいはテーブル名を入力する。

(4) CONTINUE コマンド

機　能

シミュレーションの実行を継続することを指示する。

使い方

シミュレーション・ステージ，ポーズ・モードのときに"CONTINUE"をピックする。

(5) DELETE コマンド

機　能

すでにセーブしてあるプログラムの中のあるものを消去する。

使い方

モデリング・ステージのときに"DELETE"をピックする。プログラム名の入力を要求してくるので，キーボードより入力する。

(6) DISPLAY コマンド

機　能

グローバル変数の値を表示する。

使い方

シミュレーション・ステージ，イニシャル・モードおよびポーズ・モードのときに"DISPLAY"をピックする。変数名の入力を要求してくるので，キーボードより入力する。

(7) EDITコマンド

機　能

プログラムの編集を行うことを指示する。

使い方

モデリング・ステージのときに"EDIT"をピックする。EDITサブパネルに移行し，サブコマンドが表示されるので，その中から任意のものをピックする。

① DELETEサブコマンド

機　能

ステートメントの削除を指示する。

使い方

EDITサブパネルのとき"DELETE"をピックする。行番号の入力を要求してくるので，ルールに従って入力する。

② EDIT-ENDサブコマンド

機　能

EDITコマンドの終了（EDITサブパネルからの脱出）を指示する。

使い方

EDITサブパネルのとき"EDIT-END"をピックする。

③ INSERTサブコマンド

機　能

ステートメントの挿入を指示する。

使い方

EDITサブパネルのとき"INSERT"をピックする。ステートメントの入力を要求してくるので，行番号とステートメントを入力する。

④ LISTサブコマンド

機　能

ソース・プログラムのリストを表示するよう指示をする。

使い方

EDITサブパネルのとき"LIST"をピックする。プログラム名と行番号の入力を要求してくるので，プログラム名・行番号の順に入力する。

⑤ RENUMBERサブコマンド

機　能

行番号のつけ直しを指示する。

使い方

EDITサブパネルのとき"RENUMBER"をピックする。行番号の入力を要求してくるので，ルールに従って入力する。

(8) EXCHANGE-INPUTコマンド

機　能

モデリング・ステージにおけるインプットモードを変更する。

使い方

モデリング・ステージのとき" EXC-INPUT"をピックする。インプットモードの入力を要求してくるので，キーボードより入力する。

(9) HARD-COPYコマンド

機　能

ディスプレイ画面のスクリーンエリアのハードコピーをとる。

使い方

" HARDCOPY "をピックする。これは全ステージで使うことができる。

(10) LOADコマンド

機　能

セーブしてあるプログラムをロードする。

使い方

モデリング・ステージのときに" LOAD "をピックする。プログラム名の入力を要求してくるので入力する。

(11) MODELINGコマンド

機　能

モデリング・ステージに遷移する。

使い方

シミュレーション・ステージ，イニシャル・モードおよびエンド・モードのときに"MODELING"をピックする。

(12) NEW-MODELコマンド

機　能

新しくモデルを作成するための準備をする。

使い方

モデリング・ステージのときに"NEWMODEL"をピックする。

(13) NEW-RUNコマンド

機　能

シミュレーション・ステージ，イニシャル・モードに遷移する。

使い方

シミュレーション・ステージ，エンド・モードのときに"NEW-RUN"をピックする。

(14) NEXT-LINE コマンド

機　能

ディスプレイ画面のスクリーンエリアを1ステートメント前進させる。

使い方

モデリング・ステージのときに"NXT-LINE"をピックする。

(15) NEXT-PAGE コマンド

機　能

ディスプレイ画面のスクリーンエリアを1ページ前進させる。

使い方

モデリング・ステージのときに"NXT-PAGE"をピックする。

(16) PREVIOUS-LINEコマンド

機　能

ディスプレイ画面のスクリーンエリアを1ステートメント後退させる。

使い方

モデリング・ステージのときに"PRE-LINE"をピックする。

(17) PREVIOUS-PAGEコマンド

機　能

ディスプレイ画面のスクリーンエリアを1ページ後退させる。

使い方

モデリング・ステージのときに"PRE-PAGE"をピックする。

(18) REPORT コマンド

機　能

シミュレーション実行中の情報を表やグラフとして出力する。なお，データを時系列に見たいときはRUNDATA-SAVEコマンドでデータをセーブしておかなければならない。

使い方

シミュレーション・ステージ，エンド・モードのときに"REPORT "をピックする。REPORTサブパネルに移行し，サブコマンドが表示されるので，その中から任意のものをピックする。

① FREQUENCY TABLEサブコマンド

機　能

度数分布表を表示する。

使い方

REPORTサブパネルのとき" FREQ-TAB"をピックする。テーブル名を要求してくるので，入力する。

② HISTOGRAM サブコマンド

機　能

ヒストグラムを表示する。

使い方

REPORTサブパネルのとき"HISTOG" をピックする。テーブル名を要求してくるので，入力する。

③ PRINT サブコマンド

機　能

RUN DATA-SAVEコマンドによって収集した△t ごとの情報を出力する。

使い方

PEPORTサブパネルのとき"PRINT"をピックする。変数名を要求してくるので，入力する。

⑲ RUN DATA-SAVE コマンド

機　能

シミュレーション実行後に，グローバル変数の△tごとの値を出力させるために，実行中のデータをセーブしておく。

使い方

シミュレーション・ステージ，イニシャル・モードのときに" RDT-SAVE"をピックする。グローバル変数名とセーブの打切り時刻の入力を要求してくるので，入力する。

⑳ SAVEコマンド

機　能

コア上のソース・プログラムを，プログラム単位あるいは全部を，ソース・プログラム・ファイルにセーブする。

使い方

使い方

モデリング・ステージのときに"SAVE"をピックする。プログラム名の入力を要求してくるので，入力する。

(21) SIMULATIONコマンド

機　能

シミュレーション・ステージ，イニシャル・モードへ遷移する。

使い方

モデリング・ステージのときに"SIMULATION"をピックする。

(22) START コマンド

機　能

シミュレーションの実行を開始する。

使い方

シミュレーション・ステージ，イニシャル・モードのときに"START"をピックする。

(23) STOP コマンド

機　能

シミュレーション・ステージ，エンド・モードへ遷移する。

使い方

シミュレーション・ステージ，ポーズ・モードのときに"STOP"をピックする。

(24) TRACEコマンド

機　能

トレースの指定をする。

使い方

シミュレーション・ステージ，イニシャル・モードおよびポーズ・モードのときに"TRACE"をピックする。ＴＲＡＣＥサブパネルに移行し，サブコマンドが表示されるので，その中から任意のものをピックする。

① START サブコマンド

機　能

トレースの実行を開始する。

使い方

TRACE サブパネルのときに"START"をピックする。

② STOPサブコマンド

機　能

トレース処理を終了させる。

使い方

TRACE サブパネルのときに"STOP"をピックする。

③ ＯＮサブコマンド

機　能

トレース・スイッチをセットする。

使い方

TRACE サブパネルのときに"ＯＮ"をピックする。トレース対象の指定を要求してくるので，入力する。

④ ＯＦＦサブコマンド

機　能

トレース・スイッチをリセットする。

使い方

TRACE サブパネルのときに"ＯＦＦ"をピックする。トレース対象の指定を要求してくるので，入力する。

表2－1　コマンド一覧表

コ　マ　ン　ド	使用可能ステージ	機　　能
ASSIGN	I, P	グローバル変数に値を代入
BYE	M, E	ジョブの終了
CLEAR	I	グローバル変数，統計用テーブルのゼロクリア
CONTINUE	P	シミュレーションの実行を継続
DELETE	M	プログラムの消去
DISPLAY	I, P	グローバル変数の値を表示
EDIT	M	プログラムの編集
EXCHANGE-INPUT	M	インプット・モードの変更
HARD-COPY	M,I,P,E	ディスプレイ画面のスクリーンエリアのハードコピー
LOAD	M	プログラムのロード
MODELING	I, E	モデリング・ステージへの遷移
NEW-MODEL	M	新しくモデルを作成するための準備
NEW-RUN	E	シミュレーション・ステージ，イニシャル・モードへの遷移
NEXT-LINE	M	ディスプレイ画面を1ステートメント前進
NEXT-PAGE	M	ディスプレイ画面を1ページ前進
PREVIOUS-LINE	M	ディスプレイ画面を1ステートメント後退
PREVIOUS-PAGE	M	ディスプレイ画面を1ページ後退
REPORT	E	収集したデータの出力
RUN DATA-SAVE	I	実行中のデータの収集
SAVE	M	プログラムのセーブ
SIMULATION	M	シミュレーション・ステージ，イニシャル・モードへの遷移
START	I	シミュレーションの実行開始
STOP	P	シミュレーション・ステージ，エンド・モードへの遷移
TRACE	I, P	トレースの指定

※　使用可能ステージについて

M：モデリング・ステージ

I：シミュレーション・ステージ，イニシャル・モード

P：シミュレーション・ステージ，ポーズ・モード

E：シミュレーション・ステージ，エンド・モード

2.2.3 プログラムの入力手順

SIMBOL-Ⅱによるモデルは，メイン・プログラム以外にアクティビティ・プログラム，プロセデュア，関数といったプログラム要素によって構成されている。メイン・プログラム以外のものは，必要なければ省略してかまわない。

SIMBOL-Ⅱはこういったいくつかのプログラム要素をコンカレントに入力できる。即ち，1つのプロセデュアを組み終った後でなくては次のプロセデュアを入力できないと言った事はなく，始めはアクティビティAのステートメントを，次にはアクティビティBのステートメントを，と全く任意の順序でステートメントを入力していける。また，SIMBOL-Ⅱではプロセデュア間とか，メイン・プログラムとプロセデュアなど，個々のプログラム要素の間ではお互いに同じラインナンバーを持つステートメントがあっても構わない。そこで，現在入力しているステートメントがどの要素に属すのかをはっきり区別するため，インプットモードという考え方を導入している。インプットモードは各プログラム要素に付けられた名前をそのまま用いるが，メイン・プログラムの場合にはMAINという名前が決められている。

例えば，アクティビティCARを入力している時には，インプットモードがCAR，プロセデュアSUBを入力している時には，インプットモードがSUBであるという。

インプットモードはEXCHANGE-INPUTコマンドによって，モデル作成中にユーザが任意に変更できる。但し，プロセデュアを使用する時は事前にプロセデュア・プログラムを作成しておかなくてはならない。

各プログラム要素はいくつかのステートメントで構成される。SIMBOL-ⅡではCRTキャラクタディスプレイ装置から入力する単位をラインと呼んでおり，ステートメントがそれに相当する。ステートメントには，必ずラインナンバーと呼ばれる数値を対応づけ，ステートメントを検索する場合のキーとしている。

ステートメントを入力するCRTキャラクタディスプレイ装置の画面は，図2-3のように大きく2分されていて，上部には入力し終ったステートメントが表示され，下部にキーインしているステートメントが表示される。

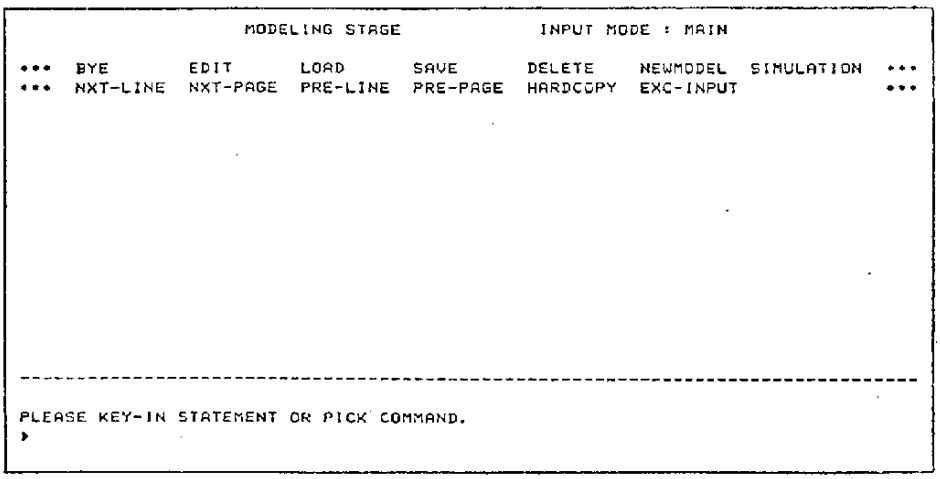


図2－3　ＣＲＴキャラクタディスプレイ装置の画面

ステートメントを入力する時には，まず，ラインナンバーを打鍵する。その後に，ステートメントラベルが必要ならこれを打鍵し，以下ステートメント本体を続けて，セミコロンで締めくくり，SENDキーを押す。

ステートメントの入力順は，ラインナンバーが増加するように行う必要は特になく，任意の順番で入力してさしつかえない。入力したステートメントにシンタックス・エラーがなければ，ディスプレイ画面の上部にラインナンバーが揃う位置に表示される。入力する順番が任意であっても，システムの内部では，入力されたステートメントのラインナンバーの順に並びかえられ，セーブされているのである。

入力したステートメントにエラーが見つかると，ディスプレイ画面の下線のすぐ下にエラーメッセージが表示される。エラーのあったステートメントはそのまま画面の下部に残っているので，エラー部分を修正し，再度 SENDキーを押して入力すればよい。

2.2.4　モデルの実行

モデルの実行開始はシミュレーション・ステージ，イニシャル・モードにおいて START コマンドをピックすることによって指示される。START コマンドをピックする以前に，モデルのグローバル変数のクリアーやシステム変数，スケジュールテーブルなどの初期設定などを行っておかなくてはならない。

シミュレーションの実行打切りは，実行中にクイットをかけポーズ・モードに移行させて，そこで STOP コマンドをピックすることでできる。しかし，あらかじめ START コマンドのパラメータ

に指定しておくこともできる。パラメータには，時刻とイベント数のどちらかの打切り条件を書くことができる。

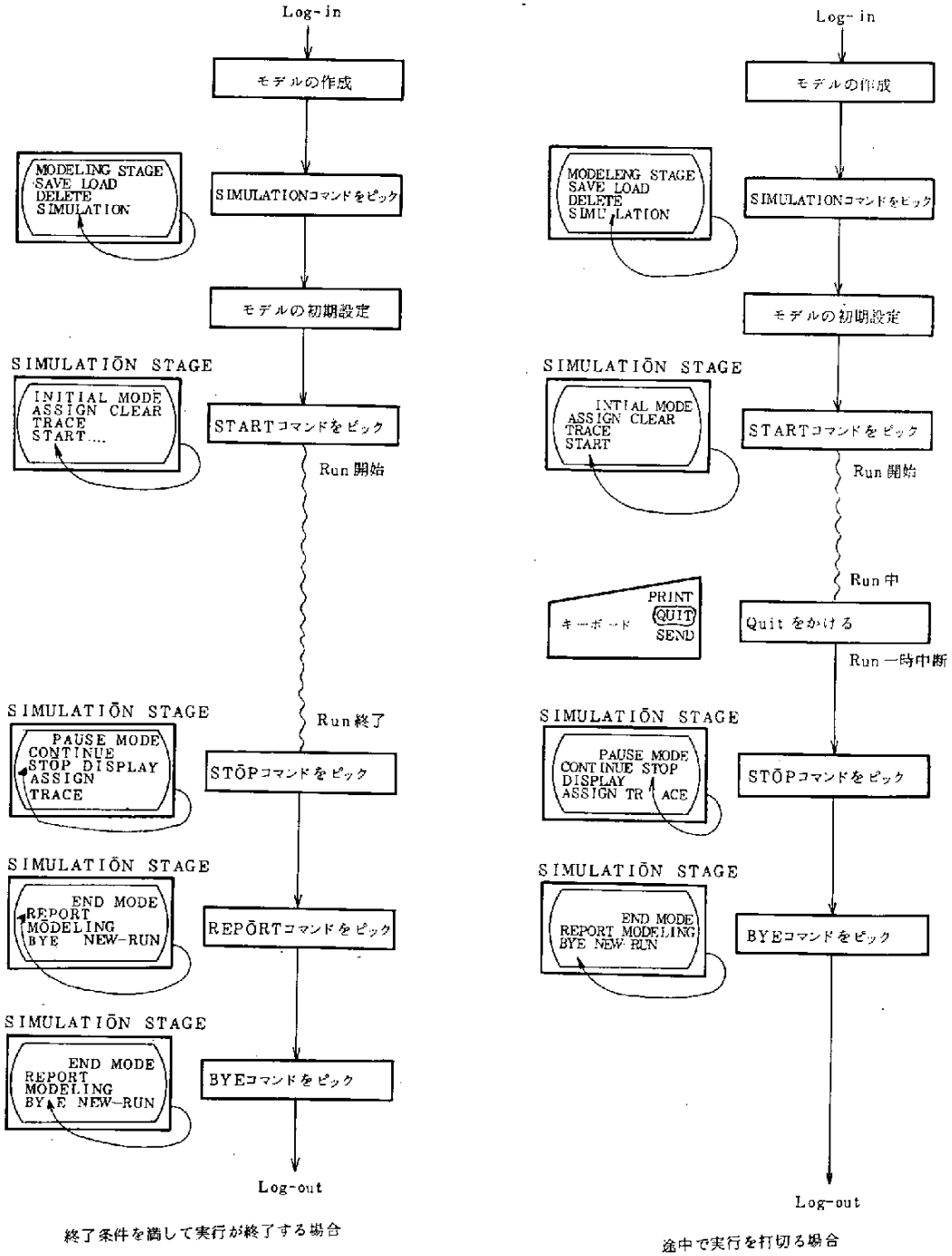


図２－４　モデルの実行手順

2.2.5 モニタリング

SIMBOL-Ⅱでは，実行中のモデルの様子を見る事をモニタリングと呼んでいるが，これには大きくわけると2通りの方法がある。第1は実行中のモデルにクイットをかけ，モデルの実行を一時中断した状態で，モデルの様子を調べる方法である。第2はトレースに依る方法で，この場合はモデルの実行を中断する必要はない。

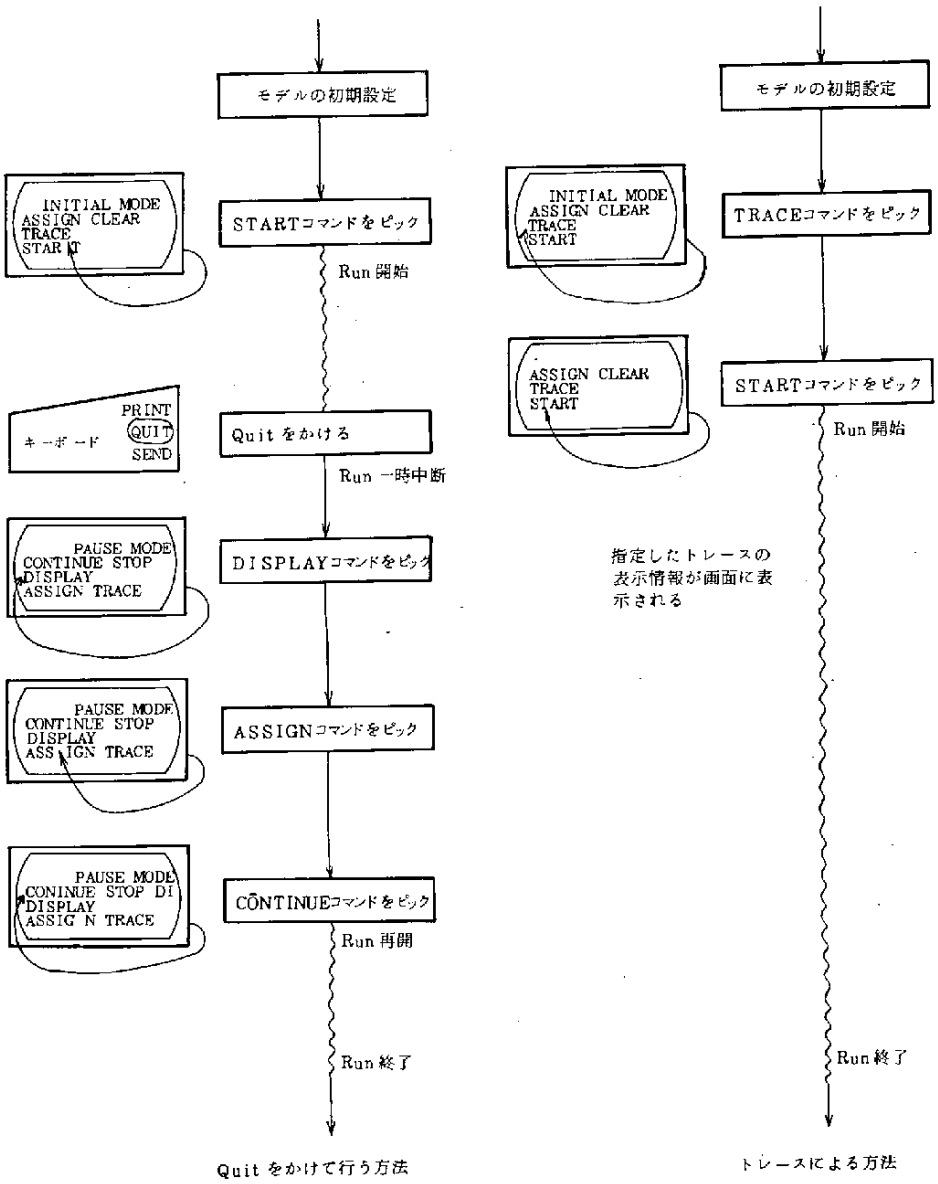


図2－5　モニタリングの方法

(1) クイットをかけて行う方法

この方法では，シミュレーションクロックを実際に進めている最中に，キーボード上のQUITキーを押下してクイットをかける。するとシミュレーション・ステージ，ポーズ・モードにモードが遷移し，ここで使用可能なコマンドが表示される。ユーザは，DISPLAYコマンドを用いてモデルの状態を調べたり，ASSIGN コマンドを用いて変数の値を変更したりすることができる。そして，このモードからそのまま実行を続ける場合にはCONTINUE コマンドをピックすれば良い。このようにユーザはシミュレーションの実行中，任意の時点でクイットをかけることによって，実行中のモデルとインタラクションをとることができる。

(2) トレースに依る方法

この方法では，シミュレーション・ステージ，イニシャル・モードあるいはポーズ・モードにおいてTRACE コマンドを用いて，実行中の情報を得るために，トレースの種類や実行を指定する。SIMBOL-IIでは，トレースをかけるタイミングの問題（開始・終了）と，何をトレースするかという情報指示の問題（設定・解除）とを明確に区分して，それぞれサブコマンドによって指定するようになっている。

タイミングに関するサブコマンドとして，トレースの開始・終了指示がある。

TRACE START

TRACE STOP

START サブコマンドはトレースの実行開始を指示するものであり，このサブコマンドが指定されなければトレースの実行は行われない。STOP サブコマンドはトレース処理を終了させるためのものであり，START サブコマンドと組み合わせて使用する。

トレース対象の指示に関するサブコマンドとしては，トレース・スイッチのセット・リセット指示がある。

TRACE ON

TRACE OFF

ONサブコマンドはトレース・スイッチのセットを指示するものであり，OFFサブコマンドはセットされているスイッチのリセットを指示するものである。トレース・スイッチがセットされていても，前述の START サブコマンドが入れられなければ，トレース処理は行われないので注意を要する。ON・OFFサブコマンドによりトレース・スイッチの操作が行われるが，その時指示できるトレース対象には表2－2に示すような4種類が用意されている。

表２－２　トレース対象

①プロセストレース（ＰＲＯＣ）	プロセスの発生・消滅	（ＰＣＲＤ）
	スケジュール関係	（ＰＳＣＨＤ）
	グループ操作	（ＧＲＯＵＰ）
②イベントトレース（ＥＶＥＮＴ）		
③ステータストレース（ＳＴＡＴＥ）		
④シーケンストレース（ＰＳＥＱ）		

①　プロセストレース

プロセストレースには，プロセスの発生，消滅に関するもの，プロセスのスケジュール・リスケジュール・キャンセル・ターミネイト等のスケジューリングに関するもの，更にプロセスのグループ内での操作に関するもの，といった３種類がある。これらに続けて「＊」あるいは「Activity list」を指定することにより，全ての関連するプロセスについてのトレースを指示する（＊を指定する場合）のか，特定のアクティビティに属するプロセスについてのトレースを指示する（Activity list を指定する場合）のかという，トレース対象を絞ることができる。

例えば　TRACE ON PROC PSCHD ＊　と指定する。

プロセストレースによる情報は，図２－６に示すように表示される。

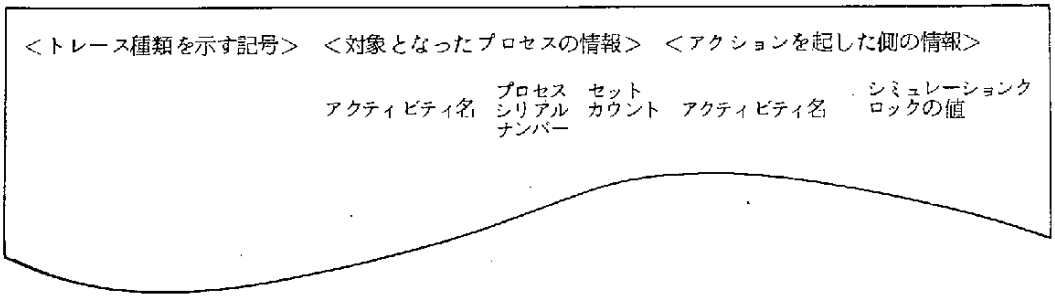


図２－６　プロセストレース表示情報－１

トレース種類を示す記号は，プロセスの発生・消滅・スケジューリング・グループ操作といった種類を識別するためのもので，表２－３に示すような８種類の４桁の省略記号がある。

表２－３　プロセストレース表示情報－２

種　　類	表示記号（４ケタ）
○　プロセスの発生	GNRT
○　プロセスの消滅	DSTY
○　プロセスがスケジュールされた	SCHD
○　プロセスがリスケジュールされた	RSCH
○　プロセスがキャンセルされた	CACL
○　プロセスがターミネイトした	TRMT
○　プロセスがセット又はキューに入れられた	INST
○　プロセスがセット又はキューから取り出された	REMV

②　イベントトレース

イベントトレースは，プロセス内のイベントの前後関係が正しく動作しているかをチェックするためのものである。

ここでもプロセストレースと同じように，「＊」か「Activity list 」かを指定することによって，対象を絞ることができる。

例えば　　TRACE ON EVENT　＊ と指定する

イベントトレースによる情報は，図２－７に示すような項目について表示される。

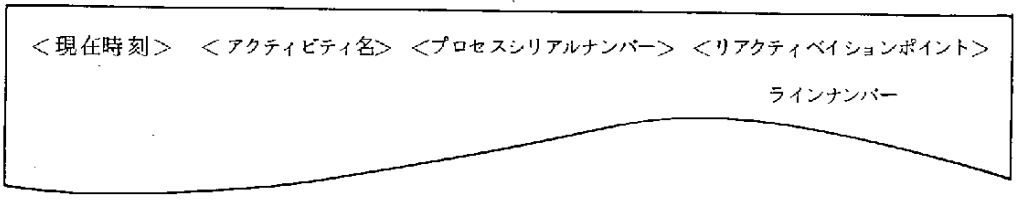


図2－7　イベントトレース表示情報

③　ステータストレース

ステータストレースは，モデルの状態変化をグローバル変数に関連させてチェックするためのものである。ここでは，

TRACE ON STATE　グローバル変数リスト

と，グローバル変数の並びを指定する。

ステータストレースは，特に連続型プロセスの状態変化をチェックするのに用いる。このトレースによる情報は，図２－８のように示される。

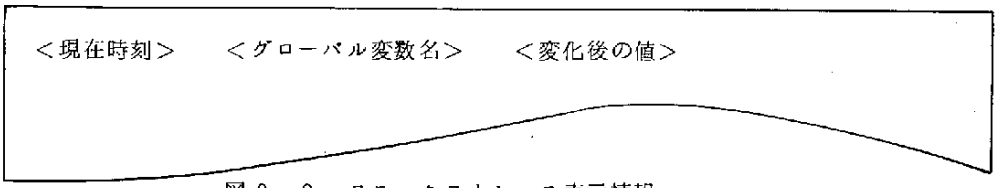


図２－８　ステータストレース表示情報

④　シーケンストレース

シーケンストレースは，プログラムの流れを追跡するもので，BRANCH，DO といった制御プログラム，あるいはCALLステートメントによるプログラムのコーリングチェック，スケジューリング関係のチェック，ラインナンバーによるもの，プログラムの一部分のチェック等ができる。このトレースには，トレースすべき対象を限定させることが可能で，全プログラムを対象とする指定（＊を指定する），プログラム単位を対象とする指定（プログラム名を指定する），プログラム単位内の特定のステートメントを指定する（ラインナンバーを指定する）という方法がある。

シーケンストレースの表示情報は，図２－９のように示される。

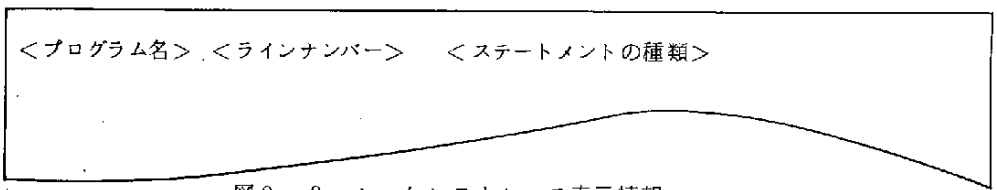


図２－９　シーケンストレース表示情報

以上，トレース対象の指定について述べたが，トレース・スイッチのセット，リセットは，シミュレーションランを通じて何度でも行うことができる。従って，トレース対象を変えたり，範囲を絞ったりあるいは広げたり，ということが容易にできる。このような操作をする場合，以前スイッチをどうセットしたかを知る事ができると都合が良い。SIMBOL－Ⅱでは，トレース・スイッチの状態を表示させるために，VIEWサブコマンドがある。

TRACE VIEW

VIEWサブコマンドを指示すれば，その時のスイッチの状態が把握できる。

2.3 言語仕様

2.3.1 モデル記述の基本概念

SIMBOL-Ⅱではコンバインドタイプのシステムを，離散系プロセスと連続系プロセスという2つのプロセスという概念を用いてモデルを表現する。

離散系プロセスは，対象とするシステムの状態を，時間変化に伴って不連続的に変化すると捉えた方が適切な場合に用いる表現で，順序動作や論理判断などでその変化を記述する。一方，連続系プロセスは，その状態を時間変化に伴って連続的に変化すると捉えた方が適切な場合に用いる表現で，微分方程式を主体に記述する。

SIMBOL-Ⅱでは，これらのプロセスをアクティビティ・プログラムで定義する。離散系プロセスはディスクリート・アクティビティ・プログラムで，連続系プロセスはコンティニュアス・アクティビティ・プログラムでそれぞれ定義し，2つをはっきり区別している。アクティビティ・プログラムとプロセスは必ずしも一対一に対応するのではなく，1つのアクティビティ・プログラムから1つ以上のプロセスが発生する。

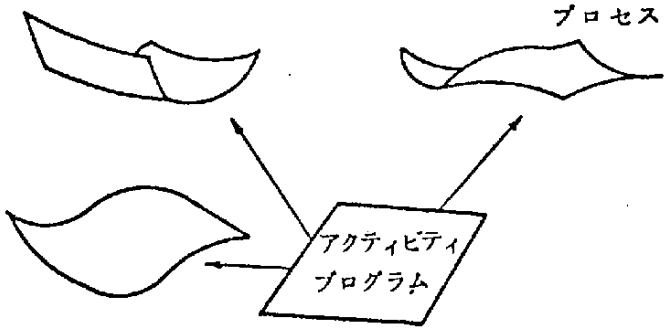


図2-10　アクティビティプログラムとプロセスの関係

1つのアクティビティ・プログラムによって定義されたプロセスの類のことを単にアクティビティと呼んでおり，離散系アクティビティおよび連続系アクティビティがある。プロセスの具体的な扱い方は離散系プロセスと連続系プロセスの区別はなく，同一に取り扱う。これに関しては後述する。

2.3.2 モデル・ストラクチャー

SIMBOL-Ⅱのモデルは，メイン・プログラム，ディスクリート・アクティビティ・プログラム，

コンティニュアス・アクティビティ・プログラム，プロセデュア，関数などを基に構成される。

メイン・プログラムはグローバルなデータを定義したり，シミュレーション実行時のイニシャライズをしたり，などするために設けられ，1つのモデルには必ず唯一つ組込まれていなければならない。メイン・プログラムであることを宣言するステートメントはなく，終りを示すENDステートメントが用いられる。これは他のプログラムとの関係で解るようになっている。

ディスクリート・アクティビティ・プログラムは離散系プロセスを定義するプログラムで，DACTステートメントで始まり，ＥＮＤステートメントで終る

```
ＤＡＣＴ    アクティビティ名
宣 言 文
   ⋮
   ⋮
Ｅ Ｎ Ｄ
```

ディスクリート・アクティビティ・プログラム

コンティニュアス・アクティビティ・プログラムは連続系プロセスを定義するプログラムで，CACTステートメントで始まり，ＥＮＤステートメントで終る。

```
ＣＡＣＴ    アクティビティ名
宣 言 文
   ⋮
   ⋮
Ｅ Ｎ Ｄ
```

コンティニュアス・アクティビティ・プログラム

SIMBOL-IIではアクティビティ・プログラムをサブルーチンとか関数といった単位で分割してプログラム出来るようになっている。このサブルーチンに相当するプログラムをプロセデュアと呼び，PROCステートメントで始まり，ＥＮＤステートメントで終る。

```
ＰＲＯＣ    プロセデュア名(パラメータ)
   ⋮
   ⋮
Ｅ Ｎ Ｄ
```

プロセデュア

関数はタイプ付きのプロセデュアで表わしタイプ付きPROCステートメントで始まり，ＥＮＤステートメントで終る。ここでタイプとはREAL，INTEGER，LOGIC，PROCESSの事をさす。

```
タイプPROC プロセデュア名(パラメータ)
  ⋮
  ⋮
END
```

関　数

これらのプログラムの構造を示すと次のようになる。

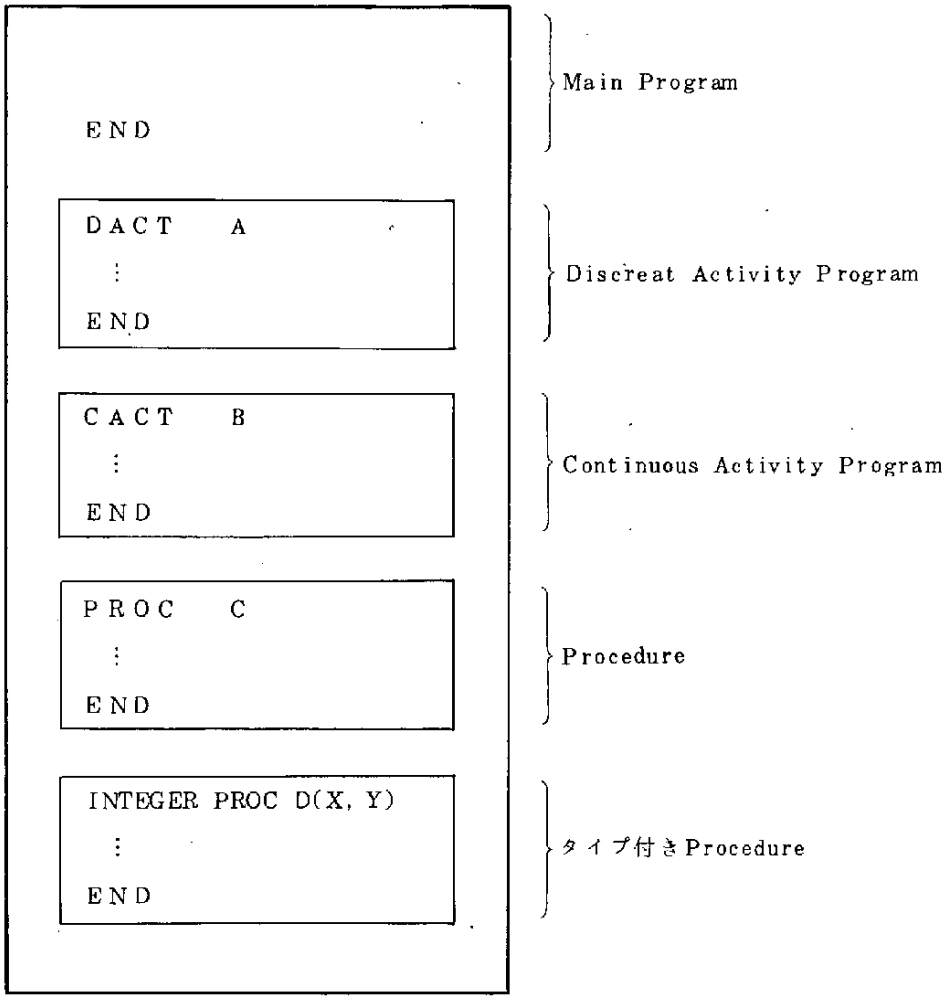


図２－１１　プログラムの構造

2.3.3 SIMBOL－Ⅱで扱うデータ

SIMBOL－Ⅱで扱うことのできるデータとしては，次の5つのタイプがある。

1. 実　数　型
2. 整　数　型
3. 論　理　型
4. プロセス型
5. グループ

実数型データ，整数型データ，論理型データ，プロセス型データは各々変数と定数がある。変数はその名前を宣言文（REAL，INTEGER，LOGIC，PROCESSの各ステートメント）で定義して使用する。定数は実数型，整数型の場合はFORTRANと全く同じパターンで書くことができ，論理型の場合にはTRUE(真)，FALSE(偽)がある。プロセス型の場合はプロセスを参照するための名前として扱われる。プロセス定数とでも呼ばれるべきものにNONEがある。NONEは存在しないプロセスを示し，いわば数値の0に相当する。

グループにはセットとキューがあり，それぞれＳＥＴステートメント，QUEUE ステートメントで名前を定義し，そこで与えた名前によって扱う。

グループは配列としても使用できるが，配列は2次元までしか使えない。

システム変数にはSELFとTIMEがあり，SELFはその時点でコントロールが渡っているプロセスを表わし，TIMEはシミュレーション・クロックを表わす。

なお，SIMBOL－Ⅱではデータをその参照のされ方でグローバルデータ，ローカルデータと分ける場合もある。グローバルデータはメイン・プログラムで定義され，全てのプロセスで共通に参照されるものである。これは変数の名前と実体が1対1に対応しており，変数の名前だけで参照できる。一方，ローカルデータはアクティビティ・プログラムで定義され，プロセス個々に取扱うデータ（1つのプロセスの中でお互いのイベント間の情報を受渡すのに用いられるようなデータ）やプロセスの特性を示す属性として参照される。これは定義された名前と実体は必ずしも1対1には対応しない。一般には1つのアクティビティ・プログラムから，複数個のプロセスが発生するのでローカルデータは同じ名前の変数に対して，発生したプロセスの数だけ実体が関係づけられている。ローカルデータの参照に関しては特別な扱い方を許すため，通常の変数の名前とは参照の仕方をはっきり区別している。これに関しては後述，プロセス間コミュニケーションのローカルデータのアクセスを参考のこと。

2.3.4 プロセス間コミュニケーション

SIMBOL－Ⅱでは，シミュレーション・モデルを組み立てる場合，対象とするシステムをプロセ

ス単位に分割した上で，あらためてプロセス間でお互いに関連付けを行う方法をとっている。

プロセス間コミュニケーションとは，プロセスとプロセスの間でお互いに関連付けを行う方法の事を言い，プロセスのローカルデータをアクセスする方法とプロセスを操作する方法の２つに大別できる。後者を具体的に言うと，プロセスを発生したり消去したりする事，セットやキューに入れたり出したりする事，プロセスの実行をコントロールする事，が主な事がらである。

(1) ローカルデータのアクセス

プロセスのローカルデータは，それが離散系プロセスであるか連続系プロセスであるかにかかわらず，どちらのプロセスからもアクセスする事ができる。

ローカルデータの参照の仕方には，エクスターナル・リファレンス（external reference）とローカル・リファレンス（local reference）の２通りある。

エクスターナル・リファレンスとは，プロセスが他のプロセスのローカルデータを参照する場合の方法で，次のように指定する。

プロセス：アクティビティ名・ローカルデータ名

ローカルデータの変数は，同一アクティビティから発生したプロセスでは全て同じなので，これらを区別するためにプロセスを指示する名前を付けて参照する形態をとる。プロセスを指示する名前の所には，単にプロセス変数であってもよいし，プロセス値関数でもよいし，これ自身エクスターナル・リファレンスの形をしていてもかまわないが，その値はプロセスでなければならない。

例えば，アクティビティが

```
DACT  CAR
REAL  WEIGHT, SPEED
  .
  .
  .
END
```

と定義されると，プロセスが発生するたびにWEIGHT，SPEEDに対応するデータエリアが確保される。その関係は次の図のようになる。

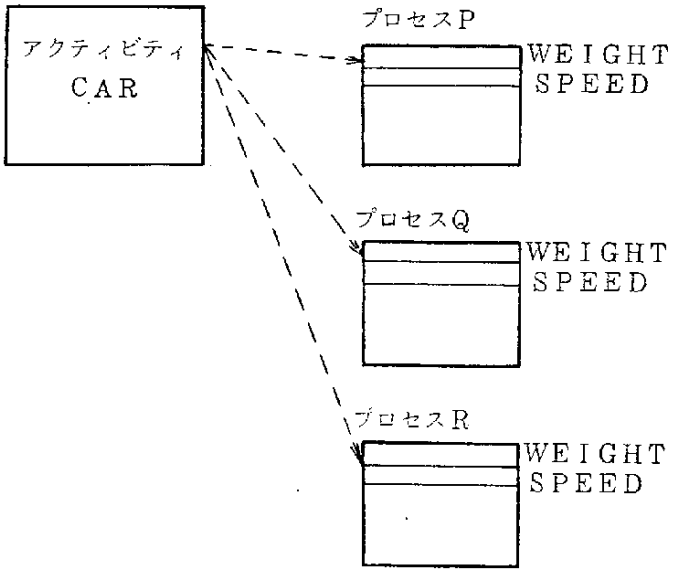


図２－１２　ローカル変数とアクティビティ・プログラム

各プロセスのデータに対してWEIGHT，SPEED という同じ名前がついているので，単に名前だけではどのWEIGHTやSPEED であるか判別がつかず，参照できない。さらにSIMBOL-Ⅱでは，異なったアクティビティにおいても同じ名前を使う事を許している。それらを考慮に入れて，SIMBOL-Ⅱではプロセスのローカルデータを参照するにはプロセス，アクティビティおよび変数の３者の名前を組み合わせて指定することになる。以上の事から，図の例では

プロセスPのWEIGHTを参照するには　P：CAR.WEIGHT

プロセスQのWEIGHTを参照するには　Q：CAR.WEIGHT

プロセスRのWEIGHTを参照するには　R：CAR.WEIGHT

と指定する。

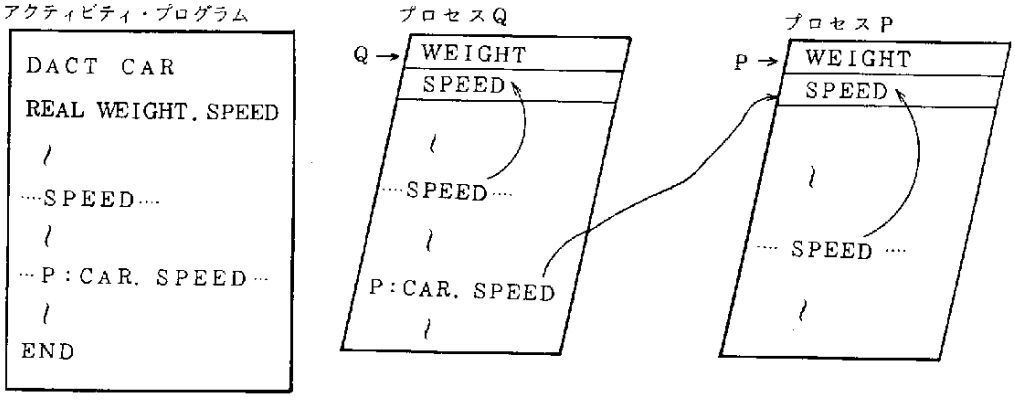


図２－１３　ローカル変数の参照

ローカル・リファレンスとは，ローカルデータを参照する特別なケースで，プロセスが自分自身にローカルなデータを参照する場合の方法である。この場合には，プロセスの属するアクティビティは明らかなのでアクティビティ名は指定する必要がなく，グローバル変数の場合と同じ様にデータ名だけを指定すればよい。

(2) プロセスの発生と消去

プロセスは他のプロセスに依って発生させられるので，メイン・プログラムで一番始めにプロセスを発生させておかなくてはならない。具体的にはSTART ステートメントを用いて，

START アクティビティ名；

とアクティビティ名を指定する。このステートメントが実行されると，指定されたアクティビティ名を持つアクティビティ・プログラムからプロセスが1つ発生し，それがスケジュールテーブルに時刻0でスケジュールされ，エグゼキュータにコントロールが渡される。そして，順次プロセスが発生する。この様な場合以外は，プロセスを定義したアクティビティ・プログラムの名前の前にNEWというオペレータを付けて指定する。実行時にこの項が算定されるとプロセスが発生する。

プロセスは実行がENDステートメントまでたどりつくと自然に消滅するが，他のプロセスを消去することもできる。プロセスを消去するにはDESTROYステートメントを用いるが，次に説明するセットやキューのメンバーになっているプロセスを消去することはできない。

(3) グループの扱い方

グループはセットとキューを総称した呼び方であり，セットもキューも共にプロセスから成る順序のついた集合を扱う手段である。

グループに対しては，要素であるプロセス（メンバーと呼ぶ）を登録したり，削除したりするステートメント（プロセス操作ステートメント）が用意され，シミュレーションの実行中にダイナミックに変わる集合の扱いができるようになっている。

メンバーを参照する方法は種々ある。SIMBOL-Ⅱが用意した関数（プロセス値関数）により先頭のメンバー，最後尾のメンバー，第n番目のメンバーなどを参照でき，これらをプロセス操作ステートメントに用いることもできる。また，位置指定によりグループの先頭，最後尾，特定メンバーの前や後のメンバーを参照することもできる。

以上の点ではセットもキューも同じであるが，キューはセットと異なって，キューを構成する要素が変わるたびに種々の統計情報を自動的にカウントしてくれ，キューをスケジューリングと関連して取扱うためのステートメントが2つ用意されている。

(4) プロセス実行のコントロール

プロセスの実行手順はSIMBOL-Ⅱのスケジューラに従ってコントロールされる。スケジューラはスケジュール・テーブルを持っていて，それを見ながらプロセスを実行していく。スケ

ジュール・テーブルにプロセスを登録したり，削除したりする機能を持ったスケジューリング・ステートメントが用意されている。

シミュレーションの実行が指示されるとメイン・プログラムから実行される。メイン・プログラム中のSTART ステートメントによって，指定のアクティビティ・プログラムからプロセスが発生する。そのプロセスは時刻0でスケジュール・テーブルに登録され，コントロールがスケジューラに渡される。スケジューラはスケジュール・テーブルを見ながら，順次プロセスを発生させたり消滅させたりして実行を進めていく。メイン・プログラムにはRETURNステートメントによって，START ステートメントの次のステートメントに戻り，メインプログラムのＥＮＤステートメントによってシミュレーションの実行が終了する。

2.3.5 ステートメント

(1) ステートメントの種類

SIMBOL-Ⅱのステートメントは，プログラム単位の定義，データの定義，入出力，データ収集関係，代入，実行制御，プロセス操作，スケジューリング，および連続系記述用ステートメントの9種類に大別できる。

① プログラム単位の定義ステートメント

1つのモデルを構成するいくつかのプログラムは，各々その属性を定義しておく必要がある。それを宣言するステートメントとしては次のものがある。

DACT (Discreat Activity) ステートメント

CACT (Continuous Activity) ステートメント

PROC (Procedure) ステートメント

タイプ付きPROC ステートメント

宣言の仕方はどのステートメントも同じで，ステートメント名の後にプログラムの名前を書くことによって行う。

例えば，

```
DACT   CAR ;
DACT   VELOCITY ;
```

の様に書く。

また，プログラムの終りを示すステートメントには

ＥＮＤステートメント

があり，上記の属性を宣言するステートメントと対にして，プログラムの最後に

```
END;
```

と書く。

② データの定義ステートメント

プログラムの中で扱う変数やグループ，テーブルなどは，全部宣言ステートメントによって定義しなくてはならない。それを宣言するステートメントとしては次のものがある。

REALステートメント

INTEGER ステートメント

LOGICステートメント

PROCESS ステートメント

SET ステートメント

QUEUE ステートメント

TABLE ステートメント

FUNCTION ステートメント

宣言の仕方はどのステートメントも同じで，ステートメント名の後にデータの名前を書くことによって行う。

例えば，

```
REAL   A, B, C ;
SET    S(20);
```

の様に書く。

③ 入出力ステートメント

シミュレーション実行中に，キーボードからデータを入力したり，ディスプレイ画面にデータを出力させたりできる様に，次のステートメントがある。

ACCEPT ステートメント

DISPLAY ステートメント

これらは，ステートメント名の後にデータの名前を指定する。

例えば，

```
ACCEPT   A, B ;
DISPLAY  C, D ;
```

の様に書く。ACCEPTステートメントの場合，これが実行されるとAとBに対する値を入力するよう要求されるので，キーボードより値を入力する。また，DISPLAYステートメントの場合は，これが実行されると画面にCとDのその時の値が表示される。

④ データ収集関係ステートメント

テーブルの宣言を行ったものについて，クリアやデータの収集を行うために，次のステートメントがある。

RESETステートメント

TABULATE ステートメント

指定の仕方は，ステートメント名の後に，RESETステートメントではテーブル名のみ，TABULATE ステートメントでは，算術式，テーブル名，重みを書くことで行う。

例えば

```
RESET    TAB1;
TABULATE  A+B, TAB2, 2;
```

の様に書く。

⑤ 代入ステートメント

変数に値を代入するためのステートメントで，次に示すものである。

LET ステートメント

指定の仕方は，ステートメント名の後に代入される変数の名前　代入する値を等合（＝）で結んで指定する。

例えば

```
LET    A=B+C;
```

の様に書く。

⑥ 実行制御ステートメント

プロセスを最初に発生させたり，メイン・プログラムにコントロールを戻したり，コントロールの流れを変えたり，繰り返しを指示したり，実行を中断させたり，プロセデュアを呼び出したりするために，次のステートメントがある。

STARTステートメント

RETURNステートメント

BRANCHステートメント

TRANSFER ステートメント

IF ステートメント

DO ステートメント

LOOPステートメント

PAUSEステートメント

CALL ステートメント

それぞれのステートメント名の後に書く情報は後述してあるので，それを参照されたい。

⑦ プロセス操作ステートメント

プロセスを消去したり，プロセスに名前を付けたり，あるいはグループにプロセスを挿入したり，グループから取り出したりという様な，プロセスを扱うステートメントには，次のステートメントがある。

DESTROY ステートメント

NAME ステートメント

INSERT ステートメント

REMOVEステートメント

指定の仕方は，ステートメント名の後にそれぞれ次の様なものを指定する。DESTROYステートメントではプロセスにつけた名前，NAMEステートメントではプロセス変数名とプロセス式を等合（＝）で結んだものを指定する。また，INSERTステートメントではプロセスの名前と挿入するグループの名前，および挿入する位置を，REMOVEステートメントではプロセスの名前と取り出すグループの名前を指定する。

例えば，

```
DESTROY    P1 ;
NAME TOM:=NEW BABY ;
INSERT NEW BOY ->CLASS LAST;
REMOVE MARY <- CLASS ;
```

の様に書く。

⑧ スケジューリングステートメント

スケジュールテーブルを対象として，プロセスの操作を行うためのステートメントで，次の様なものがある。

DELAYステートメント

SCHEDULE ステートメント

RESCHEDULEステートメント

CANCELステートメント

TERMINATE ステートメント

WAITステートメント

WAKEステートメント

指定の仕方および詳しい機能などについては後述してあるので，それを参照されたい。

⑨ 連続系記述用ステートメント

連続系アクティビティ・プログラムの中において，積分器に関連する式を記述する際に用いるステートメントとして，次のものがある。

INTG ステートメント　（ Integrator ）

TDELAYステートメント　（ Dead Time ）

REALPLステートメント　（ 1st order Lag ）

COMPXPL ステートメント（ 2nd order Lag ）

LEADLG ステートメント(Lead Lag)

指定の仕方は，どのステートメントも形式としては同じで，ステートメント名の後にパラメータを書くことによって行う。

例えば

```
INTG    Y, X, IC ;
REALPL  Y, X, P, IC ;
```

の様に書く。

(2) ステートメントのリスト

●PROCEDUREステートメント

機　能

PROCEDURE ステートメントは，PROCEDUREの宣言を行うステートメントである。

PROCEDURE ステートメントからＥＮＤステートメントまでが，PROCEDUREを構成する。

一般形

PROC　プロセデュア名〔（パラメータリスト）〕;

Syntax Rule

1. プロセデュア名は英字で始まる８文字以内の英数字で，特殊記号は許さない。
2. パラメータは必要なければ省略できる。パラメータリストはパラメータを，（カンマ）で区切って並べたものである。
3. パラメータには，名前だけしか書いてはいけない。つまり配列をパラメータとするときには，その名前だけで配列要素をこの部分に書いてはならない。

●タイプ付きPROCEDUREステートメント

機　能

タイプ付きPROCEDURE ステートメントは，タイプ付きPROCEDUREの宣言を行うステートメントである。

タイプ付きPROCEDURE ステートメントからＥＮＤステートメントまでが，タイプ付きPROCEDUREを構成する。

一般形

タイプ　PROC　プロセデュアー名〔（パラメータリスト）〕;

Syntax Rule

1. タイプはREAL，INTEGER，LOGIC，ELEMENT のうちいずれか１つを許す。
2. プロセデュアー名は英字で始まる８文字以内の英数字で，特殊記号は許さない。
3. パラメータは必要なければ省略できる。パラメータリストはパラメータを，（カン

マ）で区切って並べたものである。

4. パラメータには，名前だけしか書いてはいけない。つまり配列をパラメータとするときには，その名前だけで配列要素をこの部分に書いてはならない。

●DACTステートメント

機　能

DACTステートメントは，Discreat Activityの宣言を行うステートメントである。DACTステートメントからＥＮＤステートメントまでが，Discreat Activityを構成する。

一般形

DACT　アクティビティ名；

Syntax Rule

1. アクティビティ名は英字で始まる８文字以内の英数字で，特殊記号は許さない。

●CACTステートメント

機　能

CACTステートメントは，Continuous Activityの宣言を行うステートメントである。CACTステートメントからＥＮＤステートメントまでが，Continuous Activityを構成する。

一般形

CACT　アクティビティ名；

Syntax Rule

1. アクティビティ名は英字で始まる８文字以内の英数字で，特殊記号は許さない。

備　考

1. 初期設定はプロセスを発生させる側で行う。

●ＥＮＤステートメント

機　能

ＥＮＤステートメントは，PROCEDURE，タイプ付きPROCEDURE，Discreat Activity，Continuous ActivityおよびMain Programの終りを示すステートメントである。

一般形

ＥＮＤ；

●REALステートメント

機　能

REALステートメントは，指定された名前を持つデータが実数型データであることを定義する。

一般形

REAL　名前〔, 名前, …, 名前〕;

例

REAL　A, B, C;

REAL　X(10), Y(3,3);

●INTEGERステートメント

機　能

INTEGERステートメントは，指定された名前を持つデータが整数型データであることを定義する。

一般形

INTEGER　名前〔, 名前, …, 名前〕;

●LOGICステートメント

機　能

LOGICステートメントは，指定された名前を持つデータが論理型データであることを定義する。

一般形

LOGIC　名前〔, 名前, …, 名前〕;

●PROCESSステートメント

機　能

PROCESSステートメントは，指定された名前を持つデータがプロセスであることを定義する。

一般形

PROCESS　名前〔, 名前, …, 名前〕;

●SETステートメント

機　能

SETステートメントは，指定されたデータがセットであることを定義する。

一般形

SET　名前〔, 名前, …, 名前〕;

●QUEUEステートメント

機　能

QUEUEステートメントは，指定された名前を持つデータがキューであることを定義する。

一般形

QUEUE　名前〔，名前，…，名前〕；

●TABLEステートメント

機　能

TABLE ステートメントは，指定された名前を持つデータが統計用のテーブルであることを定義する。

一般形

TABLE　名前＜最小値，最大値，間隔＞
　　〔，名前＜最小値，最大値，間隔＞，…〕；

TABLE　名前〔，名前，…，名前〕；

備　考

・テーブルは配列として宣言することができる。その場合，名前の後に（ m，n ）と配列の大きさを指定する。
ただし，配列は２次元までとする。

・ランクの指定をしたテーブルの場合，次の情報を得ることができる。

各ランクのカウント
分布の平均値
分布の分散
分布の標準偏差
分布の最大値
分布の最小値
サンプル数

・ランクの指定をしないテーブルの場合，次の情報を得ることができる。

分布の平均値
分布の分散
分布の標準偏差
分布の最大値
分布の最小値
データの時間に関する積分値
（これは，$\Sigma X_i(T_i - T_{i-1})$ として求めたもので X_i は時刻 T_i にカウントしたときの値で，T_{i-1} は１回前のカウントした時刻である）

●FUNCTIONステートメント

機　能

FUNCTIONステートメントは，指定された名前を持つデータが関数であることを定義

する。

一般形

FUNCTION　名前，X_1 ，Y_1 ，X_2 ，Y_2 ，……；

備　考

この関数は，名前の後に続くXn，Yn（n＝1，2，…）によって示された各点を結んだ折線の形で示される。

●ACCEPT ステートメント

機　能

ACCEPT ステートメントは，指定された変数に端末装置から入力された値を代入する。

一般形

ACCEPT　変数名〔，変数名，…，変数名〕；

●DISPLAY ステートメントに関する。

機　能

DISPLAY ステートメントは，指定された変数の値を端末装置に出力する。

一般形

DISPLAY 変数名〔，変数名，…，変数名〕；

●TABULATE ステートメント

機　能

TABULATE ステートメントは，指定された算術式の値を，TABLE 宣言ステートメントに従い，指定した量（重み）をテーブルにエントリする。

一般形

TABULATE　算術式，テーブル名〔，重み〕；

備　考

「重み」を省略した場合は「1」とみなす。

●RESET ステートメント

機　能

RESETステートメントは，指定されたテーブルをクリアする。

一般形

RESET　テーブル名〔，テーブル名，…，テーブル名〕；

●LETステートメント

機　能

LETステートメントは，右辺に指定された式の計算を行い，その結果を左辺の数値変数に代入する。

一般形

$$\text{LET}\quad \text{変数} = \left\{ \begin{array}{l} \text{定　数} \\ \text{変　数} \\ \text{算術式} \end{array} \right\} ;$$

●START ステートメント

機　能

START ステートメントは，最初に実行するプロセスに制御を渡す。

一般形

START　アクティビティ名；

●RETURN ステートメント

機　能

RETURN　ステートメントは，メイン・プログラムに制御を戻す。

一般形

RETURN　；

●BRANCH ステートメント

機　能

BRANCH ステートメントは，変数のとる値によって，その値の 1，2，…，n に対応してk1，k2，…，kn のラベルを持つステートメントに制御を移す。

一般形

BRANCH　（k1，k2，…，kn）I；

但し，I：整数形変数

k1，k2，…，kn：ラベル

●TRANSFER ステートメント

機　能

TRANSFER ステートメントは，制御を，指定されたラベルを持つステートメントに移す。

一般形

TRANSFER　ラベル；

●IF ステートメント

機　能

IF ステートメントは，プログラムの流れを式の値によって変更する。

一般形

IF　論理式　ステートメント；

備　考

・ステートメントにはＩＦ，ＤＯステートメントは許さない。

・論理式中の関係演算子としては次のものが許される。

＜	小さい	(less than)
＃＜	小さいか等しい	(less than or equal to)
＃	等しい	(equal to)
／＃	等しくない	(not equal to)
＞	大きい	(greater than)
＞＃	大きいか等しい	(greater than or equal to)

・論理式中の論理演算子としては次のものが許される。

＆	ＡＮＤ
！	ＯＲ
￢	ＮＯＴ

●ＤＯステートメント

機　能

ＤＯステートメントは，ＤＯグループの開始を指示する。

一般形

ＤＯ　　ラベル　ＦＯＲ　変数＝初期値，終値〔，増分値〕；

備　考

・「初期値」，「終値」，「増分値」には算術式を指定できる。

・「増分値」を省略すると「１」とみなす。

・「ラベル」はLOOP ステートメントのラベルと対応していなければならない。

例

```
┌──          DO  LABEL FOR I＝1，１００，１０；
│             ⋮
└─ LABEL     LOOP；
```

●LOOP ステートメント

機　能

LOOP ステートメントは，ＤＯグループの終了を指示する。

一般形

ラベル　　LOOP

備　考

・「ラベル」はＤＯステートメントのラベルと対応していなければならない。

●PAUSEステートメント

機　能

PAUSEステートメントは，モデルの実行を一時中断させる。

一般形

PAUSE ;

●CALLステートメント

機　能

CALLステートメントは，プロセデュアを呼び出し，それに制御を渡す。

一般形

CALL　プロセデュア名 〔（パラメータリスト）〕;

●INSERTステートメント

プロセスをグループ（セットやキュー）に挿入する。

一般形式

INSERT　プロセス　->　グループ　〔位置指定〕;

Syntax Rules

1. 「位置指定」の形式

AT FIRST	..... ①
AT LAST	..... ②
AFTER メンバー	..... ③
BEFORE メンバー	..... ④

2. 「プロセス」としては，一般にプロセス式を書くことができる。
3. 「グループ」はグローバルなグループだけでなく，プロセスにローカルなグループを書いてもかまわない。

備　考

1. 位置指定は，挿入するプロセスをグループのどの位置に入れるかを指定する。

   形式①はグループの先頭に挿入する事を指示している。

   形式②はグループの最後に挿入する事を指示している。

   形式③と④の場合には，そのグループにすでに入っているプロセスを指定して，その直前（④の場合），直後（③の場合）に挿入する事を指示している。
2. 位置指定が省略された場合には，形式②が指定されたものとみなされる。

●REMOVEステートメント

グループに登録されているプロセスをグループから消去する。

一般形式

REMOVE　〔メンバー〕 <－ グループ；

REMOVE　プロセス変数＝メンバー <－ グループ；

Syntax Rules

1. 「メンバー」にはプロセス値関数を書くことができる。

備　考

1. メンバーが省略された場合は，先頭のメンバーが指定されたものとみなされる。

●DESTROYステートメント

プロセスをシステムの場から消去する。

一般形式

DESTROY　プロセス；

備　考

1. デストロイするプロセスは，パッシブまたはターミネイト状態になくてはならない。
2. 対象プロセスは，その時点でグループに属していてはならない。

●NAMEステートメント

機　能

NAMEステートメントは，プロセス式の値をプロセス変数に代入する。

一般形

NAME　プロセス変数名：＝プロセス式；

●DELAY ステートメント

プロセスの実行時間を指定時間（シミュレーション・クロックで）だけ中断する。

一般形式

DELAY　時間；

Syntax Rules

1. 「時間」には一般の算術式が書ける。

備　考

1. このステートメントはアクティビティ・プログラムの中でしか使うことができない。従って，MAINプログラム，プロセデュアーで使うことはできない。
2. 時間の値が負であってはならない。負の値が指定されると，その旨エラー・メッセージをＣＲＴディスプレイに表示し，実行を一時中断し，PAUSE状態になる。
3. このステートメントを実行したプロセスは．アクティブ状態からパッシブ状態となり，スケジュール・テーブルに再登録される。指定した時間後に再び，次のステートメントから実行される。

●SCHEDULE ステートメント

パッシブ状態にあるプロセス（①全く新たにプロセスが発生された時，②過去に少なくとも一度，スケジュールされたがCANCELステートメントなどでスケジュールをキャンセルされている状態にあるプロセス）をスケジュールする。

一般形式

SCHEDULE　プロセス　スケジュール句；

SCHEDULE　プロセス変数：＝NEW ACTIVITY名　スケジュール句；

Syntax Rules

1. スケジュール句の形式

DELAY 算術式 〔, PRIOR〕	…… ①
AT 算術式 〔, PRIOR〕	…… ②
AFTER プロセス	…… ③
BEFORE プロセス	…… ④

備　考

1. スケジュール句の意味

形式①は現在の時刻より「算術式」の値，後にスケジュールする。

形式②は「算術式」の値の時刻にスケジュールする。

形式③は指定したプロセスの直後にスケジュールする。

形式④は指定したプロセスの直前にスケジュールする。

2. 形式①の算術式は0.0以上でなければならない。
3. 形式②の算術式はTIME（システム変数で，シミュレーション・クロックによる現在の時刻）の値以上でなければならない。
4. 形式①と②の場合のみ「, PRIOR」を指定できる。これを指定すると，同時刻の先頭にスケジュールされる。
5. 形式③と④のプロセスは，スケジュールされていなければならない。

● RESCHEDULEステートメント

現在アクティブ，またはスケジュールド状態にあるプロセスをスケジュールしなおす。

一般形式

RESCHEDULE　プロセス　スケジュール句；

Syntax Rules

1. 「スケジュール句」の形式は，SCHEDULE ステートメントのSyntax Rulesに順ずる。

備　考

1. RESCHEDULEの対象となるプロセスは，スケジュールされていなければならない。
2. スケジュール句の意味および注意事項は，SCHEDULEステートメントの備考を参照のこと。
3. 「RESCHEDULE SELF DELAY ××××× 」は「DELAY ××××× 」と同様である。

●CANCELステートメント

現在アクティブ，またはスケジュールド状態にあるプロセスをパッシブ状態にする。

一般形式

CANCEL 〔プロセス〕;

備　考

1. CANCELの対象となるプロセスは，スケジュールされていなければならない。
2. 指定したプロセスがパッシブ状態であれば，影響ない。
3. プロセスを省略した場合は，SELF指定とみなされる。
4. キャンセルされたプロセスは，再びスケジュールされるまで実行されない。

●TERMINATEステートメント

アクティブ，スケジュールド，パッシブ，この内のいづれかの状態にあるプロセスをターミネイト状態にする。

一般形式

TERMINATE 〔プロセス〕;

備　考

1. TERMINATEの対象とするプロセスは，必ずしもスケジュールされている必要はない。
2. プロセスを省略した場合は，SELF指定とみなされる。
3. ターミネイトしたプロセスは，再び実行されることはないが，データとしてシステムの場には残っている。
4. すでにターミネイトしてあるプロセスが指定された場合は，それを無視し，エラーとはならない。

●WAITステートメント

現在アクティブなプロセス（SELF）をキューに入れて，待ち状態（パッシブ）にする。

一般形式

WAIT -> キュー [FIRST | LAST | AFTER メンバー | BEFORE メンバー] ;

Syntax Rules

1. プロセスの指定はできない。

備　考

1. このステートメントを実行したプロセスは，他のプロセスによってスケジュールされないかぎり実行されない。
2. スケジュール後アクティブになると，この次のステートメントから実行が再開される。
3. 位置指定が省略された場合は，LAST指定とみなされる。

WAKE ステートメント

キューに入って待ち状態（パッシブ）にあるプロセスを現在時刻で実行すべくスケジュールする。

一般形式

WAKE 〔メンバー〕<- キュー ;

Syntax Rules

1. 「メンバー」にはプロセス値関数が書ける。

備　考

1. メンバーを省略すると，キューの先頭メンバーとみなされる。

例

WAKE FIRST(S) <- S

ここで，Sはキュー，

FIRSTは先頭メンバーを参照するエレメント関数

このステートメントが実行されると，キューSから，先頭メンバーがとり出され，スケジュールされる。

連続系記述用ステートメントのリスト

名　称	一　般　形	要素の説明	演算子表現
1.積 分 器	INTG Y, X, IC; IC=Y(0)	$Y=\int_0^t X\,dt+IC$	$\frac{1}{S}$
2.時間遅れ	TDELAY Y, X, P, N, A, M; P＝遅れ時間 N＝ステップ数 A＝配列名 M＝配列サイズ	$Y(t)=X(t-p)$, $t\geqq p$ $Y=0$ $t<p$	e^{-ps}
3.一次遅れ	REALPL Y, X, P, IC ; IC=Y(0)	$PY+Y=X$	$\frac{1}{ps+1}$
4.二次遅れ	COMPXPL Y, X, P_1, P_2, IC_1, IC_2; IC_1 $=Y(0)$ IC_2 $=\dot{Y}(0)$	$\ddot{Y}+2P_1P_2\dot{Y}+P_2^2Y=X$	$\frac{1}{S^2+2P_1P_2S+P_2^2}$
5. LEAD-LAG	LEADLG Y, X, P_1, P_2;	$R_2\dot{Y}+Y=P_1\dot{X}+X$	$\frac{P_1S+1}{P_2S+1}$

2.3.6 関　数

SIMBOL-Ⅱシステムには次にあげる関数が用意されていて，LETステートメントあるいはNAMEステートメントと共に使うことができる。

(1) 標準関数

(2) 乱　数

(3) プロセスに関する関数

(4) グループに関する関数

(5) 連続系関数

(1) 標準関数

標準関数としてはEXP, ALOG, ALOG10, SIN, COS, ATAN, ATAN2, SQRT, ABS, IABSの１０種が用意してある。

関数名	書き方	定義	引数の個数	型	
				引数	関数
指数関数	EXP(X)	e^X	1	R	R
自然対数	ALOG(X)	$LOG_e(X)$	1	R	R
常用対数	ALOG10(X)	$LOG_{10}(X)$	1	R	R
正弦関数	SIN(X)	sin(X)	1	R	R
余弦関数	COS(X)	cos(X)	1	R	R
逆正接関数	ATAN(X)	$\tan^{-1}(X)$	1	R	R
	ATAN2(X, Y)	$\tan^{-1}(\frac{X}{Y})$	2	R	R
平方根	SQRT(X)	$\sqrt{X}$	1	R	R
絶対値	ABS(X)	$\|X\|$	1	R	R
	IABS(X)	$\|X\|$	1	I	I

(2) 乱数

乱数としてはRANDM, UNIFORM, NORMAL, NEGEXP, POISSON, RANDL, RANDINTがある。

乱数名	書き方	機能	乱数の型
一様乱数	RANDM(N)	区間〔0, 1〕の一様乱数を発生する	実数型
一様乱数	UNIFRM(A, B, N)	区間A, Bの一様乱数を発生する	実数型
正規乱数	NORMAL(E, S, N)	平均E, 標準偏差Sの正規分布に従う乱数を発生する	実数型
指数乱数	NEGEXP(E, N)	平均Eの指数分布に従う乱数を発生する	実数型
ポアソン乱数	POISSON(E, N)	平均Eのポアソン分布に従う乱数を発生する	整数型
論理値乱数	RANDL(P, N)	確率PでTrue(1)をとり, 確率1－Pで False(0)を値として取る乱数を発生する	整数型
整数一様乱数	RANDINT(I1, I2, N)	区間〔I1, I2〕の整数を等確率でとる整数乱数を発生する	整数型

但し　Nは乱数系列番号（整数型）　Sは標準偏差（実数型）
Aは区間の下限（実数型）　PはTrueをとる確率（実数型）
Bは区間の上限（実数型）　I1は区間の下限（整数型）
Eは平均（実数型）　I2は区間の上限（整数型）

(3) プロセスに関する関数

プロセスに関してはEXISTとPRLORの2種の関数がある。

・EXIST

指定方法

EXIST(S)　ここでSはプロセス名

機　能

プロセスSが存在するか否かを調べ，存在する場合True,存在しない場合Falseを結果とする。

・PRIOR

指定方法

PRIOR(S)　ここでSはプロセス名

機　能

プロセスSの優先順位を調べ，その値を結果とする。

(4) グループに関する関数

グループに関しては，結果が数値，論理値，プロセス値となる関数に分けられる。数値となる関数はGCOUNT，MEMBER，ORDINAL，論理値となる関数はGEXIST，プロセス値となる関数はFIRST，LAST，NUMBER，PRED，SUCがある。

・GCOUNT

指定方法

GCOUNT(S)　ここでSはプロセス名

機　能

プロセスSがいくつのグループに属すかを調べ，その個数を示す数値となる。どこにも属していなければ，0となる。

・MEMBER

指定方法

MENBER(G)　ここでGはグループ名

機　能

グループGにいくつのメンバーが属すかを調べ，その個数を示す数値となる。

・ORDINAL

指定方法

ORDINAL(S, G)　ここでSはプロセス名，Gはグループ名

機　能

プロセスSがグループGの，前から何番目に位置しているかを示す数値となる。

・GEXIST

指定方法

GEXIST(S, G)　　ここでSはプロセス名，Gはグループ名

機　能

プロセスSがグループGに属しているか否かを調べ，属している場合True，属していない場合Falseを結果とする。

・FIRST

指定方法

FIRST(G)　　ここでGはグループ名

機　能

グループGの先頭のプロセスを調べ，その名前を結果とする。

・LAST

指定方法

LAST(G)　　ここでGはグループ名

機　能

グループGの最後尾のプロセスを調べ，その名前を結果とする。

・NUMBER

指定方法

NUMBER(G, N)　　ここでGはグループ名，Nは位置を示す数値

機　能

グループGのN番目の位置にあるプロセスを調べ，その名前を結果とする。

・PRED

指定方法

PRED(S, G)　　ここでSはプロセス名，Gはグループ名

機　能

グループGにおけるプロセスSの直前のメンバーを調べ，その名前を結果とする。

・SUC

指定方法

SUC(S, G)　　ここでSはプロセス名，Gはグループ名

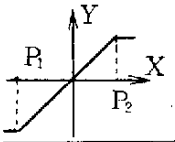
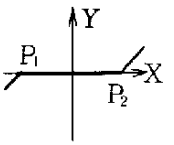
機　能

グループGにおけるプロセスSの直後のメンバーを調べ，その名前を結果とする。

(5) 連続系関数

連続系関数は非線形関数，Switching 関数，論理関数に分けられる。

① 非線形関数

名　称	一 般 形	要素の説明	特　性
LIMITER	$Y=LIMIT(X, P_1, P_2)$	$Y=P_1$　$X<P_1$ $Y=P_2$　$X>P_2$ $Y=X$　$P_1 \leqq X \leqq P_2$	
DEAD SPACE	$Y=DEADSP(X, P_1, P_2)$	$Y=0$　$P_1 \leqq X \leqq P_2$ $Y=X-P_2$　$X>P_2$ $Y=X-P_1$　$X<P_1$	

② Switching関数

名　称	一 般 形	要 素 の 説 明
FUNCTION SWITCH	$Y=FNCSW(P, X_1, X_2, X_3)$	$Y=X_1$　$P<0$ $Y=X_2$　$P=0$ $Y=X_3$　$P>0$
INPUT SWITCH	$Y=INSW(P, X_1, X_2)$	$Y=X_1$　$P<0$ $Y=X_2$　$P \geqq 0$
COMPARATOR	$Y=COMPAR(X_1, X_2)$	$Y=0$　$X_1<X_2$ $Y=1$　$X_1 \geqq X_2$
RESETTABLE FLIP-PLOP	$Y=RES(X_1, X_2, X_3)$	$Y=0$　$X_1>0$ $Y=1$　$X_1 \leqq 0, X_2>0$ $Y=0$　$X_3>0, Y(t-\Delta t)=1$ $Y=1$　$X_1 \leqq 0$ $X_3>0, Y(t-\Delta t)=0$ $Y=0$　$X_2 \leqq 0$ $X_3 \leqq 0, Y(t-\Delta t)=0$ $Y=1$　$X_3 \leqq 0, Y(t-\Delta t)=1$

③ 論理関数

名　　称	一　般　形	要　素　の　説　明
NOT	Y=NOT(X)	Y＝1　X≦0 Y＝0　X>0
AND	Y=AND(X_1, X_2)	Y＝1　X_1 >0, X_2 >0 Y＝0　その他
NOT AND	Y=NAND(X_1, X_2)	Y＝0　X_1 >0, X_2 >0 Y＝1　その他
INCLUSIVE OR	Y=IOR(X_1, X_2)	Y＝0　X_1 ≦0, X_2 ≦0 Y＝1　その他
EXCLUSIVE OR	Y=EOR(X_1, X_2)	Y＝1　X_1 ≦0, X_2 >0 Y＝1　X_1 >0, X_2 ≦0 Y＝0　その他
NOT OR	Y=NOR(X_1, X_2)	Y＝1　X_1 ≦0, X_2 ≦0 Y＝0　その他
EQUIVALENT	Y=EQUIV(X_1, X_2)	Y＝1　X_1 ≦0, X_2 ≦0 Y＝1　X_1 >0, X_2 >0 Y＝0　その他

3. ＳＩＭＢＯＬ－IIのシステム設計

3. ＳＩＭＢＯＬ－Ⅱのシステム設計

3.1 システム設計の方針

コンバインド型のシミュレーション言語を設計するにあたって，ベースとなる離散型・連続型のそれぞれの言語をどのように扱うかが大きな問題となるが，ＳＩＭＢＯＬ－Ⅱはプロセスタイプの離散型シミュレーション言語ＳＩＭＢＯＬをベースとし，これに連続型シミュレーション用の機能を追加する形で設計した。すなわち，ＳＩＭＢＯＬ－ⅡではＳＩＭＢＯＬの離散型プロセスに連続型プロセスという概念を導入し，プロセスタイプのシミュレーション言語という基本思想をそのまま生かす形で設計することにした。

ＳＩＭＢＯＬ－Ⅱはキャラクタ・ディスプレイを使った会話形システムである。一般にシミュレーションの実行には時間がかかるといわれているので，プログラムの実行効率を上げるよう努め，プログラムの入力時の融通性を多少犠牲にし，実行中のモデルと簡単にインタラクションをとれるように考えた。また，ＥＤＩＴの機能についてはハードウェア（ＣＲＴキャラクタ・ディスプレイ装置）の持っている機能を充分に活用する形で，ＳＩＭＢＯＬ－Ⅱのソフトウェアを作成した。そしてキャラクタ・ディスプレイではあるが，グラフや図形を表示することによりユーザがリアクションを速くとれるようにしようということも心がけた。

3.2. システムの構成

ＳＩＭＢＯＬ－ⅡはＦＡＣＯＭ２３０／７５にインプリメントされ，Monitor－Ⅶ（Ｍ－Ⅶ）のもとで稼動する。

Ｍ－Ⅶは幾つかのサブモニタから構成されており，それぞれの業務にあったサブモニタを用いることにより，バッチ処理（センタ・バッチ処理，リモート・バッチ処理），会話形処理，オンライン処理（リアルタイム処理）等を並行して処理できる機能を持っている。

サブモニタの1つであるＣＰＳ（Conversational Programming System）サブモニタのＴＳＳ管理下では，プログラムを事前に登録しておくことにより，必要の都度端末からジョブを起動させることができる。

ＳＩＭＢＯＬ－Ⅱは，このＣＰＳサブモニタを用いて会話形処理を行っている。

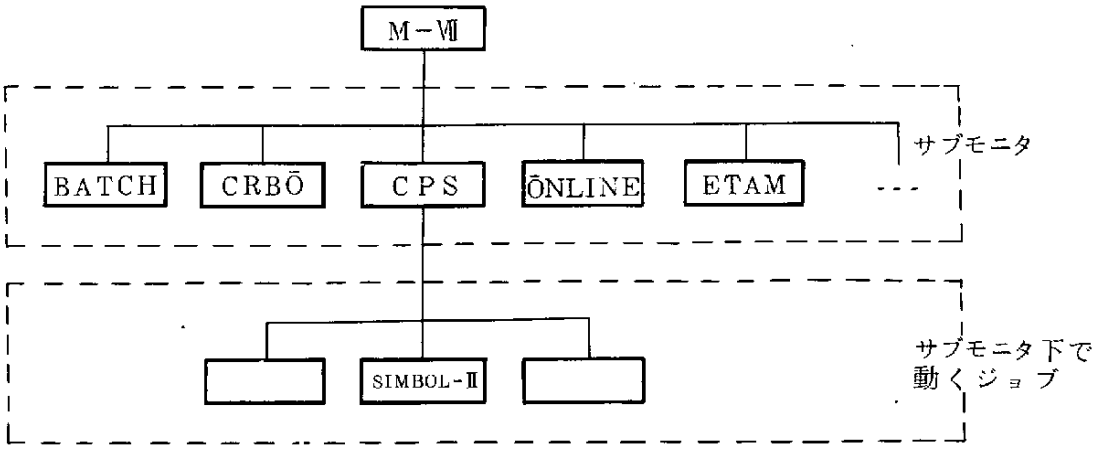


図３－１　Ｍ－Ⅶの構成概略図

3.2.1　中央の機器構成

図３－２にＦＡＣＯＭ２３０／７５の機器構成を示す。

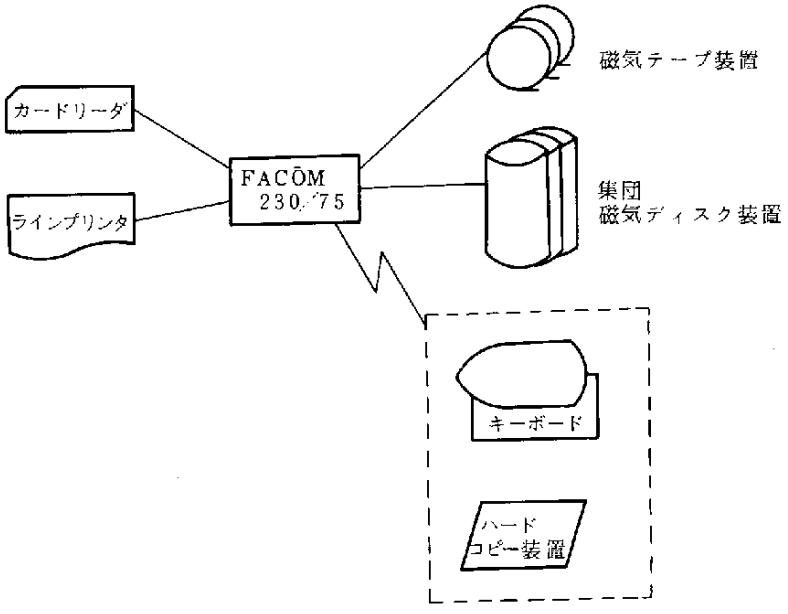


図３－２　ＦＡＣＯＭ２３０／７５機器構成図

以下に，各機器の性能を簡単に記す。

① 主記憶装置

コアメモリー

容　　量	320 K 語
サイクルタイム	1 μs ／ 2 語

バッファメモリー

容　　量	2 K 語
アクセスタイム	45 ns ／ 2 語

② 集団磁気ディスク装置

駆動装置台数	8 台
容　　量	100 × 8 MB
アクセスタイム	34.4 ms
情報転送速度	806 KB ／ S

③ 磁気テープ装置

記憶密度	1600 RPI
トラック数	9 トラック
情報転送速度	192 KB ／ S
テープ速度	120 I ／ S

④ カードリーダ

読取速度	600 枚／m，　2000 枚／m

⑤ ラインプリンタ

印字速度	1260 行／m（英数字）
活字種類	109 字種（英字，数字，記号）
1 行印字数	136 字

3.2.2 端末の機器構成

ＳＩＭＢＯＬ－ⅡはＴＳＳ会話端末であるＣＲＴキャラクタディスプレイ装置と，それに接続しているハードコピー装置を使用する。以下に端末構成図を示し，あわせて各機器の概略を示す。

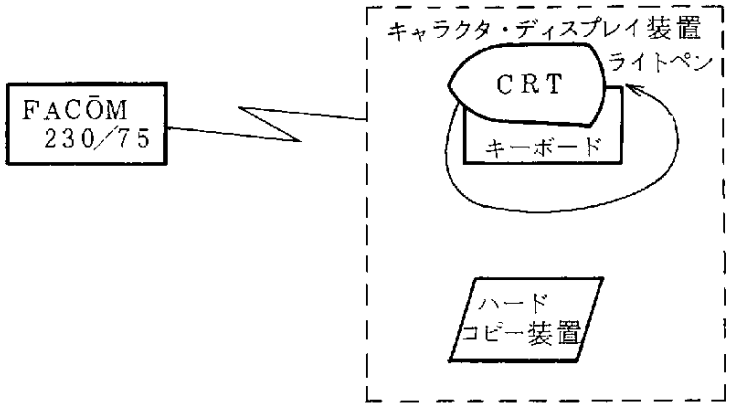


図３－３　端末構成図

(1)　ＣＲＴキャラクタ・ディスプレイ装置

①　ディスプレイスクリーン

画面上に文字を表示する。概略仕様は次の表のとおりである。

表３－１　ＣＲＴ機能仕様

表示文字数	1920字
文字数／行	80字
行数	24行
表示字種	128種（英，数，カナ，記号）
表示文字フォント	7×9ドットマトリクス
表示文字寸法	約2.5㎜（横）×4㎜（縦）
表示色	緑
ＣＲＴ＊の大きさ	17型（17吋）
画面の大きさ	約280㎜（横）×約200㎜（縦）

＊ＣＲＴ：陰極線管（ブラウン管）

②　キーボード

プログラムやデータの入力に使用するキーボードはタイプライタ型であり，キーの配列はＪＩＳ規格による。次にキーの配列図を示し，あわせてキーの種類を簡単に記す。

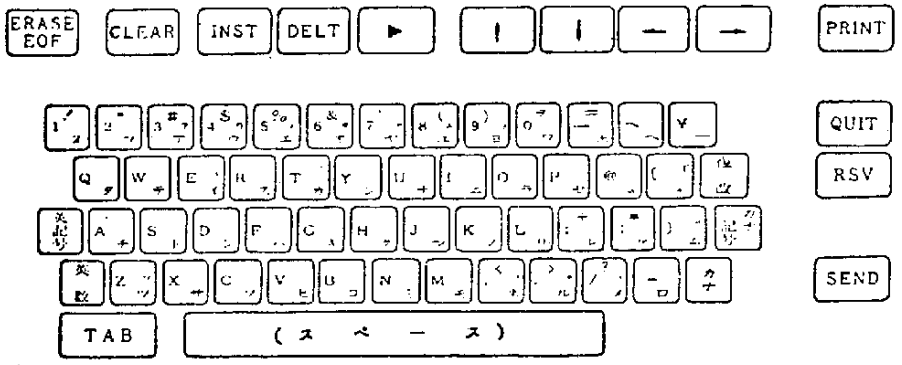


図３－４　キー配列図

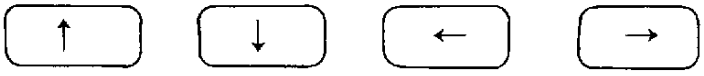
a．文字キー

b．シフトキー

c．スペースキー

d．コントロールキー

• カーソル移動キー



• エディットキー

• フィールド制御キー

• プリンタ起動キー

• メッセージ送信キー

• 伝送ファンクションキー

③　ライトペン

ディスプレイスクリーン上の文字をピックすることにより，文字位置情報を入力する手段で

ある。ＳＩＭＢＯＬ－Ⅱでは，コマンドの選択に使用する。

(2) ハードコピー装置

キーボード上のプリンタ起動キーの押下により，ディスプレイスクリーン1画面分のハードコピーが得られる。

表３－２　ハードコピー機能仕様

印字方式	マルチスタイラス静電記録方式
印字速度	約１６秒／枚
記録面積	約 250 mm (W) × 140 mm (H)
印字文字数	８０字／行
	２４行／枚
文字の大きさ	約 2.2 mm (W) × 2.8 mm (H)
文字フォント	７×９ドットマトリクス
用紙型式	静電記録用シート紙
用紙大きさ	Ａ４シート

3.3 処理概要

3.3.1 プロセッサーの構造

ＳＩＭＢＯＬ－Ⅱプロセッサーはいくつかの大きなモジュールと，サブルーチン群からできている。そのうち主なものを次に挙げ，これらの機能を簡単に後述する。

(1) コマンドディスパッチャー

(2) ステートメントプロセッサー

(3) 各種ステートメント処理サブルーチン

(4) 各種コマンド処理サブルーチン

(5) 各種ディスプレイ表示処理サブルーチン

(6) 算術式処理サブルーチン

(7) プロセス式処理サブルーチン

(8) エディター

(9) エクゼキュータ

(10) スケジューラ

(11) その他のランタイムシステム

上記のモジュール間の関係を図に示すと，図３－５のようになる。

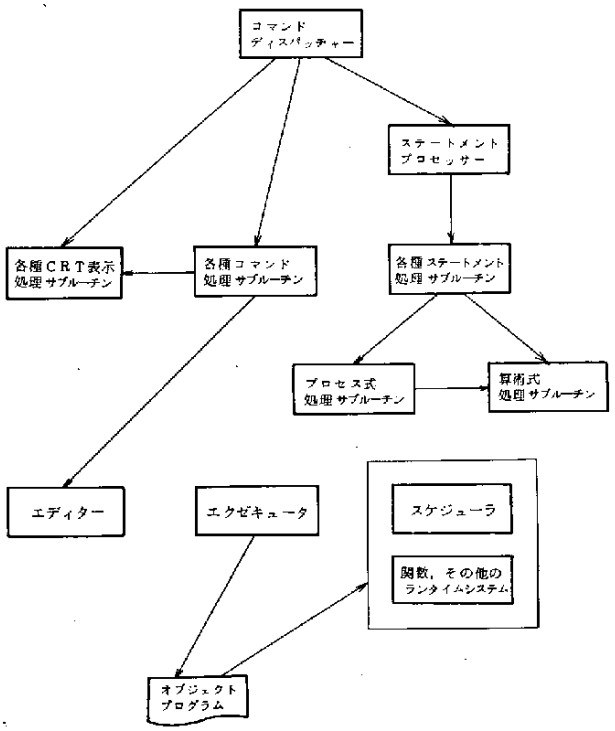


図 ３ － ５

(1) コマンドディスパッチャー

これはＳＩＭＢＯＬ－Ⅱシステムの中核を成すモジュールで，その機能は

① ＣＲＴキャラクタディスプレイ装置からのユーザの入力が，コマンドであるか，ステートメントであるか識別する。

② ステートメントであることが判ると，ステートメントプロセッサーにコントロールを渡す。

③ コマンドであることが判ると，コマンドの種類を識別し，該当するコマンド処理ルーチンにコントロールを渡す。

などである。

(2) ステートメントプロセッサー

ステートメントプロセッサーは，いわゆるインクリメンタルコンパイルの制御モジュールである。このモジュール自身では，ステートメントを解読してオブジェクトコードを作成することは行わない。

このモジュールの機能は，

① 各種ステートメント処理サブルーチンの中の必要なルーチンを呼び出して，オブジェクトコ

ードの作成を依頼する。

② 出来たオブジェクトコードを，自身で管理するオブジェクトエリアにストアし，必要なチェイニングを行なう。

③ ソースエリア（ソース・ステートメントをセーブしておくためのエリア）の管理およびラインナンバーの管理を行なう。

などである。

(3) 各種ステートメント処理サブルーチン

これは，全体で1つのサブルーチンとなっているのではなく，ステートメントごとに1つずつサブルーチンが対応して存在する，サブルーチン群である。

それぞれのサブルーチンは，ステートメントの本体（入力されたラインから，ラインナンバーおよびステートメントラベルを除いた部分）をもらい，そのオブジェクトコードを作成する機能を持っている。コンパイル作業に必要であれば，算術式処理サブルーチンやプロセス式処理サブルーチンを呼び出すこともある。

(4) 各種コマンド処理サブルーチン

これも各種ステートメント処理サブルーチンと同様に，各コマンドごとに1つずつサブルーチンが存在する，サブルーチン群である。コマンドディスパッチャーより，各コマンドのルーチンへ直接コントロールが渡る。

それぞれのサブルーチンは，必要であればコマンドパラメータの入力要求とその解読を行い，コマンド機能の処理を行う。中には，ＥＤＩＴコマンドのように，自分自身では処理せずに，エディターを呼び出して処理を任せる場合もある。

(5) 各種ディスプレイ表示処理サブルーチン

これは，ＣＲＴキャラクタディスプレイ装置の画面に，現在のステージおよびモードを表示したり，使用可能なコマンドを表示したり，コミュニケーションのためのメッセージを表示したりするためのサブルーチン群である。従って，コマンドディスパッチャーや各コマンド処理ルーチンから，必要な時に呼び出される。

(6) 算術式処理サブルーチン

このサブルーチンは，各ステートメントの中に出てくる算術式の処理をするもので，与えられた算術式からそれに相当するオブジェクトコードを作成するためのものである。従って，各ステートメント処理サブルーチンから，必要な時に呼び出される。

(7) プロセス式処理サブルーチン

このサブルーチンも，その名の通りプロセス式の処理をするためのもので，パラメータとしてもらったプロセス式に対してオブジェクトコードを作成する機能を持っている。しかし，プロセス式の処理といってもエクスターナルリファレンスなどの処理は，一部算術式処理サブルーチン

が受け持っており，ここでは主に生成式の処理を行っている。生成式の中には，パラメータ部に制限した形で算術式が書けたりするため，このサブルーチンが算術式処理サブルーチンを呼び出し，その処理を仕せている。

(8) エディター

エディターは，メインメモリーにセーブされているソースステートメントのエディットを行うためのモジュールである。ソースステートメントがセーブされているエリアをソースエリアと呼び，フリー・ストレッジ（Free storage）として管理されている。このエリアに登録されているステートメントは，ラインナンバーで検索できるように，そのためのテーブルが設けられている。エディターはステートメント単位にエディットを行い，個々のステートメントの一部だけを修正したりする機能は持っていない。それはＣＲＴキャラクタディスプレイ装置自身がハード的に持っているエディット機能が使えるので，その必要がないからである。

エディターはステートメント・プロセッサーやＥＤＩＴコマンド処理サブルーチンから呼び出されるが，それぞれの場合によって一部処理が異なっている。

(9) エクゼキュータ

エクゼキュータは，スケジューラと共にモデルの実行をコントロールするためのモジュールである。この両者の機能的な相異は，エクゼキュータはローカルなプログラム・シーケンスに従って実行手順を決めるのに対し，スケジューラはコンカレントに実行されるプロセスの流れを時間軸にそってコントロールする点である。

エクゼキュータの主な機能を次にあげてみると，

① 上記のプログラム実行シーケンスの決定と制御

② トレース処理

③ 統計データの収集機能

などである。なお，エクゼキュータが行う処理について，詳しいことは後述してある。

(10) スケジューラ

スケジューラはランタイムシステムの一部として考える事ができる。インタラクティブなシステムで，ランタイムシステムと言った呼び方は適当でないかもしれない。しかし，一般のステートメントはオブジェクトコードに変換されて，そのオブジェクトコードブロックの中でそのステートメントに対する処理が行われるのに対し，スケジューリングステートメントはオブジェクトコードとして，スケジューラを呼び出す形が作られ，実態の処理はスケジューラが行うので，この様な呼び方をしている。

スケジューラの機能は，プロセスの実行順序をコントロールする事が主なものであるが，スケジューリングに関するトレース処理の一部も受け持っている。スケジューラについては，エグゼキュータと同様に，後に詳しく述べてあるのでここでは省略する。

3.3.2　ステートメント処理

ＣＲＴキャラクタディスプレイ装置から入力されたものがステートメントであると，コマンドディスパッチャーが識別すると，ただちにステートメントプロセッサーへコントロールを渡す。

ステートメントプロセッサーは，まずライナンバーを取り出し，同じものがすでに登録されているか否かを調べる。もしあればそのステートメントを表示し，ユーザの処置を待つ。もしない事がわかれば，ステートメントのコンパイル処理に入る。ステートメントプロセッサーは，後でエディットをしたり，コンパイルをする時のために，ソースステートメントをそのままのイメージでメインメモリーのソースエリアに登録しておく。

ステートメントのコンパイルは，ステートメントプロセッサーがステートメントの種類を識別した後に，各ステートメントごとに用意された処理サブルーチンを呼び出しコンパイル処理をまかせる。

これらサブルーチンは，共通に使うワークエリアにオブジェクトコードを作成し，ステートメントプロセッサーにコントロールを戻す。ステートメントプロセッサーはオブジェクトコード以外に必要な情報を付けて，１つのレコードとしてオブジェクトエリアにセーブし，すでに登録されているこの様なレコードとチェイニングする。

１ステートメントのコンパイルが終了すると，ステートメントプロセッサーはコマンドディスパッチャーにコントロールを戻し，次の入力を待つ。次に入力されたものが，ステートメントであれば再びステートメントプロセッサーにコントロールが渡るが，コマンドであればコマンドディスパッチャーが自分で処理をする。以上の関係を図にまとめて次に示しておく。

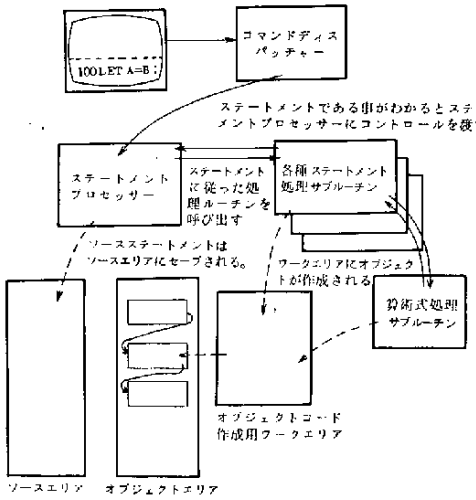


図３－６　ステートメント処理の概略図

3.3.3　コマンド処理

コマンドディスパッチャーは，入力されたものがコマンドであると判別すると，自分自身でこれを処理する。自分で処理するといっても，個々にコマンドに関連する処理はそれぞれの処理ルーチンにコントロールを渡し，完全に任せてしまう。まず，コマンドディスパッチャーはコマンドの種類を識別した後に，各コマンドごとに用意された処理サブルーチンを呼び出し，処理をまかせる。各コマンド処理サブルーチンは，必要であればコマンドパラメータの入力を要求し，入力されたパラメータの解読を行い，その場で，指示に従った処理を行う。

ＳＩＭＢＯＬ－Ⅱのコマンドはサブコマンドを持つ事が許されていて，例えばＥＤＩＴコマンドでは，ＬＩＳＴ，ＤＥＬＥＴＥ，ＩＮＳＥＲＴ，ＲＥＮＵＭＢＥＲなどといったサブコマンドを持っている。この様なサブコマンドの処理も，コマンドディスパッチャーではなく，個々のコマンド処理サブルーチンが一切の処理を行っている。

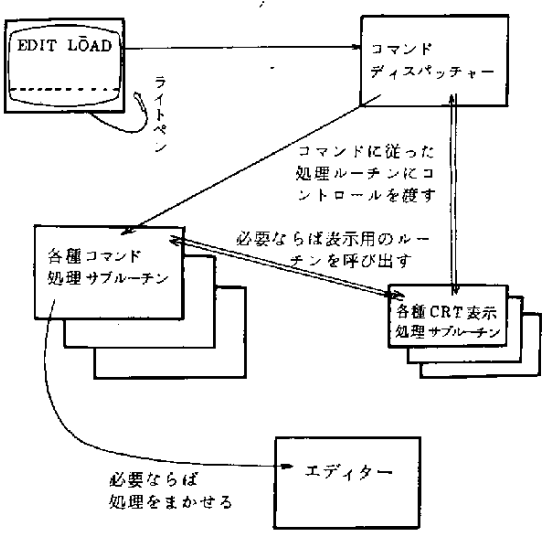


図３－７　コマンド処理の概略図

3.3.4　インクリメンタル・コンパイレーション

ＳＩＭＢＯＬ－Ⅱでは，ＣＲＴディスプレイ装置から入力されるステートメントに対し，いわゆるインクリメンタル・コンパイレーションを行っている。すなわち，ラインが入力されると，それをセーブしておき，プログラム全体のステートメントが出揃ってから，まとめてコンパイルするのではなく．１ステートメントごとにコンパイルをし，オブジェクトコードを作成する方法である。一般に，この様なコンパイルではマシンコードをオブジェクトとして出さずに，ランタイムにインタプリートしながら実行する方法が取られていたが，最近はなるべくマシンコードに近いオブジェク

トを出す方法が用いられる様になってきている。ＳＩＭＢＯＬ－Ⅱの場合にも，実行スピードはできるだけ速くする必要があるので，ほとんどマシンコードのオブジェクトコードを作り出している。

１つのステートメントに対応するオブジェクトコードはメモリーの連続したエリアに作り出されるが，一連のステートメントが必ずしも連続したエリアに作り出されるとは限らない。１つのステートメントがコンパイルされると，オブジェクトコード以外に，このステートメントの種類やトレースに使うスイッチ類，使われている変数名など，コンパイラー自身やエグゼキュータの処理のために必要なステートメント・インフォメーションと呼ばれるものが，オブジェクト・コードの前後に作成される。これは，いわば１つのデータブロック（以下，コードブロックと呼ぶ）の形をしていて，関連するステートメントのコードブロックが，お互いにポインターでチェインされる事によって，１つのプログラムができあがっている。

ここで，ステートメントの順序について考えてみると，ＳＩＭＢＯＬ－Ⅱには次の３種が考えられる。

（ⅰ）ロジカルシーケンス

ステートメントが実行される順序

（ⅱ）フィジカルシーケンス

ラインナンバーに従った順序で，１つ１つのプログラムにおけるステートメントのならびを示す順序

（ⅲ）インプットシーケンス

プログラムのインプット順序

以上のシーケンスに従って言うなら，ステートメントごとにできたコードブロックは，フィジカルシーケンスに従ってチェインされているのである。

(1) オブジェクト作成の基本方針

ＳＩＭＢＯＬ－Ⅱでは，インクリメンタルコンパイレーションの結果作り出されるオブジェクトをインタープリティブ形式と呼んでいる。しかし実際これを実行する際に，シンボルテーブルを参照したり，中間形式になっているオブジェクトをいちいち解読しながら実行する方法はとっていない。コンパイル時や実行時の融通性を損わない程度に，マシン・インタラクションに近いオブジェクトを作る様にしている。

次にオブジェクトコードを作成する上での基本的な方針を列挙してみると，

- １ステートメントごとにコンパイルし，１つのオブジェクトを作り出す。
- 原則として，１ステートメントを実行するごとにエクゼキュータへコントロールを戻し，エクゼキュータが次に実行するステートメントにコントロールを渡す様にしてある。
- 変数に対しては直接アドレスに変換されていて，シンボルテーブルを経由する方法をとらない。
- 算術式や論理式，プロセス式に対しては完全にマシン・インタラクションにおとしてしまう。

従って，ランタイムにオペレータを解読しながら実行する方法をとらない。

- ステートメントラベルの参照は直接行わず，ラベルテーブルを経由する。
- プロセデュアー呼び出しや関数呼び出しについては，直接リンクせず，プログラムテーブルを経由して行う。

などである。

(2) 変数に対する処理

ＳＩＭＢＯＬ－Ⅱで扱う変数には，コンパイルタイムにアドレスを確定できるものと，ランタイムでなくては実際のアドレスが対応づけできないものがある。グローバル変数は，１つの変数名に対して１つのメモリーセルが対応するので，コンパイルタイムに決定してしまう事ができる。しかし，ローカル変数は，それを含むプロセスが実際に作成された時に始めてメモリーが割当てられる。また１つの変数名に対し，１つ以上のメモリーが対応するのが普通である。従って，グローバル変数と同じ様にしてアドレスを決める事はできない。ローカル変数の処理についてはプロセスの処理およびスケジューラの処理とも関連するので後述するので，ここでは一応省略する。ただ，アクティビティ・プログラムで定義されたローカル変数には，相対番地を与える事によってコンパイル時に，この変数と関連するステートメントのオブジェクトコードが作成できる点だけ記しておく。

また，プログラムで使う変数については原則として，使う以前に定義されていなくてはならない。もし定義しないで変数を使うと，その時点でエラーメッセージを出し，変数の定義を要求する。従って，ステートメントをコンパイルする時には，必ず変数は定義ずみとして扱う事ができる。

(3) ステートメントラベルの処理

ステートメントラベルの場合は，変数のときと違って，定義される以前に参照されるケースが起り得る。そのため，例えばＴＲＡＮＳＦＥＲステートメントに対するオブジェクトを出すのに，直接相手ラベルを定義しているステートメントへのジャンプ命令を出す訳にはゆかない。そこで，ラベルテーブルを作り，新たにラベルが出現すると，定義するか参照するかにかかわらず，そのラベルをテーブルに登録し，オブジェクトコードにはラベルテーブルへのジャンプ命令を出しておく。

一方，ラベルテーブルの側では，未定義の場合のエラールーチンへのジャンプ命令を入れておき，未定義のままプログラムが実行されれば，エラールーチンへ飛び込んでいく様にしておく。そしてラベルが定義されると，そのラベルを定義しているステートメントのアドレスを書き込んでおき，プログラムが正常に実行される様にしておく。従って，どちらの場合にもラベルを参照するステートのオブジェクトとしては，ラベルテーブルへのジャンプ命令を出せる事になる。

ラベルテーブルの扱いには，エクゼキュータの処理と密接な関係があるので，エクゼキュータ

の節を参照されたい。なお，概念的な図だけを次に示しておく。

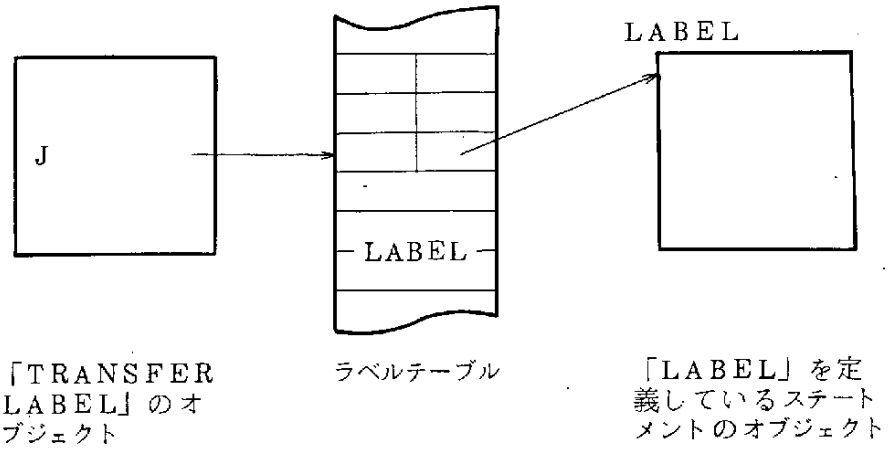


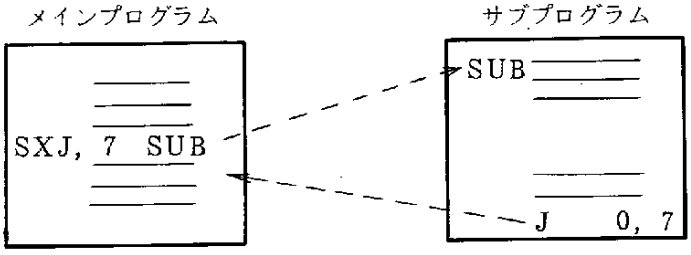
図３－８　ラベルテーブルとオブジェクトとの関連

(4)　プロセデュアと関数の呼び出し処理

プロセデュアや関数はステートメントラベルと同様に，参照するステートメントが出てきた時にすでに定義されているという保障はない。また，ＳＩＭＢＯＬ－Ⅱではランタイムにロードしてリンクする事も許している。そこで，この処理もラベルの処理と似た方法をとらなくてはならない。

プロセデュアや関数が参照されたり，定義されたりすると，プログラムテーブルに登録される。このテーブルにはプロセデュアや関数に対して１つずつアイテムが対応して，その名前，定義済であれば入口番地などを示す情報が記入されている。

オブジェクト・プログラムの中でサブルーチンコールは，ＳＸＪ命令（ＦＡＣＯＭ２３０／７５ＦＡＳＰ　Set Index and Jump 命令）が使われるが，この命令は指定したインデックスレジスターにこの命令の次のアドレスをセットし，アドレス部で指定した番地にジャンプする機能を持っている。



（説明）インデックス７に，つぎの命令の番地をセットし，ＳＵＢという番地へジャンプし，そこである処理をしてから，再びメインプログラムに戻る例である。

図３－９　ＳＸＪ命令の例

そこで，プロセデュアや関数の呼び出し側のオブジェクトとしてはＳＸＪ命令を出し，ジャンプ先をプログラムテーブルの対応するアイテムにしておくことにより，正常に実行される。このプログラムテーブルの扱いも，ラベルテーブルと同様にエクゼキュータの処理と密接な関連があるので，後述する。

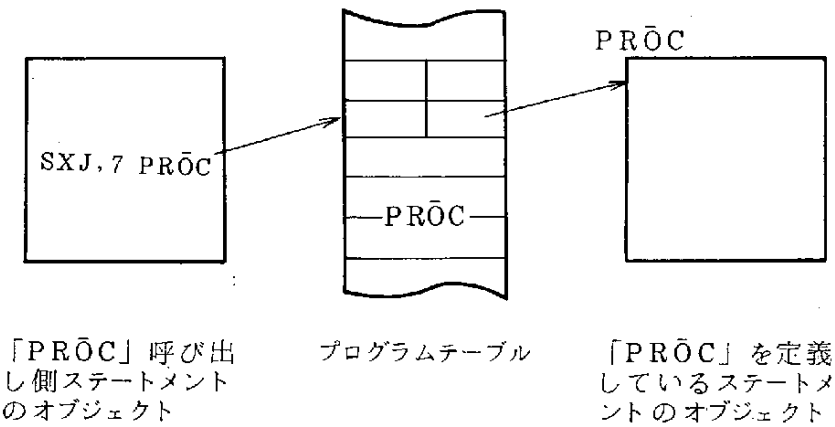


図３－10　プログラムテーブルとオブジェクトとの関連

3.3.5　プロセスコントロールブロック

アクティビティ・プログラムよりプロセスが発生すると，フリーストレッジにプロセスコントロールブロック（以下ＰＣＢと略記する）が作られる。ＰＣＢにはプロセスをコントロールするために必要な情報が記入されている。離散型ＰＣＢと連続型ＰＣＢとは多少情報に相異はあるが，共通なものを挙げると，

1. プロセスの発生時刻
2. プロセスのステータス
3. リアクティベイションポイント
4. プライオリティ
5. 最後にアクティブになった時刻
6. インベントノーティスへのポインター
7. ＰＣＢブロックサイズ
8. アクティビティテーブルへのポインター
9. プロセス番号
10. グループカウント

11. プロセス管理用スイッチ

などがある。ＰＣＢにはこれ以外に，プロセスにローカルなデータがあれば，そのためのエリアがとられている。実際には図３－11に示すように固定長の部分と可変長の部分の２つから成っている。

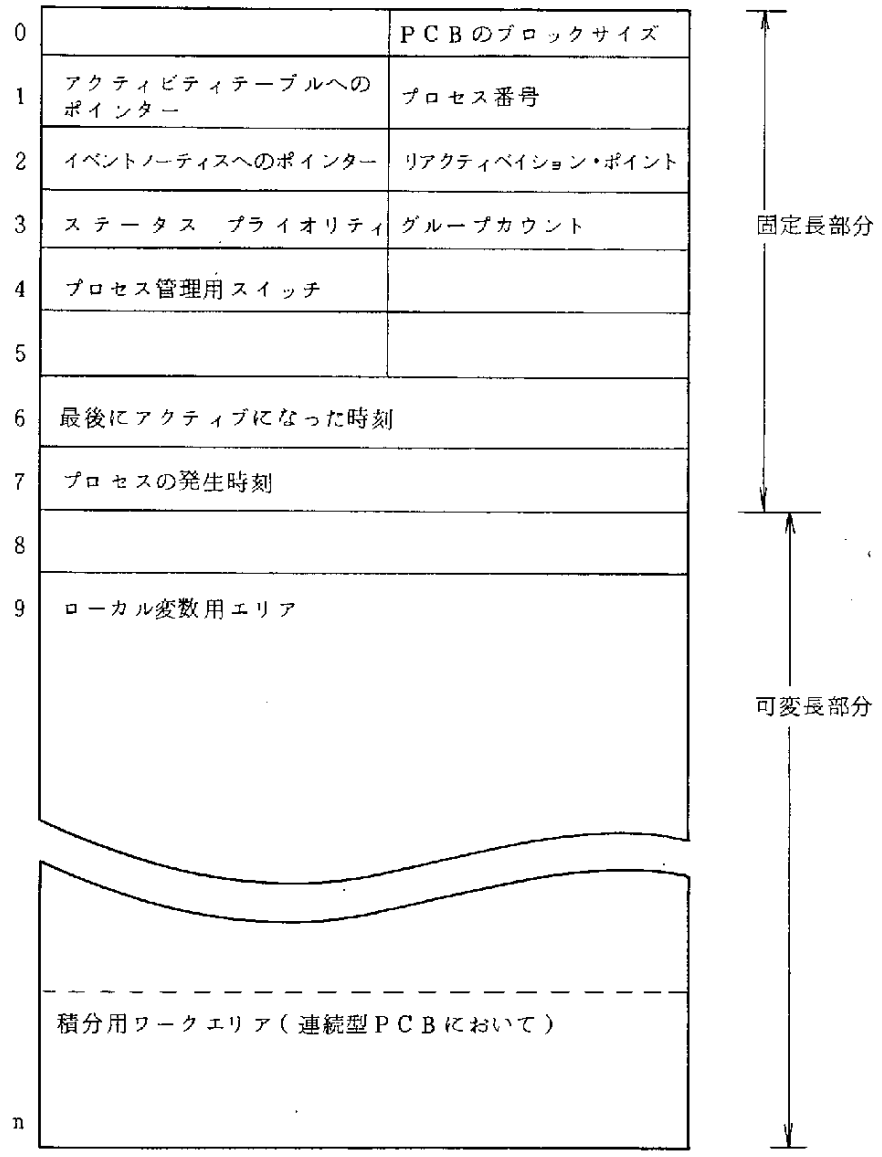


図３－11　ＰＣＢのフォーマット

ＰＣＢの固定長部分には前記の情報が書き込まれており，可変長部分にはローカル変数のためのエリアが割当てられている。連続型プロセスのＰＣＢにおいては，可変長部分に積分用のワークエリアも割当てられている。

固定長部分は，どのアクティビティに属するプロセスについてもそのサイズは一定（固定長）であるが，可変長部分はローカル変数の数によって変わるので，プロセスが属すアクティビティが異なればサイズも異なるのが普通である。もちろん同アクティビティに属するプロセスの可変長部分は同じサイズである。また，ローカル変数をぜんぜん持たないプロセスについては，可変長部分は作成されない。

離散型ＰＣＢおよび連続型ＰＣＢのフォーマットを，次に示す。

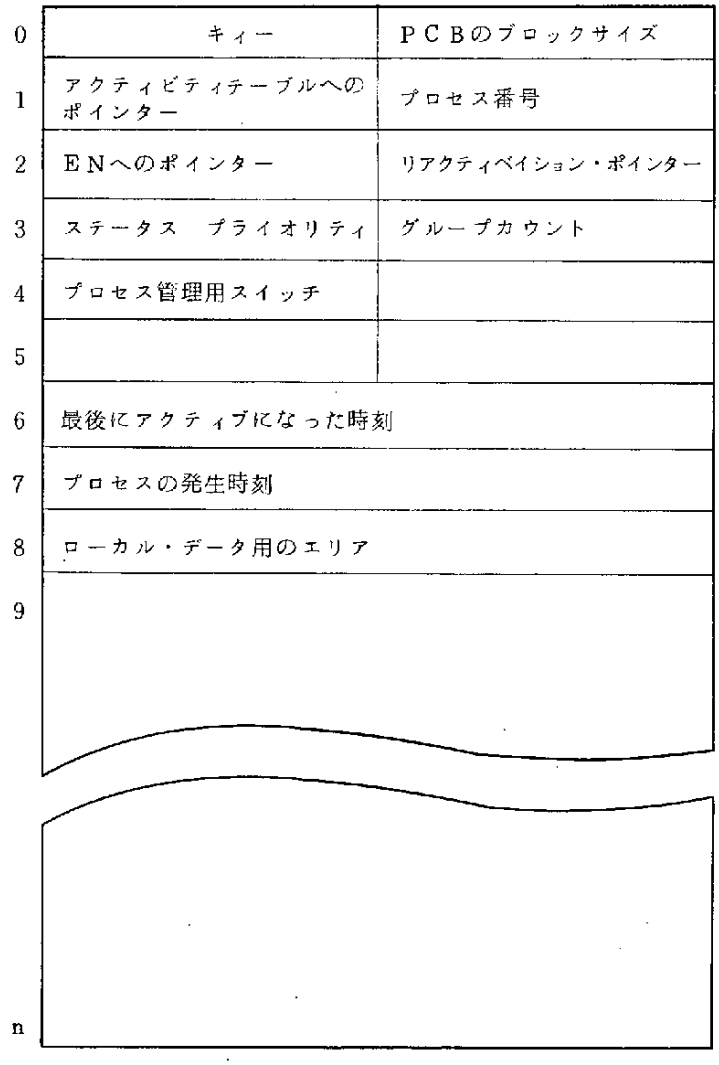


図３－12　離散型プロセス・コントロール・ブロック

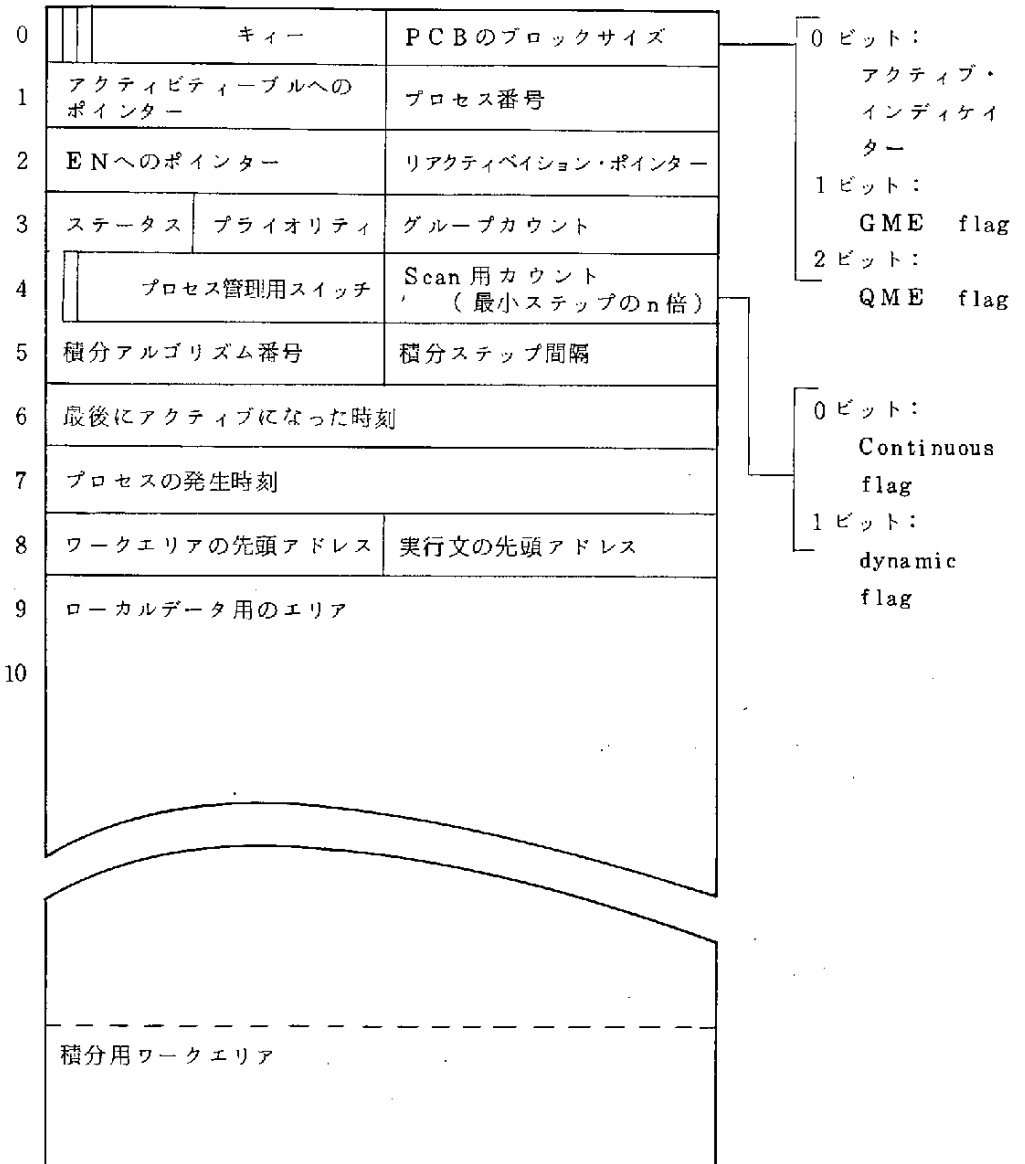


図3－13　連続型プロセス・コントロール・ブロック

固定長部分の情報のうち，リアクティベイションポイントはアクティビティ・プログラムにおいて，次に実行するラインを示すポインターである。また，イベントノーティスへのポインターは，プロセスがスケジュールされると対応して作成されるイベントノーティスを示すポインターである。従って，プロセスがスケジュールされていない時には全く意味を持たない情報である。

可変長部分では，アクティビティ・プログラムで宣言されたローカル変数に対し，宣言された順に可変長部分の先頭から相対番地が割当てられる。配列やセット，キューなどがローカルに定義された場合には，もちろん可変長部分上には各々に必要なサイズ分のメモリーが割当てられるので，次に定義される変数の相対番地は，それまでに定義された変数の数ではなく，メモリーサイズ分だけ取られた値となる。（図３－14参照のこと）

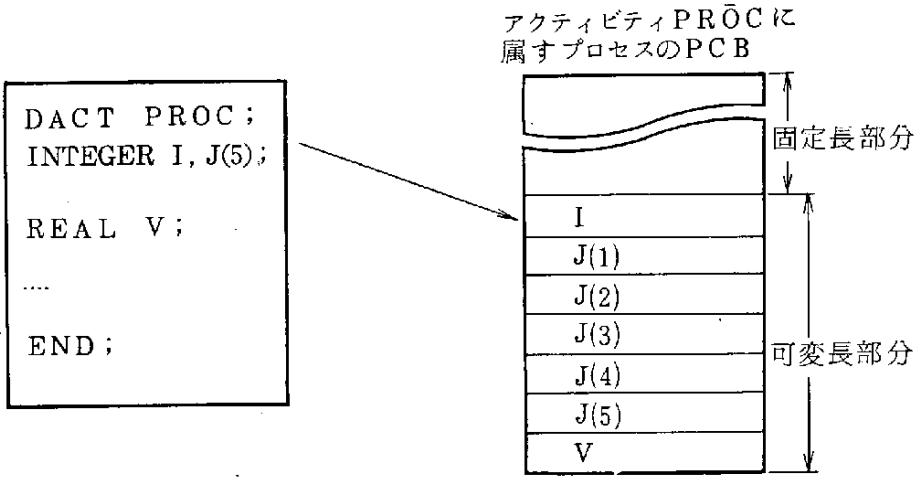


図３－14　ローカル変数の割付け

ローカル変数については，その変数が宣言されると同時にＰＣＢ可変長部分における相対番地を決めることができる。セットやキューがプロセスにローカルに宣言された時でも，ヘッドが可変長部分に取られるので，ヘッドの先頭位置を同じ方法で与える事ができる。そこで，ローカル変数を使っているステートメントに対しては，特定のインデックスによるモディファイと可変長部分における相対番地とによって，オブジェクトコードを出すことが可能となる。

(i)　ローカルリファレンスの場合

ローカル変数を参照するのは，今実行しているプロセスのはずであり，スケジューラがこのインデックスに，プロセスが変わる度にその可変長部分の番地をセットしているので，ローカル変数を参照できる。

(ii)　エクスターナルリファレンスの場合

エクスターナルリファレンスのオブジェクト自身に，そのプロセスのＰＣＢ番地を求める部分が作られており，インデックス（ローカルリファレンスの場合とは別のインデックス）にオブジェクト自身がセットしている。従って，この場合も対象としているローカル変数をまちがえなく参照することができる。

3.3.6　グループの取扱い

(1)　セットとキューの構造

セットとキューはそれぞれセットヘッド（Set head）とキューヘッド（Queue head）を基点として，これらを構成するメンバーがチェインされ対称リスト構造になっている。これらのメンバーとなるのはプロセスであるが，プロセス自身にポインターを持たせ，直接リンクする方法はとっていない。セットやキューに登録されたプロセスに対して，セットであればセットメンバーエレメント（ＳＭＥと略記する），キューであればキューメンバーエレメント（ＱＭＥと略記する）が作成され，プロセスの代りにこれらがお互いにチェインされている。その様子を次図に示す。

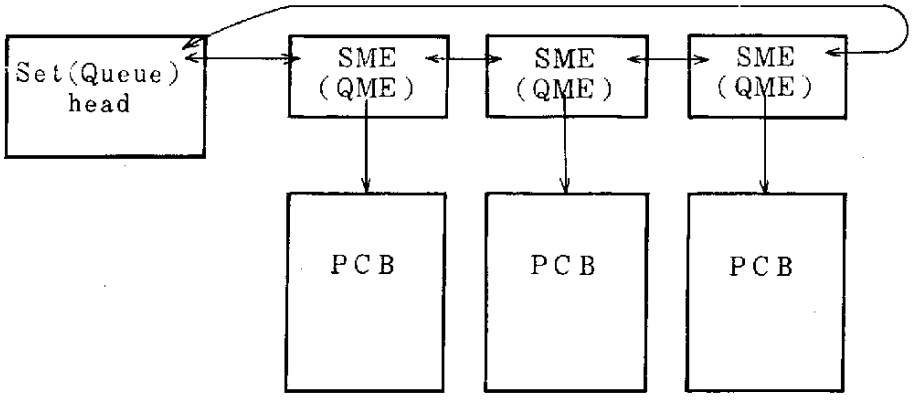
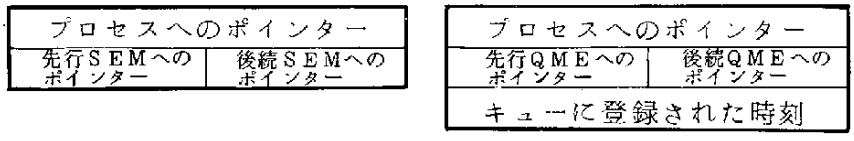


図３－15　セット（キュー）の構造

ＳＭＥとＱＭＥはそれぞれ２ワード，３ワードで構成されている。次にその構造を示す。



ＳＭＥの構造　　ＱＭＥの構造

図３－16

また，セットやキューの基点となるヘッドは，セットの場合は３ワード，キューの場合は９ワードで構成されている。その構造を次に示す。

セットヘッドを示すＩＤ	
先頭ＳＭＥへのポインター	最終ＳＭＥへのポインター
現在のメンバー数	

図３－17　セットヘッドの構造

キューヘッドを示すＩＤ	
Ｑ.ＭＥＭ　：現在登録されているメンバー数	
Ｑ.ＦＩＲＳＴＭ：先頭メンバーのポインター	Ｑ.ＬＡＳＴＭ：最終メンバーのポインター
Ｑ.ＴＭＥＭ　：それまでに登録されたメンバーの総数	
Ｑ.ＭＡＸＷＴ：最大待ち時間	
Ｑ.ＺＷＴ　：待ち時間の総和	
Ｑ.ＭＡＸＱＬ：最大待ち数	
Ｑ.ＺＥＲＯＥ：待ちなしで出たメンバー数	
Ｑ.ＬＡＴＩＭＥ：最新のアクセスタイム	
Ｑ.ＺＱＬ　：Ｑ.ＭＥＭ×△ｔの総和	

但し　△ｔ＝現在の時刻－Ｑ.ＬＡＴＩＭＥ　の意味

図３－18　キューヘッドの構造

(2)　メンバーの登録と削除

セットやキューにメンバーを登録したり削除したりする事は，ＩＮＳＥＲＴステートメントやＲＥＭＯＶＥステートメントが実行されると行われる。これらの扱いは，INSERTやREMOVEステートメントに対するオブジェクトがランタイムシステムのサブルーチンコールの形で出されているので，全てランタイムシステムで行っている。

新しくセットまたはキューにプロセスが登録される時には，ＳＭＥやＱＭＥを作り出さなくてはならない。これらはフリーストレッジにＳＭＥ同士，ＱＭＥ同士がそれぞれお互いにチェインされ，まとめて管理されている。必要な時に取り出し，使い終れば再びフリーストレッジに返却される。従って，ＳＭＥやＱＭＥはメンバーが登録される際に作成され，メンバーが消去されると消去される。ＷＡＩＴステートメントやＷＡＫＥステートメントなどが後に説明するイベントノーティスとＱＭＥを兼ねて使用するのは，その都度フリーストレッジから取り出したり返却する手間をはぶき，実行スピードを上げるための処置である。

①　セットの場合

セットに新しくメンバーを登録する時にはフリーストレッジからＳＭＥをもらい，登録する

プロセスとのリンクを付ける。その後ＳＭＥをセットの必要な場所に挿入し，ＳＭＥ同士のチェイニングを行う。そして，メンバーが１つ増えたので，セットヘッドにあるメンバー数を示す部分をアップデートする。

セットからメンバーが取り除かれる場合には丁度これとは逆に，対応するＳＭＥをセットから取り出し，チェインを修正した後にフリーストレッジに返却する。そしてセットヘッドのメンバー数が１だけ減らされる。

② キューの場合

キューにメンバーを登録したり，削除する場合には，セットと同じ様な処理の他に簡単な統計処理を行わなくてはならない。

キューに登録される場合はＱＭＥをフリーストレッジからもらいプロセスとリンクを取った後，ＱＭＥに登録時刻を書き込む。この時刻はその時点でのＴＩＭＥの値が入れられる。ＱＭＥはセットの時と同じ様にキューに挿入される。登録時にキューヘッドの各アイテムに対して，次の様な処理が行われる。

(i) Ｑ．TMEMを１だけ増す。

(ii) Ｑ．ＭＥＭを１だけ増す。

(iii) Ｑ．ＭＡＸＱＬとＱ．ＭＥＭを比較して，Ｑ．ＭＥＭが大きければＱ．ＭＡＸＱＬにＱ．ＭＥＭを入れる。そうでなければそのままにまる。

(iv) Ｑ．ＭＥＭと（ＴＩＭＥ－Ｑ．ＬＡＴＩＭＥ）の積を計算して，Ｑ．ＺＱＬに加える。

(v) Ｑ．ＬＡＴＩＭＥにＴＩＭＥを入れる。

キューからメンバーが削除される時には，セットの時と同様な処理以外に，登録の時と同じくキューヘッドに対して次の処理が行われる。（削除されるメンバーの，キューに登録された時刻をＩＴで表わす。）

(i) ＩＴ＝ＴＩＭＥなら，Ｑ．ＺＥＲＯＥを１だけ増し，処理を終る。そうでなければ (ii) 以下の処理を行う。

(ii) Ｑ．ＺＷＴに（ＴＩＭＥ－ＩＴ）を加える。

(iii) Ｑ．ＭＡＸＷＴと（ＴＩＭＥ－ＩＴ）を比較し，Ｑ．ＭＡＸＷＴが小さければ（ＴＩＭＥ－ＩＴ）をＱ．ＭＡＸＷＴに入れ，そうでなければなにもしない。

3.3.7 エグゼキュータの処理

モデル，即ちＳＩＭＢＯＬ－Ⅱによって作られたオブジェクトプログラムは，ＳＩＭＢＯＬ－Ⅱプロセッサーが介入して実行する。この場合，具体的には主にエクゼキュータがこれを行う。このような形式では，メイン・プログラム，アクティビティ・プログラム，プロセデュアや関数はインタープリティブ形式（以下Ｉ形式と略記する）のオブジェクトである。

I形式のプログラムでは，原則として1ステートメント実行する度にエグゼキュータにコントロールが戻り，エグゼキュータが次に実行すべきステートメントを決めてそこにコントロールを渡すと言ったルールで行われる。もちろん，これはローカルシーケンスに従って実行する時の事であって，スケジューリングと関連して途中から他のプロセスにコントロールが移される様な場合には，間にスケジューラが介入する事になる。しかし，一度，他のプロセスにコントロールが渡っても，そこからはまたローカルシーケンスに従って実行される事になるので，エグゼキュータが再び実行シーケンスの決定を行うことになる。この様な繰り返しによってモデルの実行が行われる。これらの関係を概念的に示したのが次の図である。

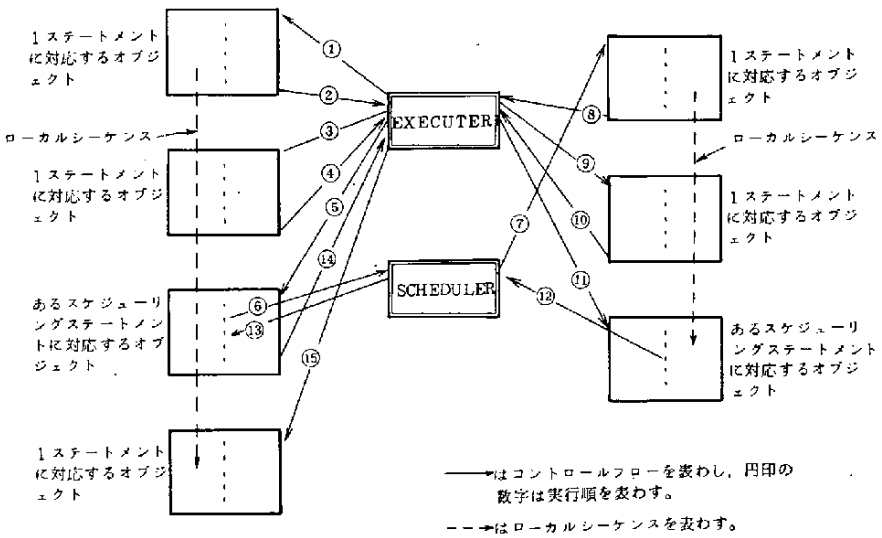


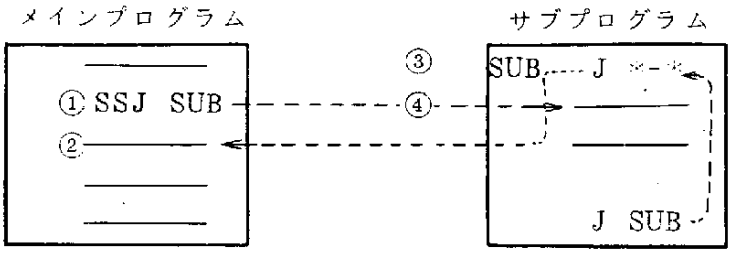
図3－19　モデルの実行過程

エグゼキュータは，ステートメントの種類によって，違ったルールで次に実行すべきステートメントを決定する様な事をなるべく避け，どのステートメントであっても同じルールで，単独にコントロールを渡す場所が求められる事が望ましい。この様な理由から，SIMBOL－Hではステートメントの方で少し工夫をし，エグゼキュータは相手がどんなステートメントであっても，同じルールで次に実行するステートメントを決められる様になっている。実行手順の変化をもとにステートメントを分けると，(i) 実行手順を変更する事のないステートメントの場合，(ii) ラベルテーブルを経由する場合，(iii) プログラムテーブルを経由する場合，の3つとなる。以下に，それぞれの場合の方法を説明する。

(i)　実行手順を変更する事のないステートメントの場合

LETステートメント，INSERTステートメント，REMOVEステートメントなど実行手順を変更する事のないステートメントでは，ステートメント実行後に，エグゼキュータに

SSJ命令でリターンする。ＳＳＪ（ Set S and Jump ）命令はサブルーチンリンクのために使われるＦＡＣＯＭ２３０／７５のＦＡＳＰの命令で，この命令を実行した次の番地をジャンプ先のアドレス部にセットし，その次の番地にコントロールを渡す機能を持っている。



（説明） ①の命令を実行して，②の番地を③のアドレス部にセットする。（つまり，＊－＊と置きかえたのと同じ。）そして④の番地から実行を開始する。従って，サブプログラムの最後で③にジャンプさせ，③を実行させれば，再びメインプログラムの②にもどる。

図３－20　ＳＳＪ命令の例

従って，ＳＳＪ命令の次の番地にフィジカルシーケンスに従って，次のステートメントがあるアドレスを書き込んでおけば，エグゼキュータは簡単に参照することができる。次の図は，この関係を示している。

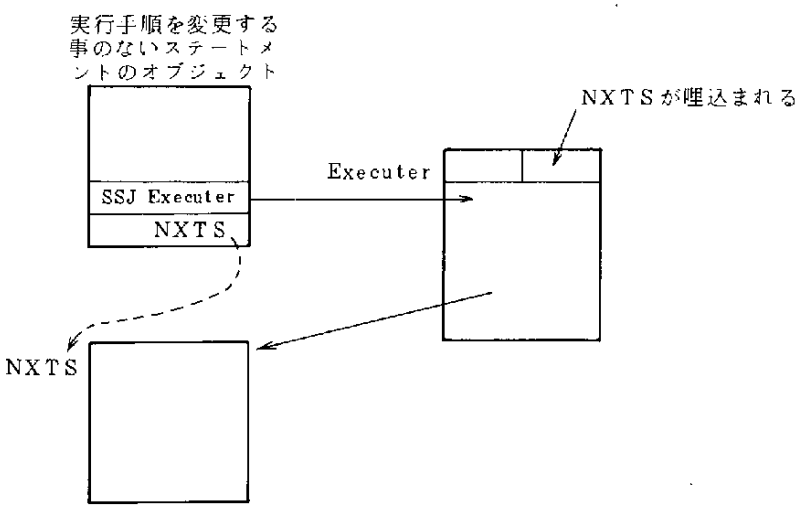


図３－21　実行手順を変更する事のないステートメントのオブジェクトとエグゼキュータの関係

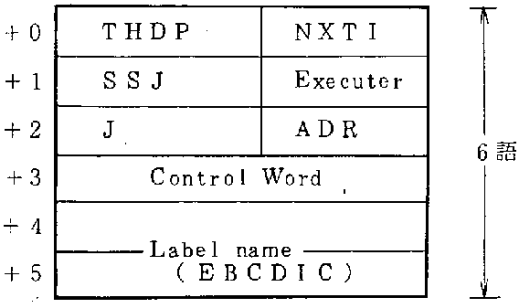
(ii)　ラベルテーブルを経由する場合

ＴＲＡＮＳＦＥＲステートメントに対するオブジェクトは無条件ジャンプ命令で，対応するラベルテーブルのアイテムへ飛び込む様になっている。

ラベルテーブルには次の図で示す通りＳＳＪ命令があり，アドレス部にはエグゼキュータへの入口番地が書き込まれている。

Label Table

Label Table itemのFormat



THDP：テーブルヘッドを示すポインター（ Absolute ）

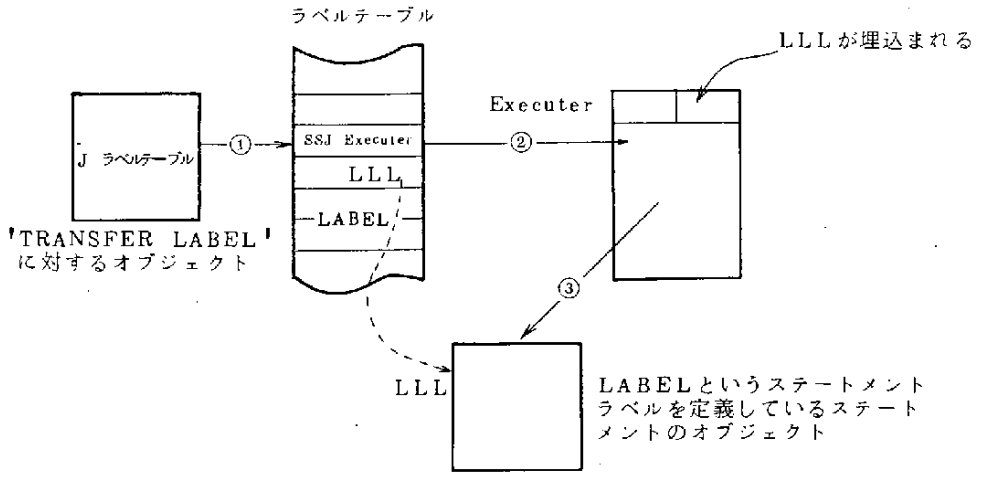
NXTI：次のitemを示すポインター（ SM．IDTABLからの相対番地）

ＡＤＲ　：実効番地またはＬＩＮＫ　ＦＡＵＬＴ　ＨＡＮＤＬＥＲの入口番地

Control Word：第0 bit ｛ ONならば定義済み
　　OFFならば未定義

図３－22　ラベルテーブルアイテムの形式

エグゼキュータからＴＲＡＮＳＦＥＲステートメントにコントロールが渡されると，ジャンプ命令を実行するので，ラベルテーブルのＳＳＪ命令にコントロールが渡る。次にＳＳＪ命令が実行される事になり，エグゼキュータにコントロールが戻される。ＴＲＡＮＳＦＥＲステートメントに書かれたラベルは，定義されると，そのアドレスがＳＳＪ命令の次のアドレスに書き込まれているので，普通のステートメントと同じルールで次に実行するステートメントを決めれば，TRANSFER ステートメントが示すステートメントにコントロールを渡す事ができる。この関係を次の図に示す。



→は実行の流れを示し，円内の数値は実行順を示す。

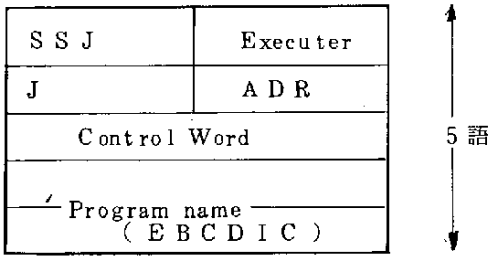
図3－23　TRANSFERステートメントのオブジェクトとエグゼキュータの関係

(iii)　プログラムテーブルを経由する場合

プロセデュアコールに対して出されるオブジェクトプログラムについては前にも述べたが，SXJ命令のジャンプ先をプログラムテーブルの対応するアイテムにしてあり，そのアイテムがサブルーチンであるかの様にして扱われる。

Program Table

Program Table item の Format



ADR：実効番地またはLINK FAULT HANDLERの入口番地

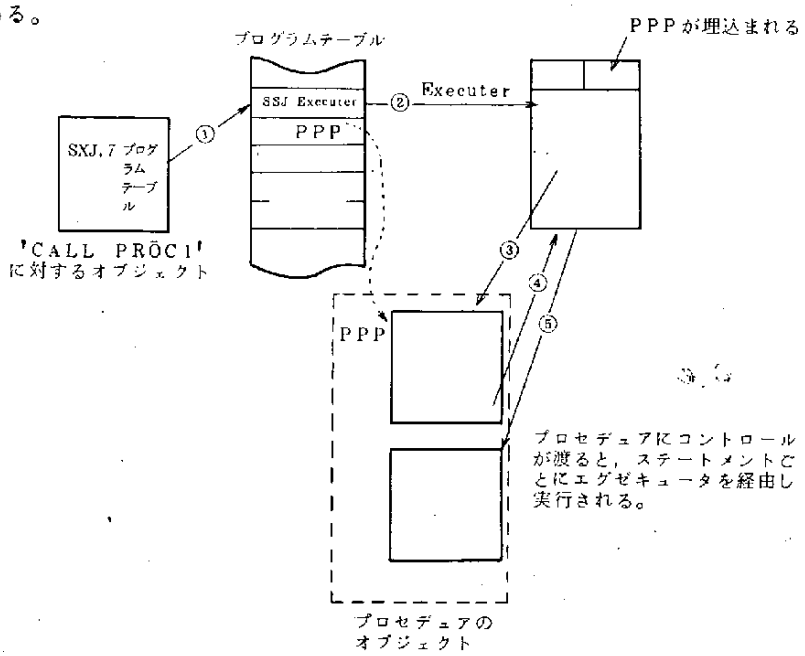
Control Word：第0 bit　{ ONならば　定義済み / OFFならば未定義

図3－24　プログラムテーブルアイテムの形式

プログラムテーブル側ではエグゼキュータへのSSJ命令が出されている。次のアドレスに実際のプロセデュアの入口番地が書き込まれている。もちろん，これは入口番地が定義されている場合で，まだ未定義である場合はリンクフォールトハンドラーのアドレスが植込まれている。従

って，もしこの様な状態でプロセデュアコールが行われると，リンクフォールトハンドラーに飛び込む事になる。もし，正常に定義されていれば，呼び出したいプロセデュアにコントロールを渡す事ができる。

仲介となるプログラムテーブルに植込まれている命令がＳＳＪ命令であるために，インデックスの値は変わらず，プロセデュア側でパラメータのあるアドレスや復帰番地は，直接呼ばれた時と同じ様にインデックスを参照して求める事が可能となる。次の図は，以上の関連を示したものである。



→は実行の流れを示し，円内の数値は実行順を示す。

図３－25　プロセデュアコールのオブジェクトとエグゼキュータの関係

3.3.8　スケジューラの処理

スケジューラが行う基本的な処理は，スケジュールテーブルを参照して次に実行するイベントを決定することおよび連続系プロセスの実行に関連する処理を行うことである。実際には，

(i)　システム・クロックの管理

(ii)　スケジュールテーブルの管理

(iii)　プロセス・コントロール・ブロック（ＰＣＢ）の更新（リアクティベイションポイントなど）

(iv)　プロセスにコントロールを渡すための準備

(v)　スケジューリングと関連したトレース処理

なども行っている。

また，前述したがスケジューリングステートメントがコンパイルされると，そのオブジェクトは

全てランタイム，すなわちスケジューラに対するサブルーチンコールの形が作り出される。従って，スケジューラは，各スケジューリングステートメントの処理に対応した入口を持っている。これらのことから，スケジューラの処理内容は，

(i) スケジューリングステートメントの処理ルーチン

(ii) 積分の処理に関するルーチン

(iii) 次に処理するプロセスを決定するためのルーチン

(iv) フリーストレッジの管理に関するルーチン

などに大別できる。以下に，スケジューリング・メカニズム，タイミング・コントロールなどについて説明する。なお，各スケジューリングステートメントの処理については，後述3.4.6スケジューリングステートメントの項を参照されたい。

(1) スケジューリング・メカニズム

スケジューリング・メカニズムの基本的な機能は，イベントを時刻に従って順次実行させる事である。ＳＩＭＯＢＯＬ－Ⅱのスケジューリング・メカニズムはスケジューラによって行われる。スケジューラはこの機能を果すために，スケジュールテーブル，プロセス・コントロール・ブロック（以後ＰＣＢと略記する），イベントノーティス（以後ＥＮと略記する），およびスケジューリング用スイッチなどを参照する。

スケジュールテーブルには離散型プロセス用と連続型プロセス用の2種があるが，シミュレーション全体のコントロールに関する情報は，全て離散型プロセス用スケジュールテーブルに入れられている。図３－26 と図 3 － 27にその構造を示す。

SM.SCHTAB	先頭イベント・ノーティスのアドレス	最終イベント・ノーティスのアドレス
+1	チェイン・アイテム個数	
SM.TIME	現在時刻	
SM.SELF	現在実行中のプロセス	
+4	最終イベントのスケジュールタイム	
+5	発生イベント個数	
SM.STATE	状態表示	
SM.TSTOP	最終時刻	
SM.NSTOP	最終のイベント個数	
+9	メイン・プログラムの復帰番地	

SM.STATEの0～3ビットは次の状態の時ＯＮにセットされる。

0ビット：
時刻指定

1ビット：
イベント指定

2ビット：
離散型処理中

3ビット：
連続型処理中

図３－26　離散型プロセス用スケジュール・テーブル

SM.SCHTABC	先頭イベント・ノーティスのアドレス	最終イベント・ノーティスのアドレス
+1	チェン・アイテム個数	
SM.TIMEC	現在時刻	
SM.SELFC	現在実行中のプロセス	
SM.DT	最小ステップサイズ	
+5		
+6		
+7		
+8		
+9	メイン・プログラムの復帰番地	

図３－27　連続型プロセス用スケジュール・テーブル

このスケジュールテーブルにはイベントがスケジュールされると，対応して作られるイベントノーティスが，先行イベントノーティス・後続イベントノーティスの両方向へポインターを持つ対称リスト構造にチェインされている。

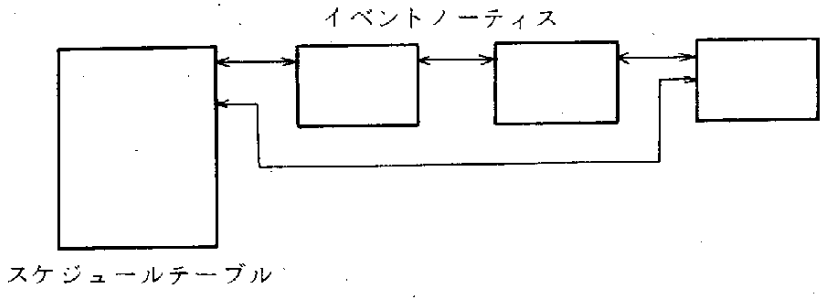


図３－28　スケジュールテーブルの概略図

離散型スケジュールテーブルにつながっているＥＮ（図３－29参照）には，３ワード目に実行予定時刻（以後ＥＴと略記する）が記入されており，このＥＴの順にソートされている。プライオリティは０～511までが許されているが，アクティビティ・プログラムで無指定の時は，そこから発生するプロセスのプライオリティは０とみなされる。

一方，連続用スケジュールテーブルにつながっているＥＮ（図3-29参照）の３ワード目にはスキャン・カウンター（Scan counter，ＳＣと略記する）の値が記入されており，こちらのENはプライオリティ順にソートされている。

0	キィー	PCBへのポインター
1	ポインター(先行EN)	ポインター(後続EN)
2	実行予定時刻（ＥＴ）	

（離散型ＥＮ）

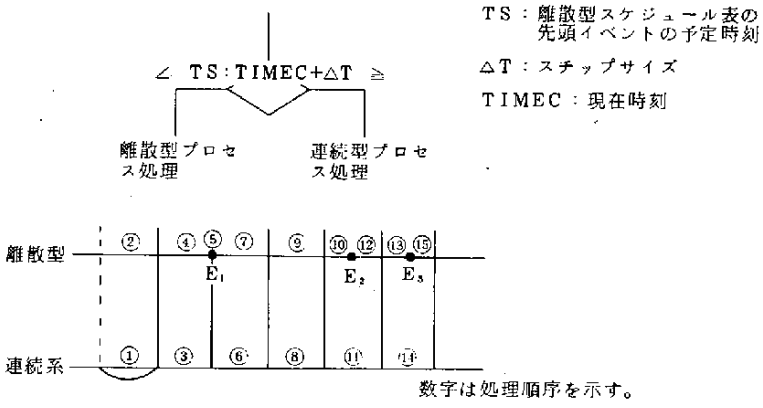
0	キィー	PCBへのポインター
1	ポインター(先行EN)	ポインター(後続EN)
2	Scan用 Connter（ＳＣ）	

（連続型ＥＮ）

図３－29　イベント・ノーティスの構造

ＳＣはタイミング・コントロール・ルーチンで連続型処理にコントロールが渡された時，つまりコントロールが現在時刻で処理すべき連続型プロセス発生させよとなった時に，どのプロセスを発生させるかをタイミング・コントロール・ルーチンに教えるという働きをする。即ち，連続型処理に入った時には，連続用スケジュールテーブルにつながっている全てのＥＮがスキャンされ，その時刻にＳＣ部が１であるイベントが発生させられる。要するに，ＳＣ部には何回に１回そのイベントを発生させるかという値がセットされている。最小ステップサイズ（連続型シミュレーション・クロックの進み幅）ごとにそのプロセスを発生させる場合には１をセットし，最小ステップサイズの３倍の時間ごとに発生させる場合には３をセットする。離散型シミュレーション・クロック（ＳＭ．ＴＩＭＥ），連続型シミュレーション・クロック（ＳＭ．ＴＩＭＥＣ）

離散型・連続型プロセスの混在における処理手順



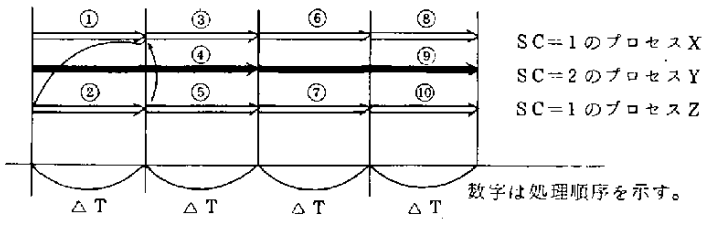


図３－30　連続型プロセスの処理手順

＋最少ステップサイズ（SM・DT）との比較により，連続型処理にコントロールが渡される度に「1」ずつＳＣ部が引かれてゆき，ＳＣ＝０の時にそのイベントが発生させられる。発生後は，そのプロセスのＰＣＢにセットされているＳＣ用の値（何回に１回そのプロセスを発生させるかの値）がコピーされる。また，スケジュリングステートメントの処理もこのＳＣ部に値をセットする事によって行う。

(2) タイミング・コントロール

例を挙げてタイミング・コントロールについて説明する。

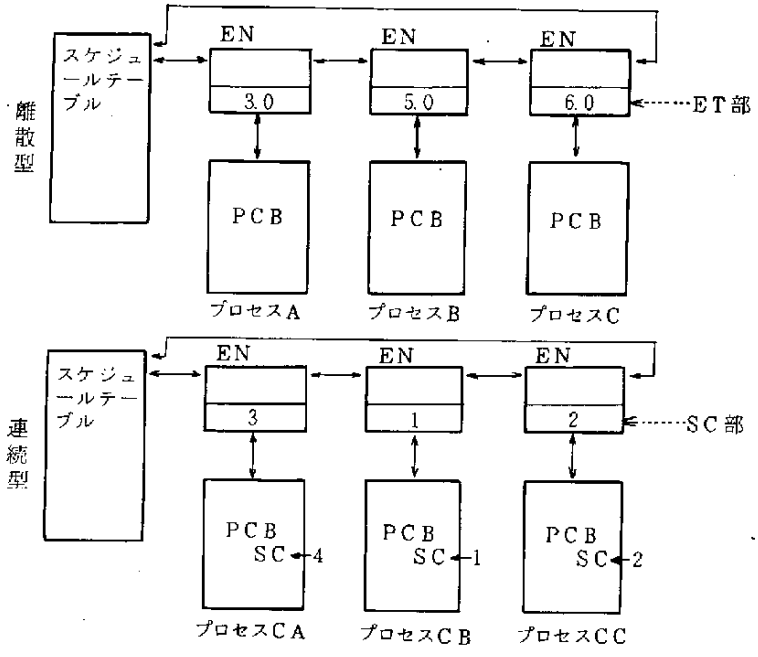


図３－31　スケジュール・テーブル

今，話を簡単にする為に次の事を仮定する。

1. 図3−31 のプロセスのプライオリティは全て「０」である。
2. 離散型のプロセスは，いづれも次のイベントの時刻において，他のプロセスへのアクセスがなく，ＤＥＬＡＹ 10.0 が記述されているものとする。
3. 連続型のプロセスは，全て他プロセスへのアクセスがなく，スケジューリング・ステートメントが記述されていないものとする。
4. 現在のシミュレーション・クロックは「2.0」であるとする。
（ＳＭ．ＴＩＭＥ＝ＳＭ．ＴＩＭＥＣ＝2.0）
5. 最小ステップサイズは「0.5」がセットされているものとする。（ＳＭ．ＤＴ＝0.5）

２，３の仮定は，図３－31 のいづれかのプロセスを処理したために図のＥＮの間に新たにＥＮ

がスケジュールされない事を仮定している。

おおまかな次のイベントの決定は図３－32の様に行う。

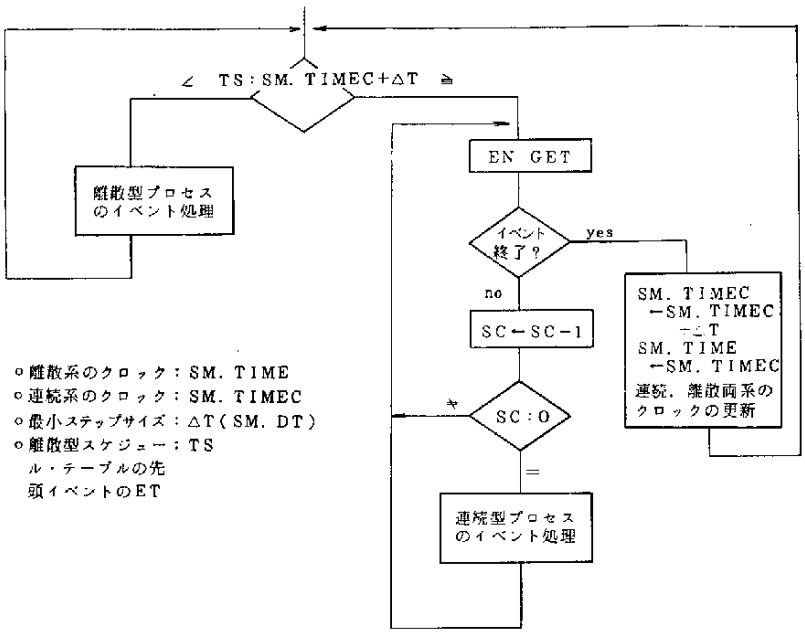


図３－32　イベントの決定

以上の条件が仮定された時に図３－31の様なイベントがスケジュールされていると，その処理は次の様になる。

Step 1.　TS＝3．0

SM．TIMEC＋△T＝2．5

連続型処理

(i)　プロセスCA　（SC←2）

(ii)　プロセスCB　イベントの発生（SC←1）PCBのSCコピー

(iii)　プロセスCC　（SC←1）

(iv)　SM．TIMEC　←SM．TIMEC＋△T＝2．5

SM．TIME　←SM．TIMEC＝2．5

Step 2.　TS＝3．0

SM．TIMEC＋△T＝3.0

連続型処理

(i)　プロセスCA　（SC←1）

(ii)　プロセスCB　イベントの発生（SC←1）

(iii)　プロセスCC　イベントの発生（SC←2）

(iv) ＳＭ．ＴＩＭＥＣ　←３．０

ＳＭ．ＴＩＭＥ　←３．０

Step 3. ＴＳ＝３．０

ＳＭ．ＴＩＭＥＣ＋△Ｔ＝３．５

離散型処理

(i) プロセスＡ　イベントの発生（ＥＴ←１３．０）

(ii) プロセスＢのイベントが先頭にくる。

Step 4. ＴＳ＝５．０

ＳＭ．ＴＩＭＥＣ＋△Ｔ＝３．５

連続型処理

(i) プロセスＣＡ　イベントの発生（ＳＣ←４）

(ii) プロセスＣＢ　イベントの発生（ＳＣ←１）

(iii) プロセスＣＣ　（ＳＣ←１）

(iv) ＳＭ．ＴＩＭＥＣ　←３．５

ＳＭ．ＴＩＭＥ　←３．５

Step 5. ＴＳ＝５．０

ＳＭ．ＴＩＭＥＣ＋△Ｔ＝４．０

連続型処理

(i) プロセスＣＡ　（ＳＣ←３）

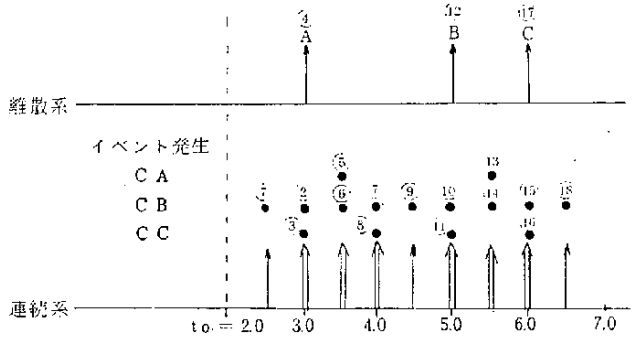
(ii) プロセスＣＢ　イベントの発生（ＳＣ←１）

(iii) プロセスＣＣ　イベントの発生（ＳＣ←２）

(iv) ＳＭ．ＴＩＭＥＣ　←４．０

ＳＭ．ＴＩＭＥ　←４．０

以下，同様に行なう。以上の事を図で表わすと次の様になる。



数字はイベントの発生順を示す。

図３－３３　イベントの発生

次に，ＳＩＭＢＯＬ－Ⅱでのタイミング・コントロールに関する考え方を列挙する。

① 基本的にＳＩＭＢＯＬ－Ⅱでは，連続型プロセスの数はそれほど多くはないと考えている。従って，連続型処理に移行した時に各イベントノーティスをスキャンする事は，単にイベントノーティスのポインターをたぐっていき，ＳＣ＝１のチェックをするだけなので，時間的lossはそれほどでないと考えている。

② プライオリティのレベルが512あること，また，同じレベルのものでもPriorの指定により優先順位をつけられること，さらに連続型プロセスの場合にはスキャンカウンターを用いているので，プロセスの順序を任意に設定する事が可能であることなどを考えると，ほとんど完璧に近い形で優先順位を表現する事が可能であると思われる。

③ システムのプロセス全体を考えた時に，同時刻に連続型プロセスと離散型プロセスの発生がスケジュールされている場合には，常に離散型プロセスを先に処理している。この事は，連続型プロセスの処理は時間が最小ステップサイズ分進められるが，離散型プロセスの処理は時間消費がないので，連続型プロセスを後に処理する必要があるからである。要するに，完全な同時はあり得ないと考えているわけである。離散型プロセスの処理を連続型の処理よりもほんのわずか遅くしたければ実行予定時刻がrealで表現されているので，その様な記述も可能である。

④ 計画変更，割り込みは，スケジューリングステートメントで巧みに表現できると考えている。

⑤ クロックの進め方は，連続型のクロックに離散型のクロックを追従させる方式をとっている。

⑥ ステップサイズの自動調整は，これを許すと他のプロセスとのずれを再調整しなければならないなど，処理時間がかかることになるので行っていない。その代り，ステップサイズの大きさを任意にとれること，および，変位の小さい連続型プロセスに対しては，最小ステップサイズの整数倍ごとに発生させることが可能である。

⑦ ＳＩＭＢＯＬ－Ⅱでは，ＮＥＷという生成オペレータによりＰＣＢが作成され，SCHEDULEステートメントによりＥＮの作成と登録を行う。このＥＮとＰＣＢをポインターによってつなぎ，ＥＮの発生，消去，スケジュールにより，あたかもプロセスの発生，消去，スケジュールを行っている。プロセスの実行過程を表現するには，スケジュール用のステートメントを単に記述するだけでなく，その命令が何度も実行されるように記述しなければならない。

ＳＩＭＢＯＬ－Ⅱでは"ＡＦＴＥＲ"や"ＢＥＦＯＲＥ"を用いて，プロセスのイベントに関連付けたスケジュールができ，あるプロセスのイベントと等しい時間属性を持つ様にスケジュールテーブルに登録できる。また，個々のプロセスに名前がつけられるので，スケジュールテーブル上に登録されている場合でも個別に呼び出す事ができ，時間属性に依らないスケジュールも簡単にできる。

⑧ 一度連続型処理に入ると，その時点で発生すべき全ての連続型プロセスを順々に処理してい

く。従って，ある連続型プロセスを処理している時に，ただちに離散型プロセスをスケジュールせよといった記述があった時でも，全ての連続型プロセスの処理を終えてから，次の処理に移る。そして，ただちに離散型プロセスをスケジュールせよといった時，該当するプロセスの実行予定時刻は，現在時刻＋△T（最小ステップサイズ）がセットされる。例えば，時刻TIMEで連続型処理中，連続型プロセスYの中で「離散型プロセスをただちにスケジュールせよ」といった記述があった時，連続型プロセスX，Y，Zの処理を終えてから離散型プロセスを処理する（図3－34参照の事）。

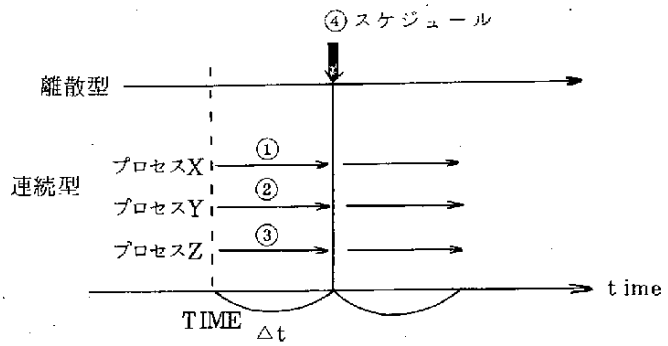


図3－34　連続処理中のスケジューリング

この離散型プロセスは，連続型プロセスYの処理を終えた時，即ち時刻がＴＩＭＥから△T進んだ時（ＴＩＭＥ＋△Ｔ）始めてスケジュールされる事になったわけであるので，実行予定時刻は，"ＴＩＭＥ＋△Ｔ"となる。ただし，システムクロックは，この連続型プロセスYを処理している時はまだＴＩＭＥのままであり，全ての連続型プロセスの処理が終った時点で"ＴＩＭＥ＋△Ｔ"に更新される。

⑨　連続型プロセスの処理中に，他の連続型プロセスへのスケジュールは，スケジュールテーブルにつながれている順序により異なる。例えば，次の連続型処理中に3番目と5番目に発生予定のプロセスがあった場合，3番目のプロセスの中で5番目のプロセスに対するリスケジュールの記述があれば，5番目のプロセスはその回には発生しない。逆に，5番目のプロセスの中で3番目のプロセスに対するリスケジュールの記述があれば，3番目，5番目両方のプロセスが発生する。

⑩　従来の離散型システムＳＩＭＢＯＬとは異なり，コンバインド・システムであるSIMBOL－Ⅱでは，任意の時点で離散型，連続型のスケジュールテーブルのどちらかに，1つ以上のプロセスがスケジュールされていればよい。つまり，ある時刻で離散型スケジュールテーブルへの登録がない事もありうる。

(3) プロセスのステータス

プロセスには，スケジューリングと関連してステータスが定義されている。離散型プロセスのステータスには，次の5つがある。

① アクティブ(Active)

現在実行中のプロセスで，システム全体を通じて一時点では唯一つのプロセスしかこの状態になり得ない。

② スケジュールド(Scheduled)

スケジュール・テーブルに登録されているプロセスで，実行予定時刻が，そのプロセスとリンクされているイベント・ノーティスに記入されている。

③ パッシブ(Passive)

キューに入って待ち状態にある時，もしくはいつでもスケジュール・テーブルに登録されうる状態にあるプロセスである。

④ ターミネィティド(Terminated)

再び実行されることのないプロセスである。他のプロセスが，このプロセスのローカルデータを参照しうるようにＰＣＢはシステムに存在している。

⑤ デッド(Dead)

プロセスがシステムから完全になくなった状態でＰＣＢもシステムの場から消滅している。

図3－35に離散型プロセスのステータスとスケジューリング・ステートメントの関係を示す。

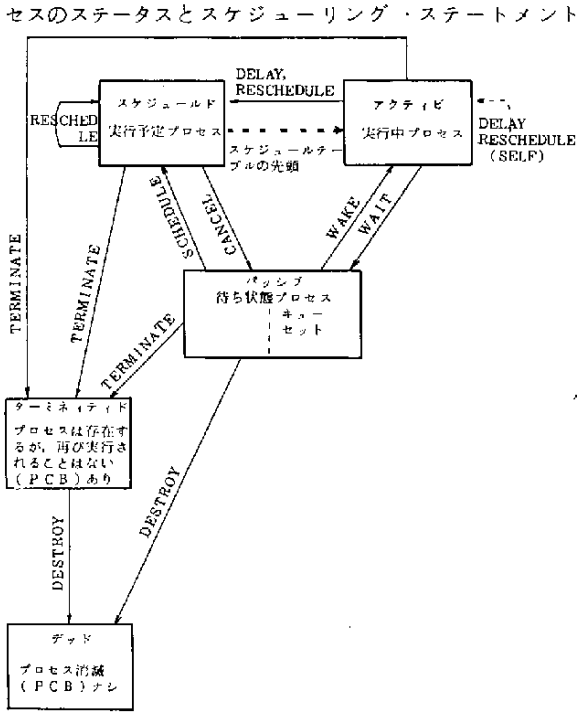


図3－35　離散型プロセスのステータスとスケジューリング・ステートメントの関係

連続型のプロセスのステータスは，図３－35のパッシブ状態がスケジュールド状態に吸収された形となる。連続型のプロセスは，一旦スケジュール・テーブルとチェインされるとＴＥＲＭＩＮＡＴＥ，ＤＥＳＴＲＯＹ以外のステートメントではチェインをはずされない。離散型プロセスのパッシブ状態に対応する状態は，連続型プロセスでは，ＳＣ部に大きな値（ＳＭ．ＴＳＴＯＰ＋１）をセットすることにより行う。従って，スケジュール・テーブルにはつながれたままであるので，特にパッシブ状態とスケジュールド状態を区別する必要はない。

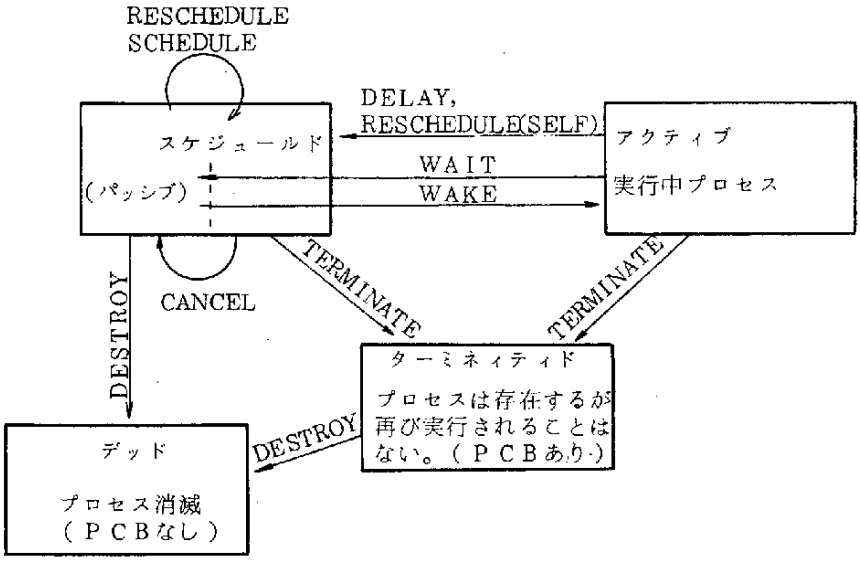


図３－36　連続型プロセスのステータスとスケジューリング・ステートメントの関係

3.4　主なモジュールの処理の説明

3.4.1　ディスパッチャーとステートメントプロセッサ

ディスパッチャーは次の処理を行なう。

①各種テーブルのイニシャライズ

②FCBのオープン

③スプリット・スクリーンの表示

④LOAD，SETB，GETDMの発信

⑤各コマンドルーチンへのジャンプ

⑥ステートメント処理ルーチンのコール

⑦リコンパイル処理ルーチンのコール

以下に，ディスパッチャーで使用するテーブルやエリア，記号の意味，処理の概略図および主なルーチンの仕様を示す。

SYMBOL TABLE STRUCTURE

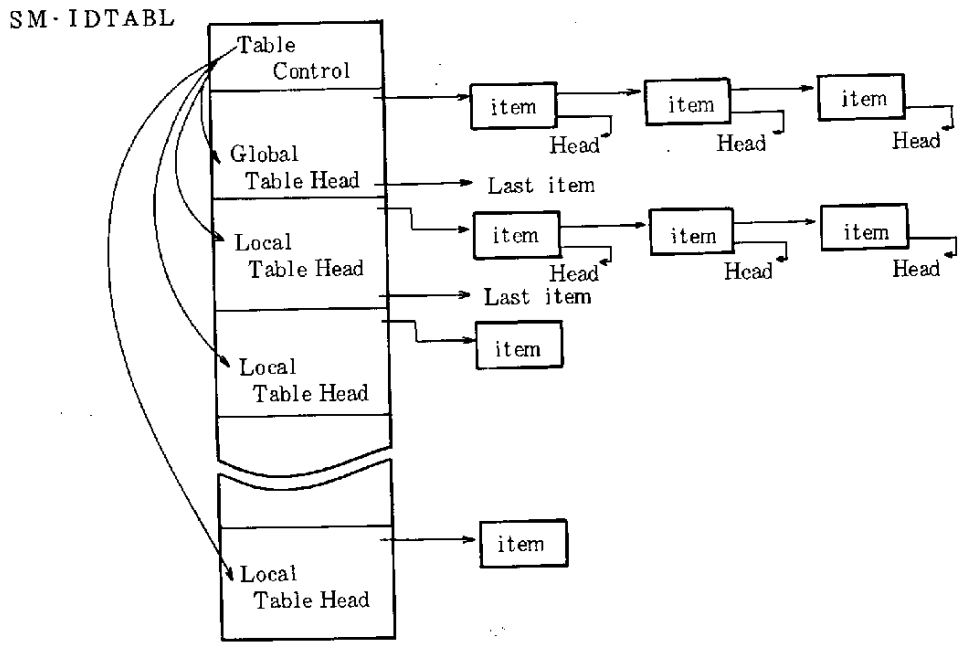


図3－37　シンボルテーブル1

SYMBOL TABLE ITEM の Format

（GLOBAL, LOCALとも同形式）

+0	Table Head へのポインター				後続 item へのポインター
+1	Attribute 1				Location
+2	1 dim				2 dim
+3	PC	attr.2	Base	0	Tōtal size
+4	Identifier (SYMBOL)				
+5					

図3－38　シンボル・テーブル2

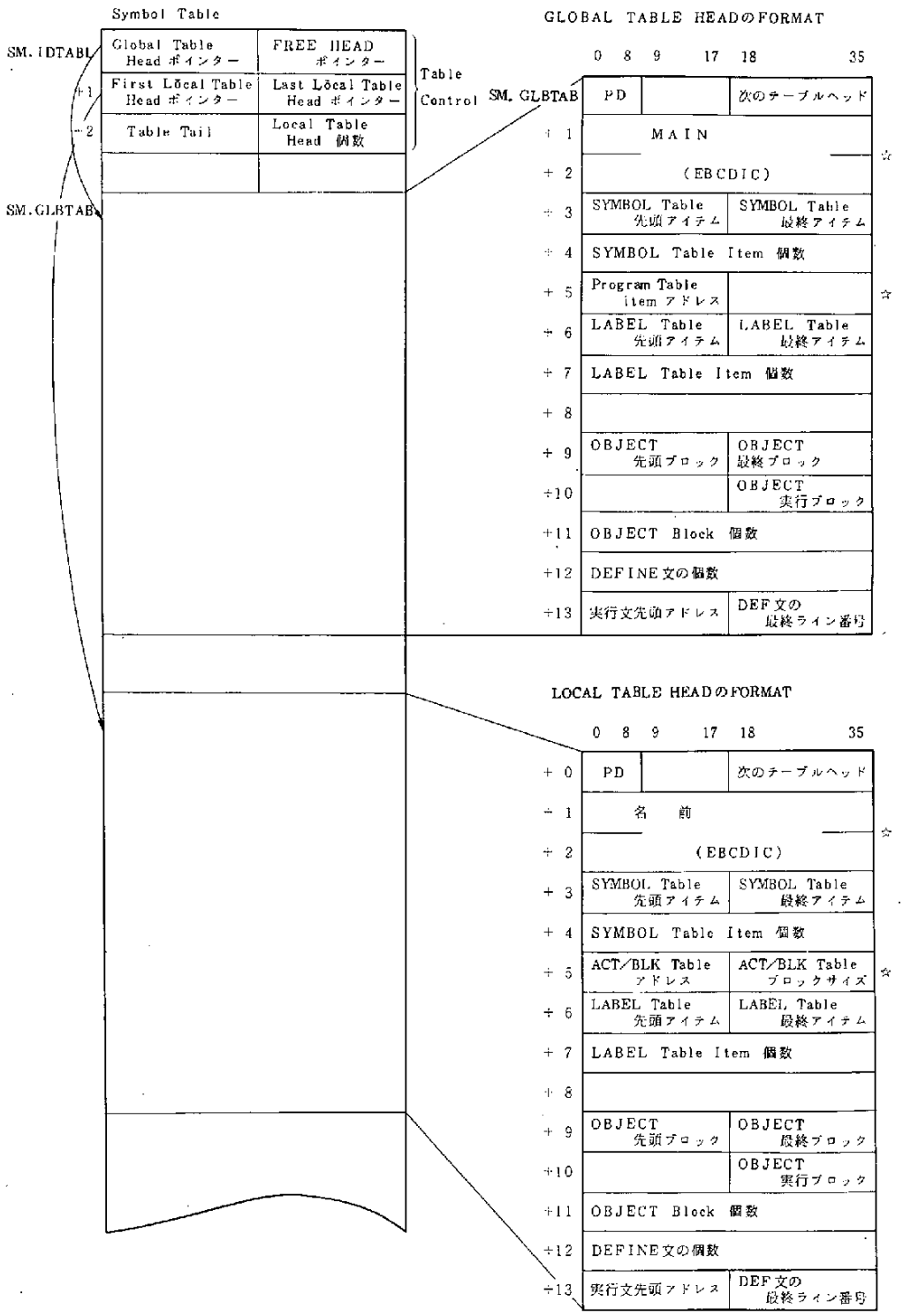
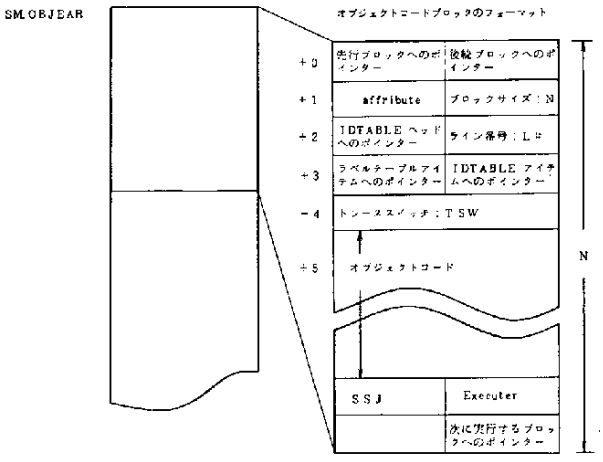


図3－39　シンボルテーブル3

・オブジェクトコードの入っているエリア（ 1024 × 3 語 ）



・残りサイズを示すカウンタ（ 1語 ）

SM.OBJREM

・ロケーションカウンタ（ 1語 ）

SM.LCC

図３－40　オブジェクト　プログラム　エリア

表３－３　プログラムの種類

PD(Binary)	Rrogram Kind
1	Main Program
2	Activity Program
3	
4	Procedure
5	Real Procedure
6	Integer Procedure
7	Logic Procedure
8	Process Procedure

(注)　PDはProgram Descriptor の略

・ディスパッチャー処理の概略図

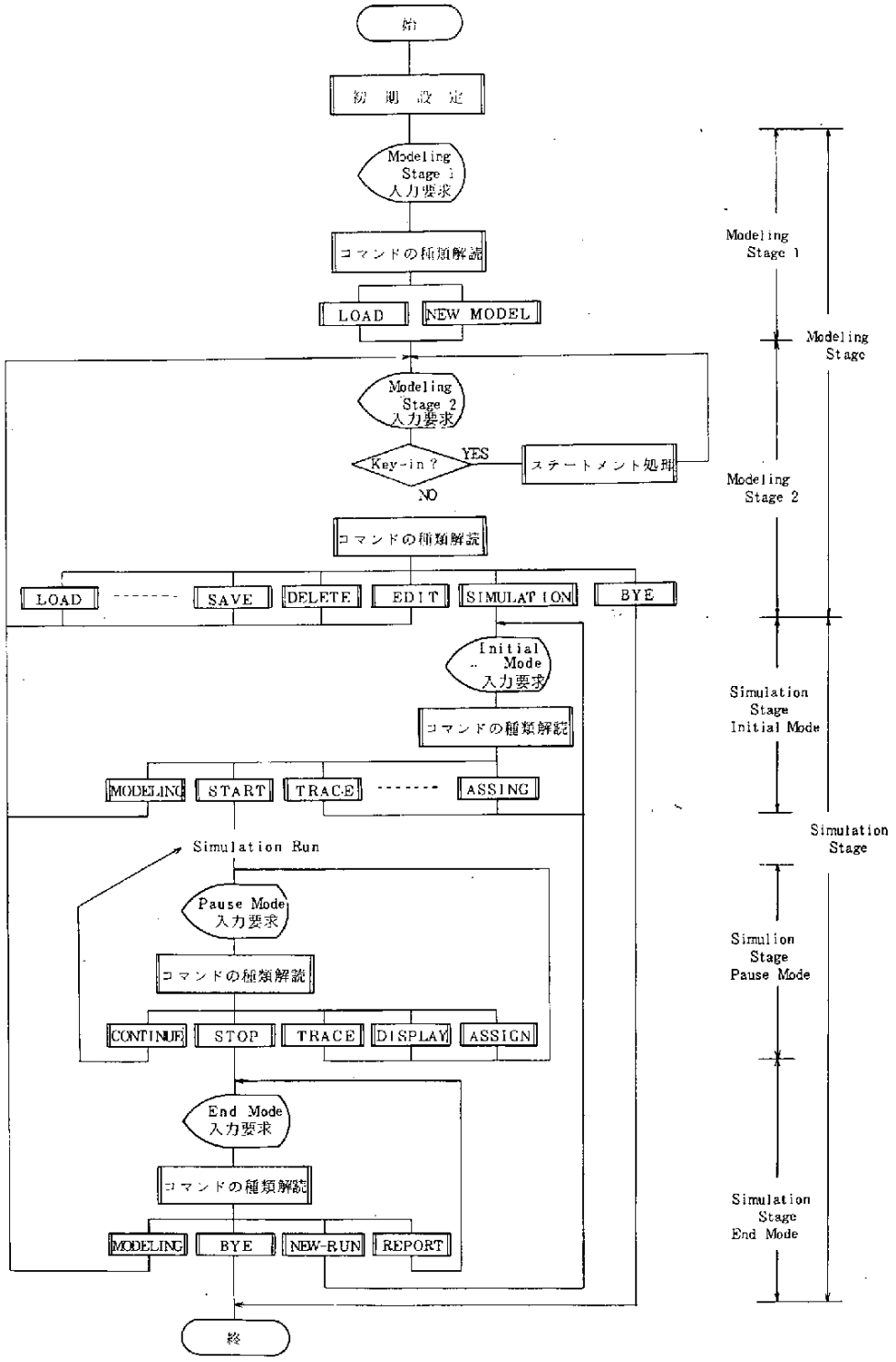
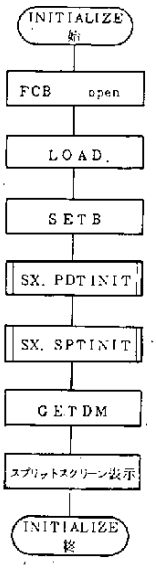


図3－41　ディスパッチャー処理の概略図

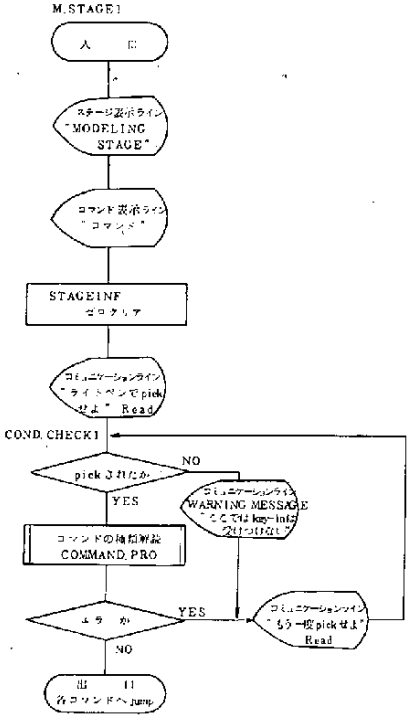
・ディスパッチャー処理

初期設定部分



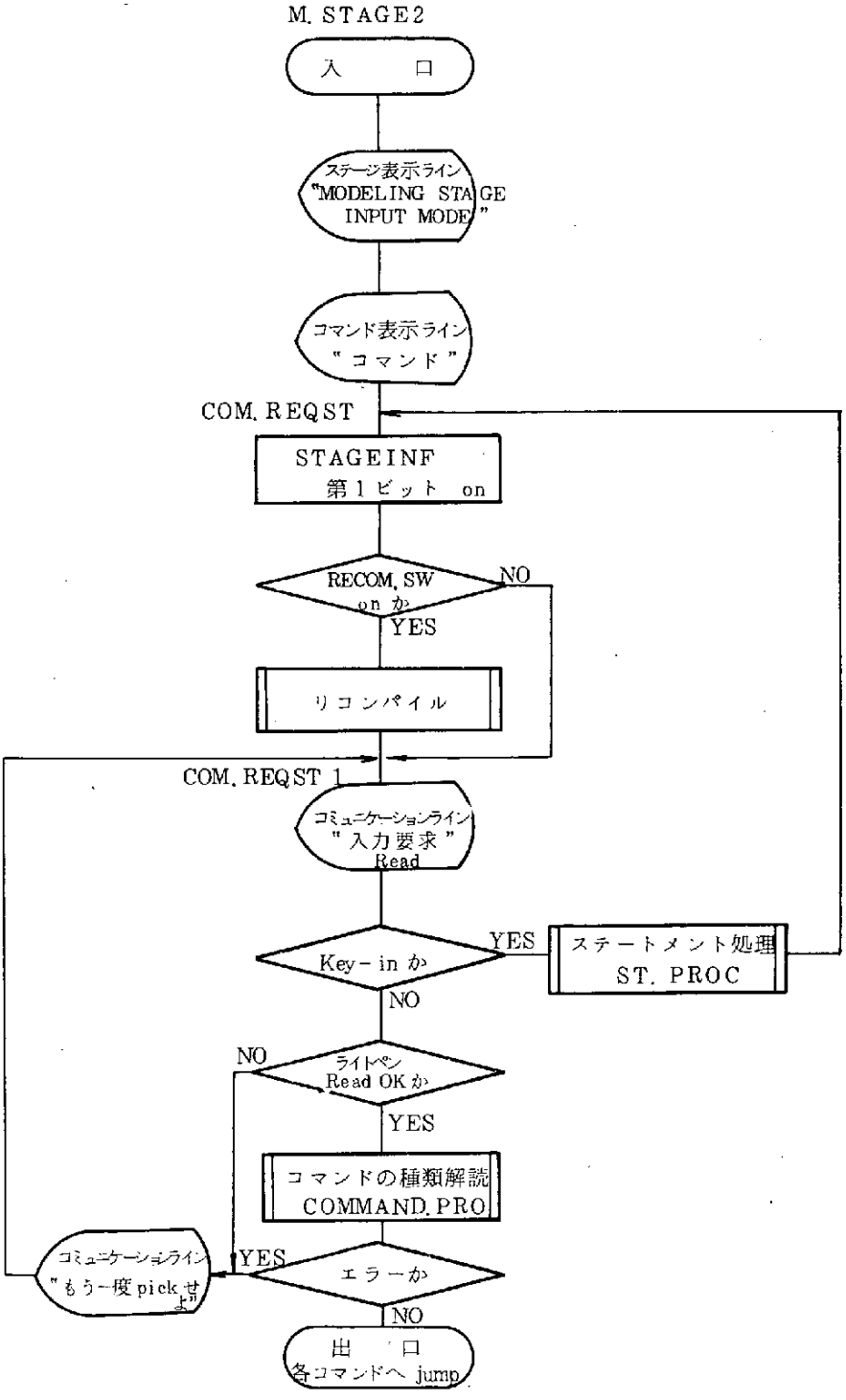
終了後，Modeling Stage 1の処理へ続く。

図3－42　ディスパッチャー処理の初期設定部分のフローチャート



※各コマンドは処理終了後，ステージ情報（STAGEINF）により戻り先を選択する。

図3－43　ディスパッチャー処理モデリングステージのフローチャート1



※各コマンドは処理終了後，ステージ情報（ STAGEINF ）により戻り先を選択する。

図3－44　ディスパッチャー処理モデリングステージのフローチャート2

ST.PROC ルーチン

1. 機　能

ステートメントの処理を行なう。

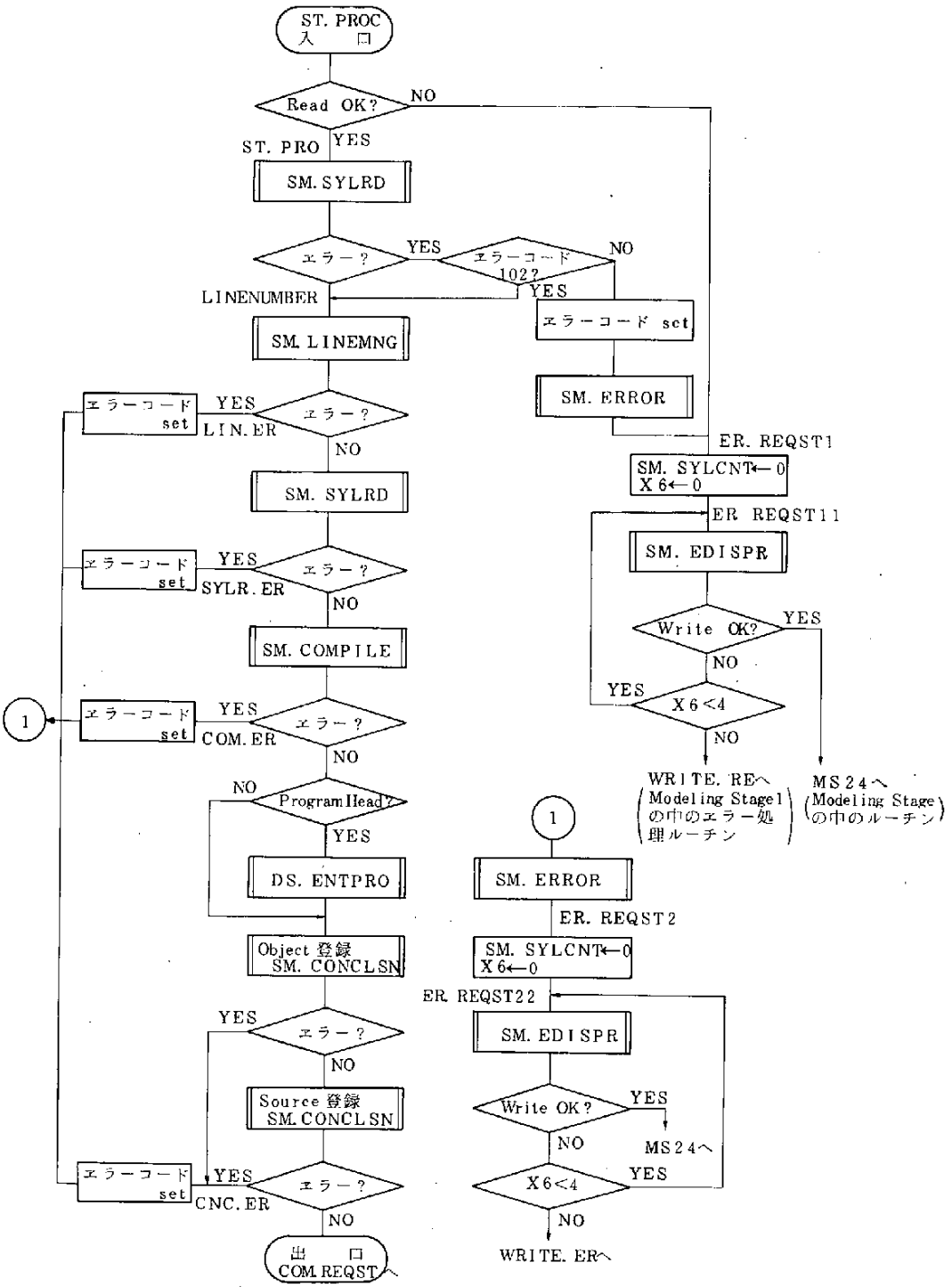
2. 処理内容

・ラインナンバーによるソースステートメント処理ルーチンのコール

・コンパイル処理ルーチンのコール

・登録処理ルーチンのコール

3. ブロックチャート


ST. PROC
入　口
Read OK?
NO
YES
ST. PRO
SM. SYLRD
エラー?
YES
エラーコード
102?
NO
YES
LINENUMBER
エラーコード set
SM. LINEMNG
SM. ERROR
エラーコード
set
YES
エラー?
LIN. ER
NO
ER. REQST1
SM. SYLCNT← 0
X 6← 0
SM. SYLRD
ER REQST11
SM. EDISPR
エラーコード
set
YES
エラー?
SYLR. ER
NO
Write OK?
YES
NO
YES
X 6 < 4
NO
SM. COMPILE
1
エラーコード
set
YES
エラー?
COM. ER
NO
WRITE. REへ
Modeling Stage1
の中のエラー処
理ルーチン
MS 24へ
(Modeling Stage
の中のルーチン)
NO
Program Head?
YES
1
DS. ENTPRO
SM. ERROR
ER. REQST2
Object 登録
SM. CONCLSN
SM. SYLCNT←0
X 6←0
ER. REQST22
YES
エラー?
SM. EDISPR
NO
Source 登録
SM. CONCLSN
Write OK?
YES
NO
MS 24へ
エラーコード
set
YES
エラー?
CNC. ER
X 6 < 4
YES
NO
NO
出　口
COM. REQST
WRITE. ERへ

SM.LINEMNGルーチン

1. 機　能

ラインナンバーによるソースステートメントの処理

2. 処理内容

次のことを行なう。

・ラインのサーチ

・ラインのデリート

・ラインナンバーのチェック

ソースプログラムテーブルのサーチ結果 / 処理の種類（入力形式）	L＃があった場合	L＃がなかった場合
サーチ（L＃　SEND　）	そのステートメントを表示	エラー
デリート（L＃　；）	そのステートメントの先頭に＊印をつけて確認	エラー
チェック（L＃　ステートメント）	リプレイスとみなす	正　常
そ　の　他	エラー	エラー

① ラインのサーチ

SM.SPT(Source Program Table）内をサーチして，もし，同じL＃のラインがあればそのラインをキーインエリアに表示する。

同じL＃のラインがなければ，エラーとしてその旨を表示する。

② ラインのデリート

SM.SPT内をサーチして，もし，同じL＃のラインがあれば，そのラインの先頭に＊をつけ，ユーザの判断を求める。しかしそのラインが，画面表示外の場合にはその旨を表示する。ラインを表示する場合，リコンパイルを必要とするケースがあるが，その場合はSM.OBJEAR(Object Area）内をサーチして，リコンパイルするか否かを判断し，リコンパイルが必要な場合は復帰情報にその情報をセットしておき，上部ルーチンがリコンパイルルーチンを呼ぶ。

もし，同じL＃のラインがなければ，エラーとしてその旨を表示する。

③ ラインナンバーのチェック

L＃の次にステートメントが続いている場合，同じL＃がなければ正常とし，L＃を除いたステートメント本体をSM.INTST（シラブル・リードの対象エリア）に入れる。

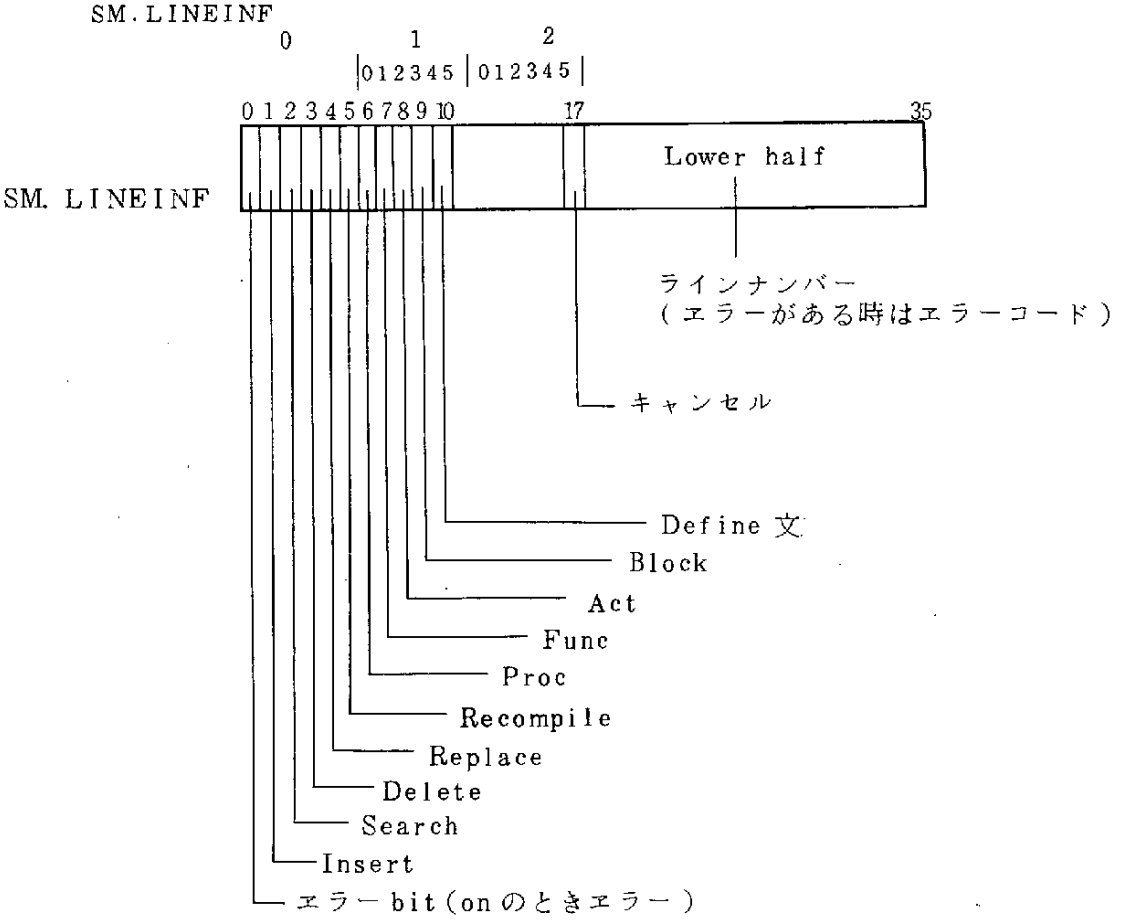
もし，同じL＃のラインがあればリプレイスとみなし，上記と同様の処理を行なう。

3. 呼び出し手順

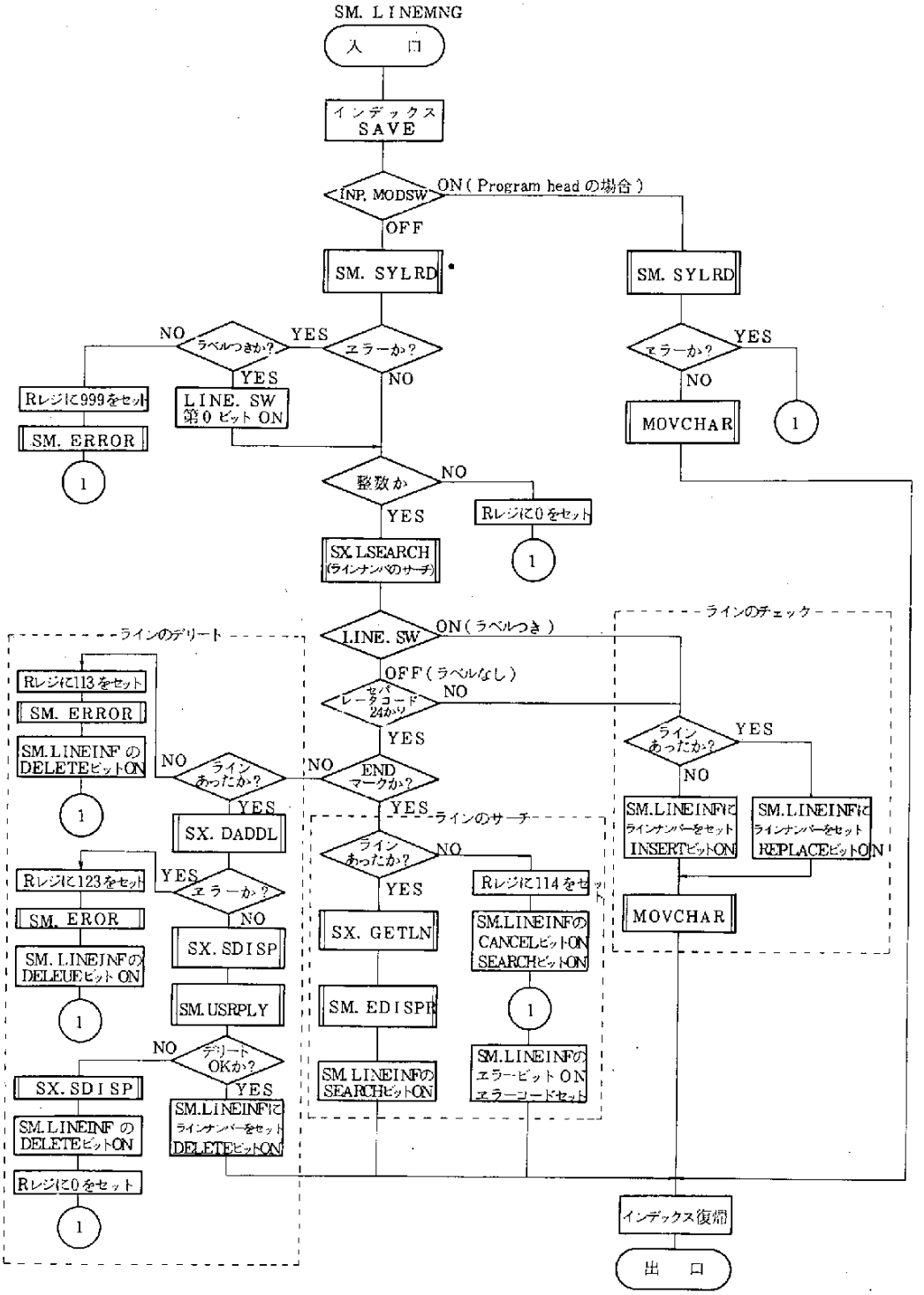
```
SXJ, 7    SM. LINEMNG
```

4. 復帰情報

SM.LINEINF



5. ブロックチャート


SM. LINEMNG
入　口
インデックス SAVE
INP. MODSW
ON (Program head の場合)
OFF
SM. SYLRD
SM. SYLRD
ラベルつきか?
NO
YES
エラーか?
YES
NO
エラーか?
YES
NO
Rレジに999をセット
SM. ERROR
1
LINE. SW 第0ビット ON
1
MOVCHAR
1
整数か
NO
YES
Rレジに0をセット
1
SX. LSEARCH (ラインナンバのサーチ)
LINE. SW
ON (ラベルつき)
OFF (ラベルなし)
セパレータコード 24かり
NO
YES
ラインのデリート
ラインのチェック
ラインのサーチ
Rレジに113をセット
SM. ERROR
SM. LINEINF の DELETEビットON
1
ライン あったか?
NO
YES
END マークか?
NO
YES
ライン あったか?
YES
NO
SM. LINEINFに ラインナンバーをセット INSERTビットON
SM. LINEINFに ラインナンバーをセット REPLACEビットON
MOVCHAR
SX. DADDL
Rレジに123をセット
SM. EROR
SM. LINEINF の DELEUEビット ON
1
エラーか?
YES
NO
SX. SDISP
SM. USRPLY
デリート OKか?
NO
YES
SX. SDISP
SM. LINEINF の DELETEビットON
Rレジに0をセット
1
SM. LINEINFに ラインナンバーをセット DELETEビットON
SX. GETLN
SM. EDISPR
SM. LINEINFの SEARCHビットON
Rレジに114をセット
SM. LINEINFの CANCELビットON SEARCHビットON
1
SM. LINEINFの エラービットON エラーコードセット
インデックス復帰
出　口

SM.COMPILEルーチン

1. 機　能

   コンパイル処理

2. 処理内容

   ・キーワードの種類を識別して，そのラインナンバーが正しい位置にあるかどうかを調べる。

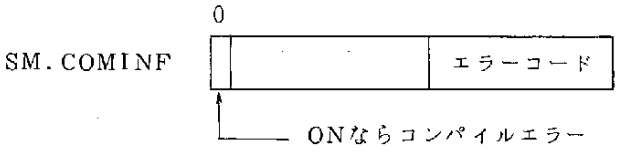
   （例えば，実行文がDefine文の前にきていないかというようなこと）

   ・各ステートメント処理ルーチン（パス1，パス2），SM. IDOPEN, SM. IDCLOSE, SM. PRDOPEN, SM. PRDCLOSE等を呼び出す。

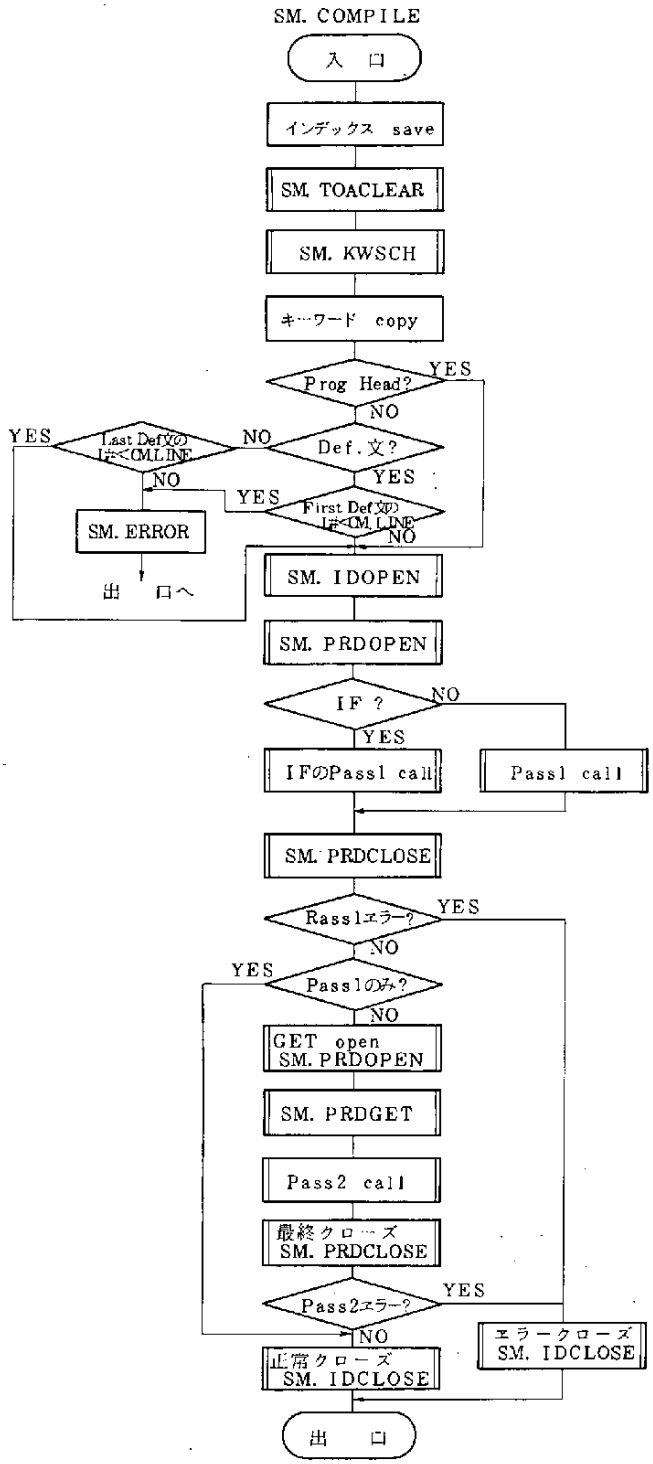
3. 呼び出し手順

```
SXJ，7        SM.COMPILE
```

4. 復帰情報



5. ブロックチャート


SM. COMPILE
入 口
インデックス save
SM. TOACLEAR
SM. KWSCH
キーワード copy
Prog Head?
YES
NO
Def. 文?
NO
YES
Last Def文の上<CM LINE
YES
NO
First Def文の上<CM LINE
YES
NO
SM. ERROR
出 口へ
SM. IDOPEN
SM. PRDOPEN
IF ?
NO
YES
IFのPass1 call
Pass1 call
SM. PRDCLOSE
Rass1エラー?
YES
NO
Pass1のみ?
YES
NO
GET open
SM. PRDOPEN
SM. PRDGET
Pass2 call
最終クローズ
SM. PRDCLOSE
Pass2エラー?
YES
NO
エラークローズ
SM. IDCLOSE
正常クローズ
SM. IDCLOSE
出 口

SM.CONCLSNルーチン

1. 機　能

登録処理

2. 処理内容

・ソースの登録

・ソースの削除

・オブジェクトの登録

・オブジェクトの削除

・SM.IDTABLEへの書き込み　等を行なう。

以下に，オブジェクトの登録・削除に伴なう各種の書き込み，変更等について示す。

オブジェクト登録処理

		プログラムヘッド	インサート	追　加
IDTABLE	オブジェクト先頭番地	新しいオブジェクト番地書き込み		
	オブジェクト最終番地	新しいオブジェクト番地書き込み		新しいオブジェクト番地書き込み
	ブロック個数	値　更新	値　更新	値　更新
	Define 個数	ゼロ	値・更新（条件によって）	値　更新（条件によって）
	実行入口番地		新しいオブジェクト番地書き込み(条件によって)	新しいオブジェクト番地書き込み(条件によって)
先行ブロック	先行ブロック番地			
	後続ブロック番地		新しいオブジェクト番地書き込み	新しいオブジェクト番地書き込み
新ブロック	先行ブロック番地	ゼロ	先行オブジェクト番地書き込み	先行オブジェクト番地書き込み
	後続ブロック番地	ゼロ	後続オブジェクト番地書き込み	ゼロ
後続ブロック	先行ブロック番地		新しいオブジェクト番地書き込み	
	後続ブロック番地			
LCC(オブジェクトエリアのロケーションカウンタ)		新しいオブジェクト番地＋アイテムの大きさ	新しいオブジェクト番地＋アイテムの大きさ	新しいオブジェクト番地＋アイテムの大きさ

オブジェクト削除処理

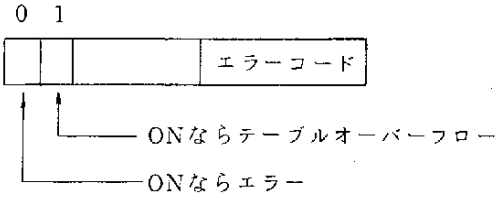
		先行，後続あり	後続なし
IDTABLE	オブジェクト先頭番地		
	オブジェクト最終番地		先行オブジェクト番地書き込み
	ブロック個数	値　更新	値　更新
	Defineｃ　個数	値　更新（条件によって）	値　更新（条件によって）
	実行入口番地	書き込み（条件によって）	書き込み（条件によって）
先行ブロック	先行ブロック番地		
	後続ブロック番地	後続オブジェクト番地書き込み	ゼロ
後続ブロック	先行ブロック番地	先行オブジェクト番地書き込み	
	後続ブロック番地		
LCC（オブジェクトエリアのロケーションカウンタ）			先行オブジェクト番地＋アイテムの大きさ

3. 呼び出し手順

```
SXJ，7          SM.CONCLSN
```

4. 復帰情報

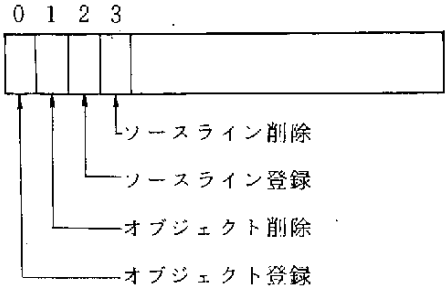
SM.CNCEXT



5. 備　考

① このルーチンを呼ぶ前に，次の情報をセットしておかなければならない。

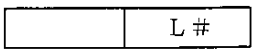
SM.CNCINF



(a) ソース登録，削除の場合

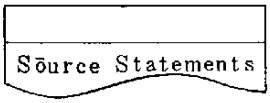
ラインナンバー

SM.LINEINF



(注) SM. LINEINFはSM. LINEMNG（L#処理）ルーチンを呼んだときの復帰情報ソースステートメントの入っているエリア。

SM.INTST



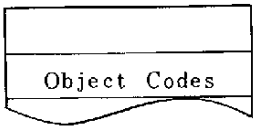
(b) オブジェクト登録，削除の場合

ラインナンバー

SM.LINEINF

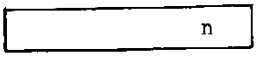
オブジェクトコードの入っているエリア

SM.TOA

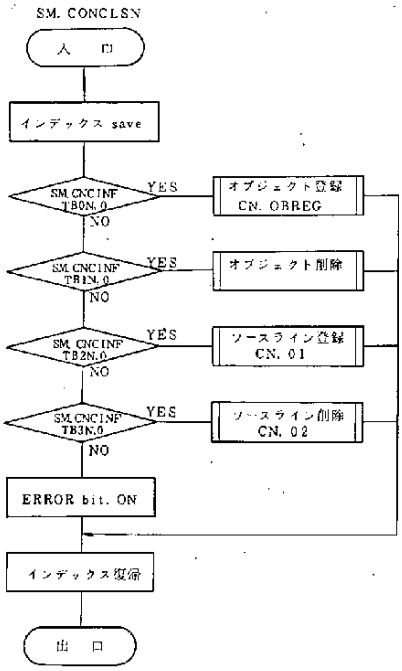


SM. TŌA内のオブジェクトのサイズ

SM. TŌACRP



6. ブロックチャート



SM.RECOMPL ルーチン

1. 機　能

リコンパイル処理を行なう。

処理概要

ソースラインのデリート，リプレース等が行なわれた場合，リコンパイルしなければならない事が起こる。この様な場合ソース・プログラム・テーブル（SPT）より1ラインずつ取り出して（SM.GTONESTを使用），コンパイルを行なう。

ラインナンバーはSPT内にステートメントと共に管理されているが，SM.GTONEST を呼んだときにSM.LINEINFの下半語にセットされる。この情報は，オブジェクト・コードを登録する場合用いられる。

コンパイル・エラーが発生した場合の処理法

・リコンパイルの最中にコンパイル・エラーが発生した時そのエラーメッセージを表示し，ユーザにインプットを要求する。

・ラインランバー処理ルーチン（SM.LINEMNG）を呼んだ時の復帰情報により次 の処理を行なう。

① ラインナンバー・エラーの場合

エラーメッセージを表示し，ユーザの再入力を要求する。

② サーチの場合

ユーザの再入力を要求する。

③ デリートの場合

SPT中のソースラインを削除し，オブジェクト・ブロックをサーチして，リコンパイルしなければならない場合，再びリコンパイルし直す。

リコンパイル不要の場合は，次のラインを取り出して，コンパイルを続ける。

④ リプレースの場合

オブジェクト・ブロックをサーチして，再びリコンパイルしなければならないかどうかを調べる。

リコンパイルの場合は，SPT中のソースラインを入替え，リコンパイルし直す。

リコンパイル不要の場合は，ソースを入替え，コンパイルを続ける。

⑤ インサートの場合

再びリコンパイルしなければならないかどうかを調べる。

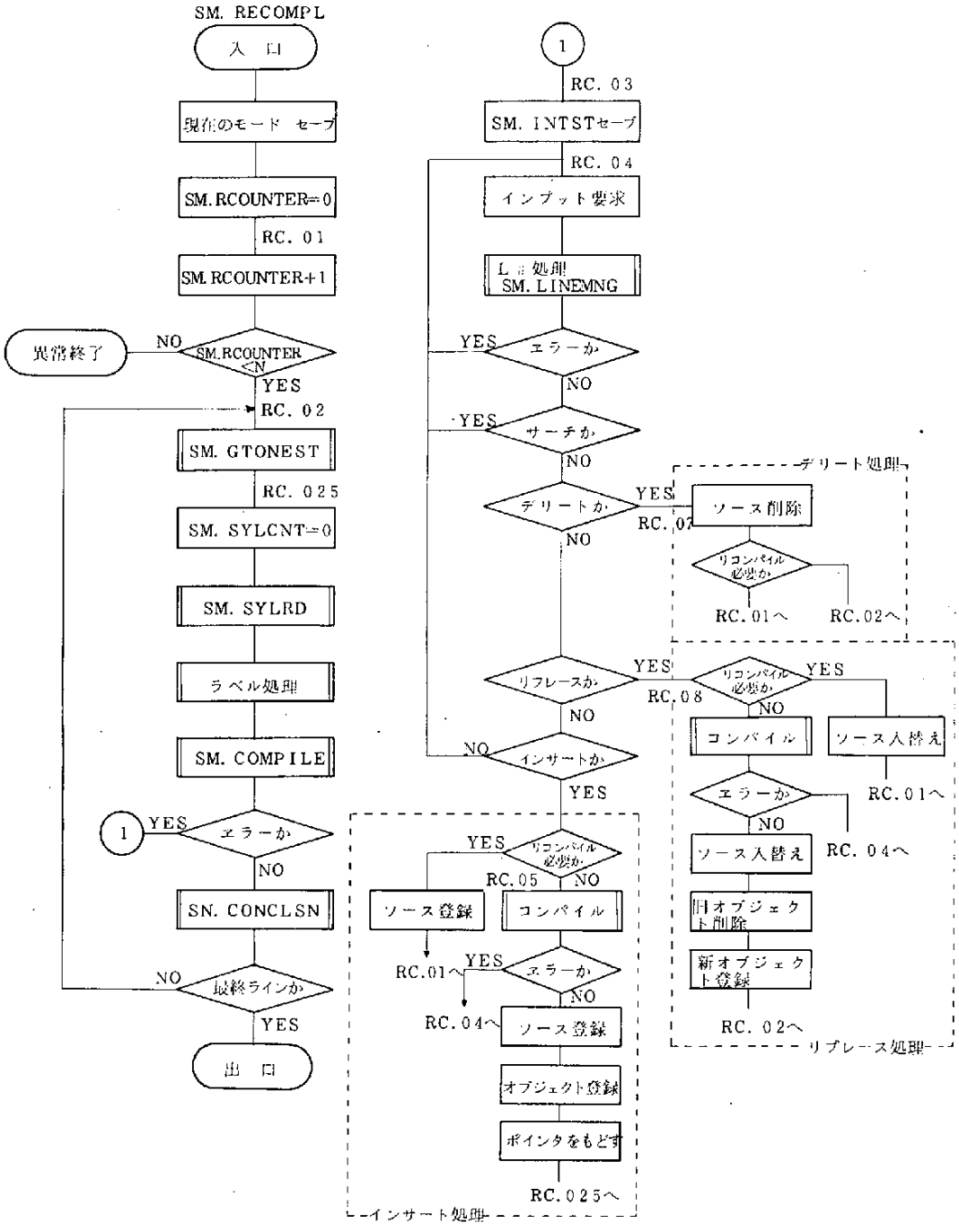
リコンパイルの場合は，SPTにソースラインを登録し，リコンパイルし直す。

リコンパイル不要の場合は，ソースとオブジェクトを登録し，ラインナンバーをリセットして，コンパイルを続ける。

2. 呼び出し手順

```
J                SM.RECOMPL
```

3. ブロックチャート


SM. RECOMPL
入　口
現在のモード　セーブ
SM. RCOUNTER=0
RC. 01
SM. RCOUNTER+1
異常終了
NO
SM. RCOUNTER <N
YES
RC. 02
SM. GTONEST
RC. 025
SM. SYLCNT=0
SM. SYLRD
ラベル処理
SM. COMPILE
1
YES
エラーか
NO
SN. CONCLSN
NO
最終ラインか
YES
出　口
1
RC. 03
SM. INTSTセーブ
RC. 04
インプット要求
L：処理
SM. LINEMNG
YES
エラーか
NO
YES
サーチか
NO
デリートか
YES
RC. 07
ソース削除
リコンパイル必要か
RC. 01へ
RC. 02へ
デリート処理
NO
リプレースか
YES
RC. 08
リコンパイル必要か
YES
NO
コンパイル
ソース入替え
エラーか
RC. 01へ
NO
ソース入替え
RC. 04へ
旧オブジェクト削除
新オブジェクト登録
RC. 02へ
リプレース処理
NO
インサートか
YES
YES
リコンパイル必要か
RC. 05
NO
ソース登録
コンパイル
RC. 01へ
YES
エラーか
NO
RC. 04へ
ソース登録
オブジェクト登録
ポインタをもどす
RC. 025へ
インサート処理

3.4.2　ディスプレイ表示処理サブルーチン

ここで述べるルーチン群は，CRTキャラクタディスプレイ装置を媒体として，実行結果，プログラムリスト，オペレーション要求などを出力する。また，ラインプリンタを媒体として出力情報のハードコピーをとる。一方，SIMBOL－Ⅱプロセッサに対し，キーボードおよびライトペンを媒体としてプログラムやデータなどを入力する。

CRTキャラクタディスプレイ装置の画面は24行×80列であるが，SIMBOL－Ⅱでは第1行から第22行までにプロテストをかけ，ユーザの書き込みを禁止している。次にディスプレイ画面の使用法を示す。

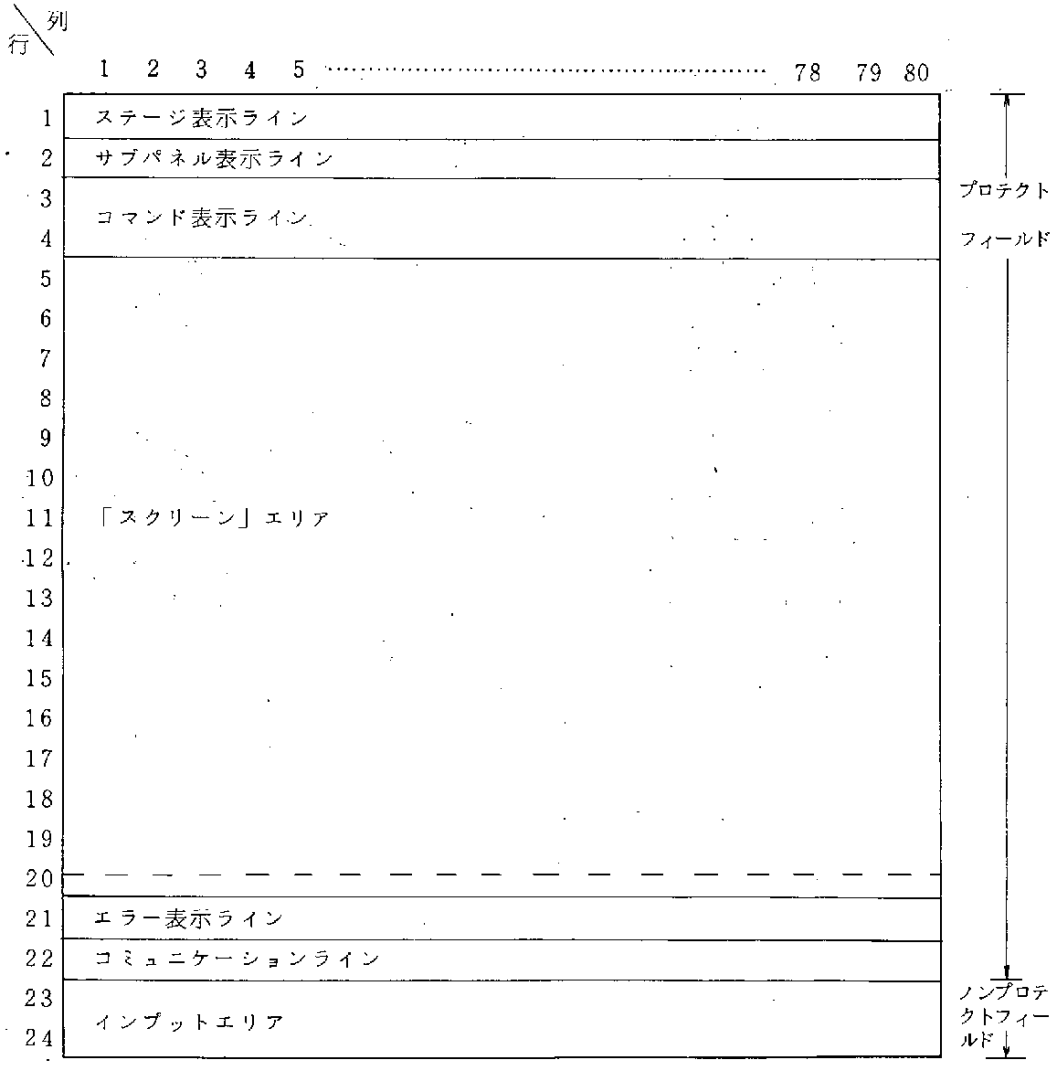


図3－45　CRTキャラクタディスプレイ画面の使用法

SM.PROTECTルーチン

1. 機　能

・CRT画面初期設定ルーチン

・SIMBOL－Ⅱ初期設定時に呼ばれる。

・CRT画面の第1行～第22行をユーザ書き込み禁止領域とするためプロテクトをかける。

・CRT画面の第20行にスプリット・ラインを表示する。

・I／Oエラーが生じた時には，指定ワードの0bit目をONにして返す。

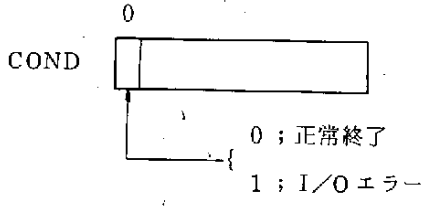
2. 呼び出し手順

```
SXJ, 7          SM.PROTECT
NOP             COND, , base
```

・COND; I／Oエラー通知復帰情報

I／Oエラーの時，0bit目をONにして返す。

呼び出し側でclearする必要はない。



SM.TITLEルーチン

1. 機　能

・CRT画面のラインプリンタ・ハードコピーのための表題をラインプリンタ上に表示するルーチン。

・SIMBOL－Ⅱ初期設定時に呼ばれる。

2. 呼び出し手順

```
SXJ, 7          SM.TITLE
```

SM.INMODEルーチン

1. 機　能

・ステージ表示ルーチン

・CRT画面のステージ表示ライン（第1行）に80文字以内の文字列を表示する。

・I／Oエラー時には，指定ワードの0bit目をONにして返す。

2. 呼び出し手順

```
SXJ, 7          SM.INMODE
NOP             STRING, , base
NOP             COND, , base
```

・STRING；表示文字数（ｎ）および表示文字列が格納されているエリアの先頭番地。

STRING

n			
A	B	C	D
E	F	G	H

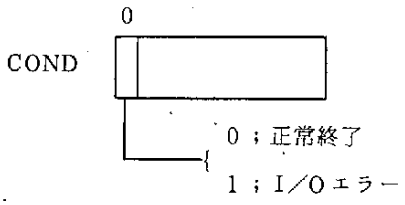
ただし

$1 \leqq n \leqq 80$

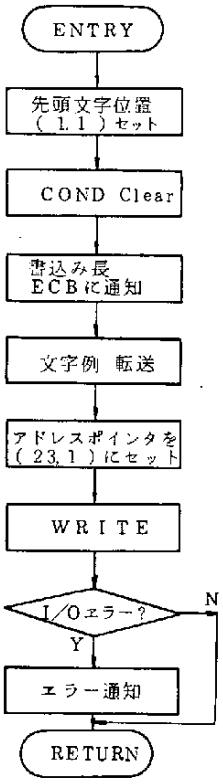
・COND；　I／Oエラー通知復帰情報

I／Oエラーの時，0bit目をONにして返す。

呼び出し側でclearする必要はない。



3. ブロックチャート



SM. SPANELルーチン

1. 機　能

・サブパネル表示ライン (第2行) に，80文字以内の文字列を表示する。

・I/Oエラーの時には，復帰情報エリアの0bit目をONにして返す。

2. 呼び出し手順

```
SXJ，7          SM.SPANEL
NOP             STRING，，base
NOP             COND，，base
```

STRING，CONDの意味は，SM.INMODEに同じ。

SM.COMNDルーチン

1. 機　能

・コマンド表示エリア (第3行～第4行) に，160文字以内の文字列を表示する。

・I/Oエラーの時には，復帰情報エリアの0bit目をONにして返す。

2. 呼び出し手順

```
SXJ, 7          SM.COMND
NOP             STRING, , base
NOP             COND, , base
```

・STRING；表示文字数（ｎ）および表示文字列が格納されている領域の先頭番地

STRING

n			
A	B	C	D
E	F	G	H

ただし

$1 \leqq n \leqq 160$

・COND； SM.INMODEに同じ。

SM.ERRORルーチン

1. 機　能

・エラーコード（３ケタの整数）をエラー表示ライン（第21行）に表示する。

・エラーコードをＲレジスタに格納してからこのルーチンを呼ばなければならない。

・Ｉ／Ｏエラー時には指定ワードの0bit目をONにして返す。

2. 呼び出し手順

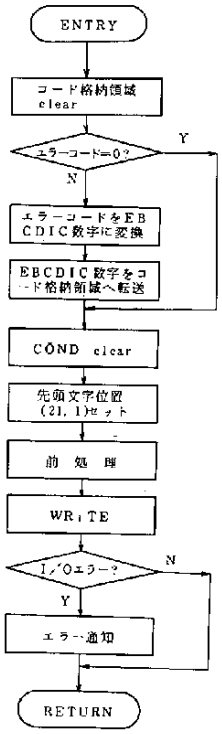
```
LRI             N (ERROR CODE)◎
SXJ, 7          SM.ERROR
NOP             COND, , base
```

・COND； Ｉ／Ｏエラー通知復帰情報

Ｉ／Ｏエラーの時，0bit目をONにして返す。

◎Ｒレジスタにエラーコードが格納されればよいのであって，必ずしも `LRI N` という形式をとる必要はない。

3. ブロックチャート



SM.RDISPLYルーチン

1. 機　能

・「スクリーン」上の（Y，X）座標を与えて，文字列をCRTのスクリーンに表示する。

・もし，文字列がスクリーンをはみ出す場合には，スクリーンに書き込める文字のみ表示し，復期情報エリアの文字オーバー表示bitをONにして返す。

・I／Oエラーの時，復帰情報エリアの0bit目をONにして返す。

2. 呼び出し手順

```
SXJ, 7      SM.RDISPLY
NOP         CODE, , base
NOP         STRING, , base
NOP         LENG, , base
NOP         COND, , base
```

・CODE；表示文字列の先頭座標（Y行，X列）を，それぞれ0バイト目，1バイト目に2進表現で入れる。

CODE

Y	X	

ただし　5≦Y≦19

1≦X≦80

・STRING；表示文字列が格納されているエリアの先頭番地。

STRING

A	B	C	D
E	F	G	H
I	J	K	L

（文字はEBCDIC表現）

・LENG；　表示文字数（ｎ）が入っている。

LENG

n

1≦n

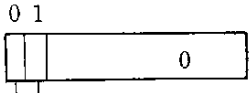
・COND；　I／Oエラーおよび文字オーバーの通知のための復帰情報。

呼び出し側でclearする必要はない。

I／Oエラーの時，0bit目がON

文字オーバーの時，1bit目がON

COND



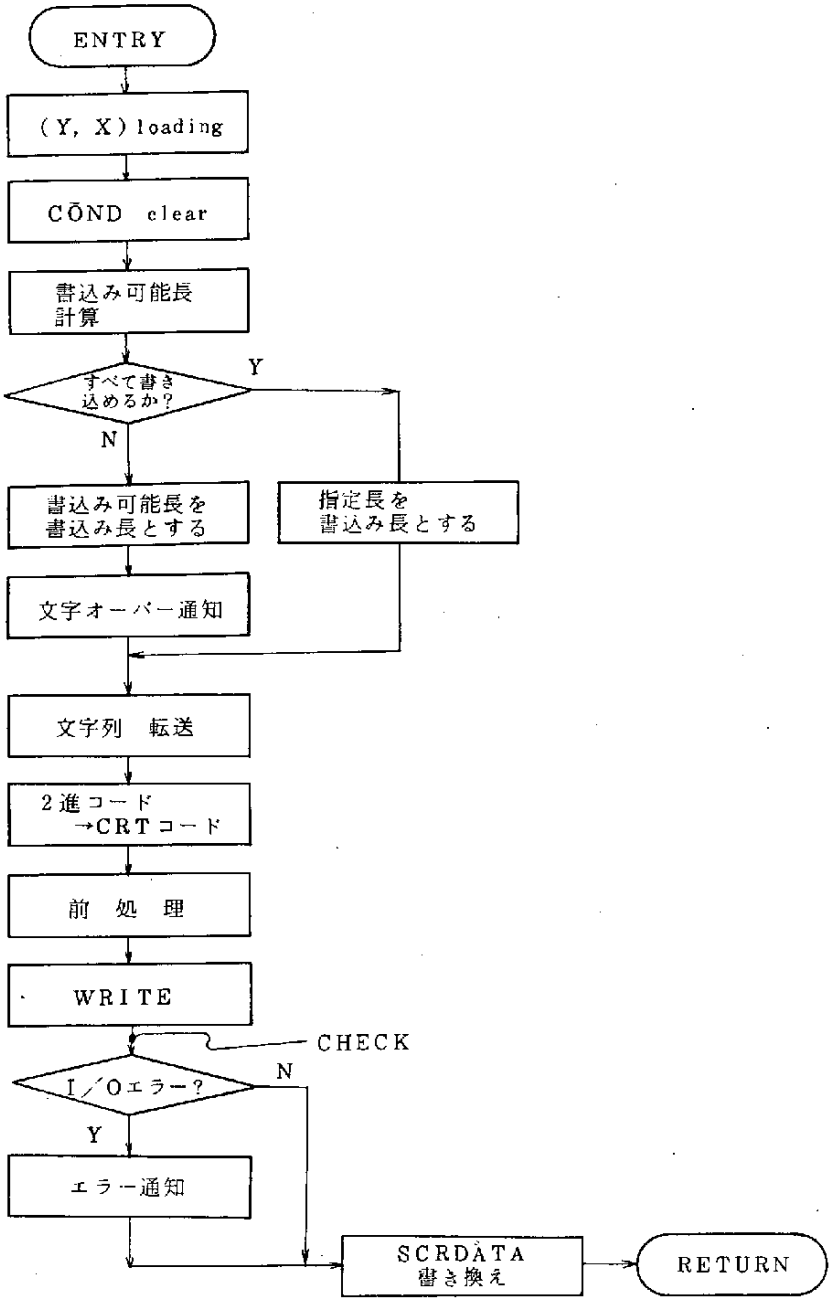
0；オーバーでない。

1；文字オーバー

0；I／O正常終了

1；I／Oエラー

3. ブロックチャート


ENTRY
(Y, X) loading
COND clear
書込み可能長
計算
すべて書き
込めるか?
Y
N
書込み可能長を
書込み長とする
指定長を
書込み長とする
文字オーバー通知
文字列 転送
2進コード
→CRTコード
前 処 理
WRITE
CHECK
I/Oエラー?
N
Y
エラー通知
SCRDATA
書き換え
RETURN

SM.SDISPLYルーチン

1. 機　能

・このルーチンは，X座標を与えて呼ぶ度にY座標を更新して文字列をスクリーンに表示する。

・与えられた文字列が「スクリーン」をはみ出してしまう場合には，表示せずに復帰情報の1bit目をONにして返す。

・文字列表示後，または文字オーバー表示後，Y座標がスクリーンをはみ出しても，Y座標の値をsetし直すことはしない。

・I／Oエラーの時，復帰情報ワードの0bit目をONにして返す。

2. 呼び出し手順

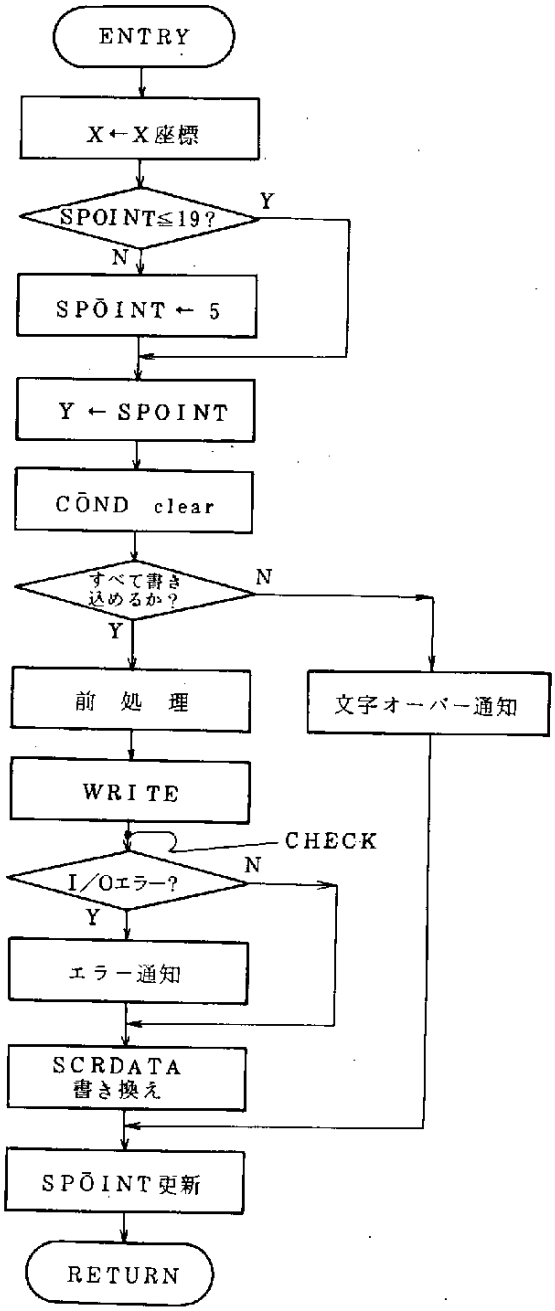
```
SXJ, 7          SM.SDISPLY
NOP             XCODE, , base
NOP             STRING, , base
NOP             LENG, , base
NOP             COND, , base
```

・XCODE ；表示文字列のX座標を1バイト目に2進表現で入れる。

XCODE | | X | | ただし　1≦X≦80

・他はSM.RDISPLYに同じ。

3. ブロックチャート


ENTRY
X←X座標
SPOINT≦19?
Y
N
SPOINT ← 5
Y ← SPOINT
COND clear
すべて書き込めるか?
N
Y
前 処 理
文字オーバー通知
WRITE
CHECK
I/Oエラー?
N
Y
エラー通知
SCRDATA 書き換え
SPOINT更新
RETURN

SM.EDISPルーチン

1. 機　能

・コミュニケーション・ライン（第22行）に，80文字以内の文字列を表示する。

・I／Oエラーの時は，復帰情報ワードの0bit目をONにして返す。

2. 呼び出し手順

```
SXJ，7          SM.EDISP
NOP             STRING，，base
NOP             COND，，base
```

・STRING，CONDの意味はSM.INMODEに同じ。

SM.EDISPRルーチン

1. 機　能

（出力）

・コミュニケーションライン（第22行）に，80文字以内の文字列を表示し，インプットエリアからのメッセージ入力，またはライトペン入力を要求する。

・入力された時点でエラー表示ラインをclearする。

・インプットエリアのclear指定がある場合，インプットエリアをclearする。

・インプットエリアに文字列表示の指定がある場合，指定文字数（160文字以内）を表示する。

・インプットエリアのclear指定の場合，文字表示指定の場合，共にインプットエリアの第1文字目にHDM（ヘディングマーク；➡）を表示し，カーソルをHDMの次の位置におく。

（入力）

・メッセージ入力の場合，入力文字数を第3パラメータの先頭語に，入力EBCDIC文字列を，第3パラメータの第2語以降にそれぞれ格納する。

また，コンディションコードのメッセージbitを，ONにして返す。

・ライトペン入力の場合，ライトペンset位置（Y，X）を第3パラメータの先頭語に2進表現で格納する。

また，コンディションコードのメッセージbitを，OFFにして返す。

㊟　ライトペン入力は，CRT画面上の位置に対応した特殊コードで入力する。したがってこのルーチンでは，呼び出し側に2進表現で通知するため，コード変換を行なう。

（エコー指定）

・SM.SPCFICSWの0bit目がONであるとエコー指定とみなし，コミュニケーションラインに表示した文字列，入力メッセージをラインプリンタに出力する。

㈱ ラインプリンタ上では，コミュニケーション表示文字列の頭に"COMM"，キーインメッセージの頭に"KEY"という記号を印字して，出力種別を表示する。

2. 呼び出し手順

```
SXJ, 7          SM.EDISPR
NOP             STRING 1, , base
NOP             STRING 2, , base
NOP             STRING 3, , base
NOP             COND, , base
```

・STRING1；コミュニケーションラインに表示する文字数（n）および文字列が格納されている。

STRING1

n			
A	B	C	D
E	F	G	H
.....			

ただし

$1 \leqq n \leqq 80$

・STRING2；インプットエリアに表示する文字数（n）および文字列が格納されている。

データの形式はSTRINGIと同じ。

ただし，$0 \leqq n \leqq 160$で，$n = 0$の場合，インプットエリアclear 指定となる。

また，表示文字列の第1文字（例の A ）はCRTに表示された時，HDMがくるので，画面上に表示されないのと同じことになる。

・STRING3；入力データを格納し，呼び出し側に返すための領域の先頭番地。

データの形式はSTRINGIと同じ。

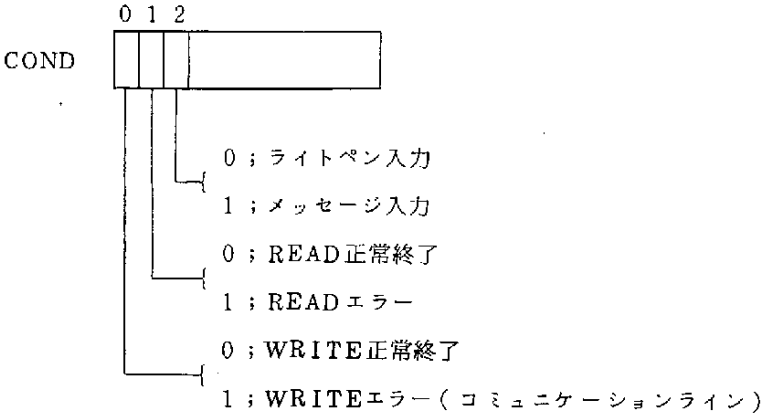
ただし，ライトペン入力の場合，第1語目にはライトペンset位置が2進表現で次の形式で入る。

STRING3

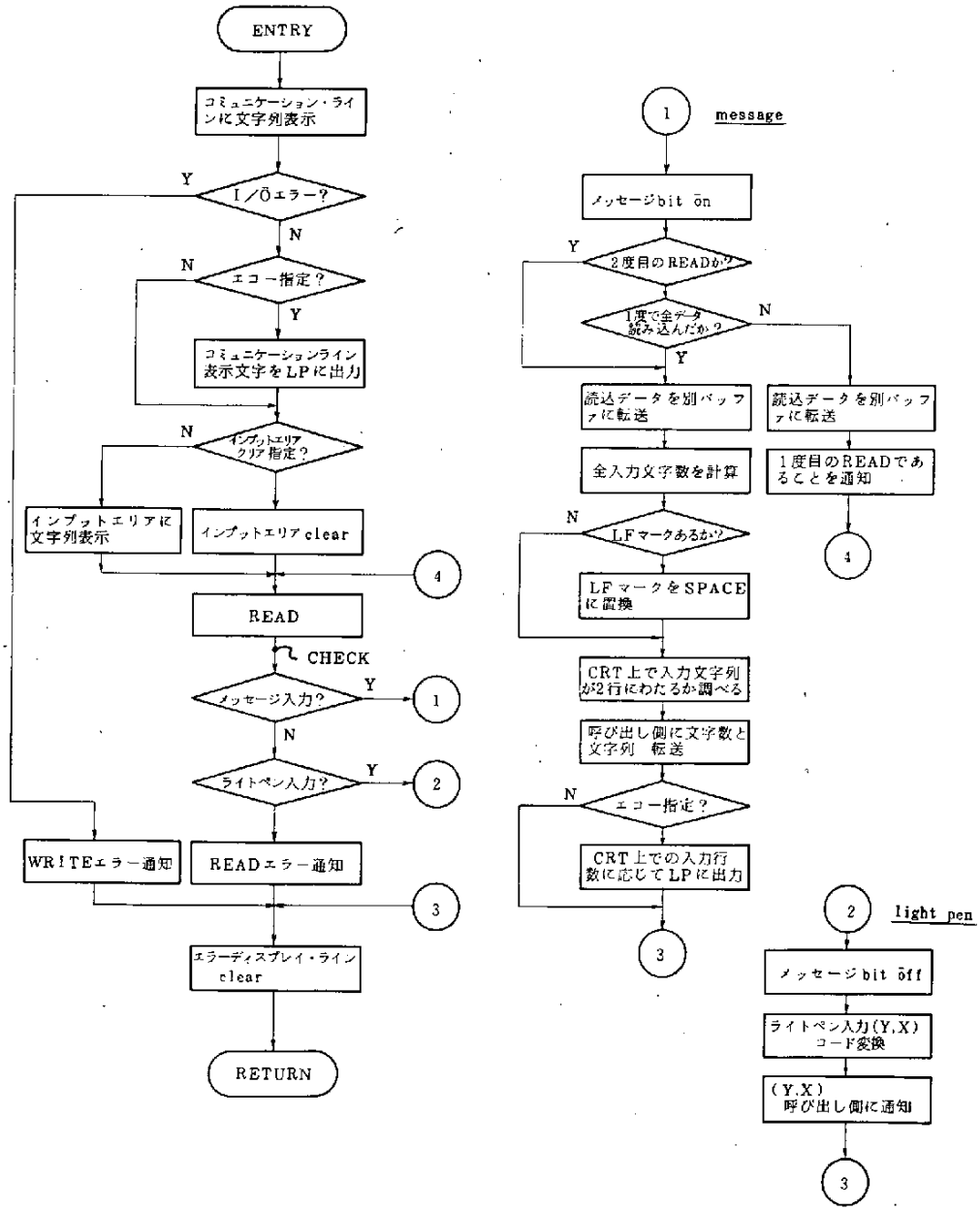
Y	X

（Y行，X列）

・COND ； I／Oエラー表示および入力データ種別を呼び出し側に通知するための１ワード。



3. ブロックチャート


ENTRY
コミュニケーション・ラインに文字列表示
I/Oエラー?
Y
N
エコー指定?
N
Y
コミュニケーションライン表示文字をLPに出力
インプットエリアクリア指定?
N
インプットエリアに文字列表示
インプットエリアclear
4
READ
CHECK
メッセージ入力?
Y
1
N
ライトペン入力?
Y
2
WRITEエラー通知
READエラー通知
3
エラーディスプレイ・ライン clear
RETURN
1
message
メッセージbit on
2度目のREADか?
Y
1度で全データ読み込んだか?
N
Y
読込データを別バッファに転送
全入力文字数を計算
N
LFマークあるか?
LFマークをSPACEに置換
CRT上で入力文字列が2行にわたるか調べる
呼び出し側に文字数と文字列 転送
N
エコー指定?
CRT上での入力行数に応じてLPに出力
3
読込データを別バッファに転送
1度目のREADであることを通知
4
2
light pen
メッセージbit off
ライトペン入力(Y,X)コード変換
(Y,X)呼び出し側に通知
3

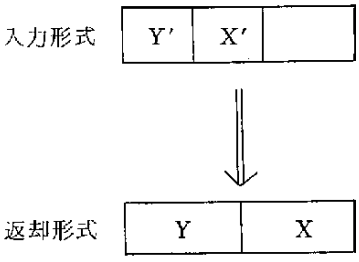
SM.LTPENルーチン

1. 機　能

・80文字以内の文字列をコミュニケーションラインに表示して，ライトペン入力を要求する。

・エラー表示ラインは，ライトペン入力がきた時点でclear する。

・ライトペン入力である画面上の位置（Y行，X列）を示すアドレスコードは，特殊なコードなので，これをbinaryに変換して指定されたWORDに格納し，呼び出し側に返す。



ライトペンの入力データは，作業エリアの先頭2bytesにアドレスコードで格納される。

Y，XはY行，X列を示すbinary コード。

・I/Oエラーおよび入力データの種別（メッセージかライトペンか）の通知は，指定されたWORDの対応するbitをONにすることでなされる

・SM.SPCFICSWの0bit目がONの時は，文字列をLPに出力する。

2. 呼び出し手順

```
SXJ，7          SM.LTPEN
NOP             STRING，，base
NOP             YXCODE，，base
NOP             COND，，base
```

・STRING；SM.INMODEの場合に同じ。

・YXCODE；ライトペンset位置がbinaryで格納される。

ただし，READエラーの場合およびユーザが間違ってメッセージをkey inした場合にはclearされる。

YXCODE

Y	X

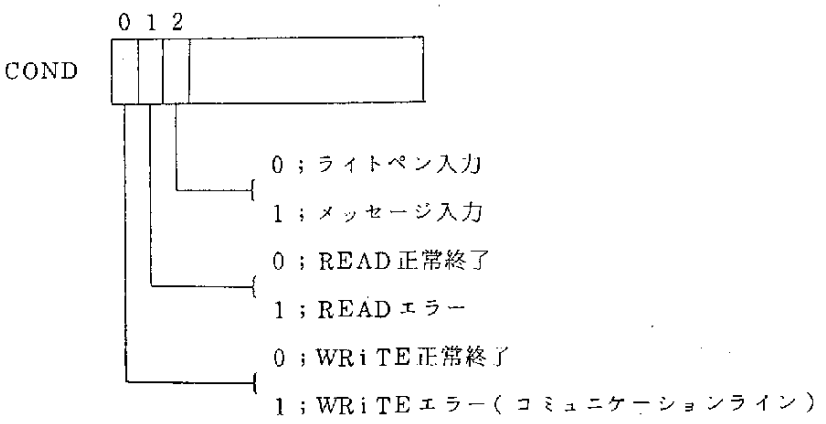
Y，Xはbineryで Y行X列の意。

・COND；　I/Oエラーおよび入力データ種別（メッセージかライトペンか）を呼び出し側に通知するための1　WORD。

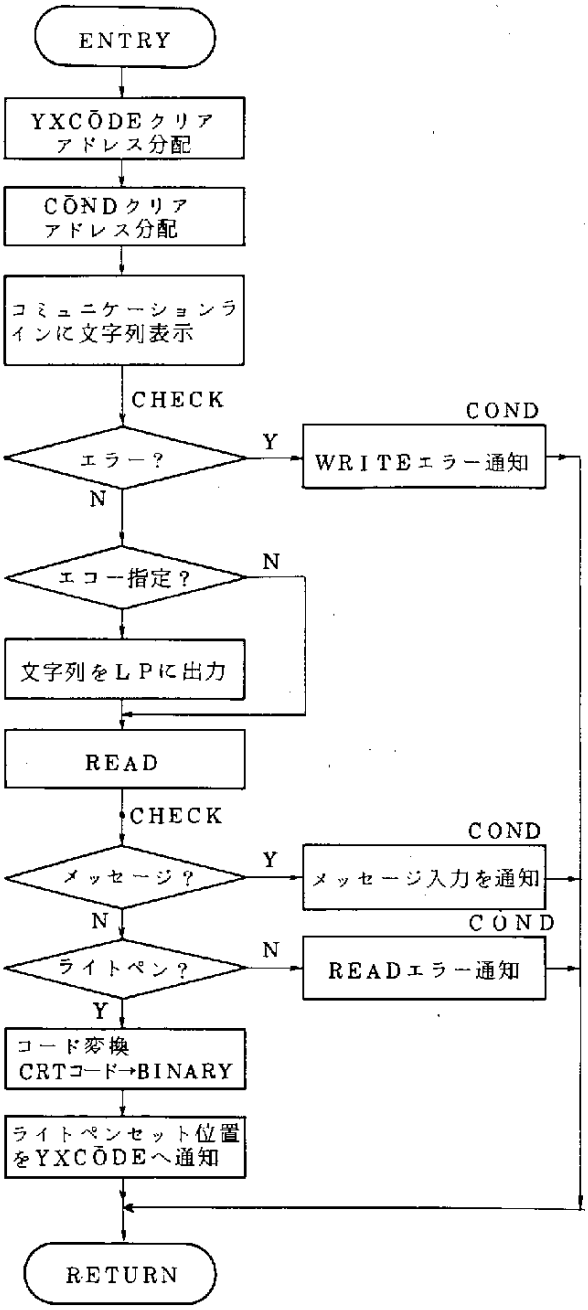
ユーザが指定通りにライトペンをsetすれば問題はないが，万一メッセージを

key inした場合には，2bit目をONにする。

なお，このエリアは，このルーチンでclearするので，呼び出し側でclear
する必要はない。



3. ブロックチャート


ENTRY
YXCODEクリア
アドレス分配
CONDクリア
アドレス分配
コミュニケーションライ
インに文字列表示
CHECK
エラー？
Y
COND
WRITEエラー通知
N
エコー指定？
N
文字列をＬＰに出力
READ
CHECK
メッセージ？
Y
COND
メッセージ入力を通知
N
ライトペン？
N
COND
READエラー通知
Y
コード変換
CRTコード→BINARY
ライトペンセット位置
をYXCODEへ通知
RETURN

SM.DCLEARルーチン

1. 機　能

・「スクリーン」（CRTの第5行～第19行）に一度に表示するための領域（SCRDATA)をclearするルーチン。

・このルーチンは，入出力媒体（CRT，LP）に直接的には関係しない。

2. 呼び出し手順

```
SXJ，7          SM.DCLEAR
```

SM.DPUTルーチン

1. 機　能

・「スクリーン」上の位置（Y，X），文字数，文字列を与えられて，「スクリーン」に一度に表示するための領域（SCRDATA）を書き換えるルーチンである。

・このルーチンは，入出力媒体（CRT，LP）に直接には関係しない。

2. 呼び出し手順

```
SXJ，7          SM.DPUT
NOP             CODE，，base
NOP             STRING，，base
NOP             LENG，，base
```

・CODE；　「スクリーン」上の位置（Y，X）を先頭2バイトに2進表現で格納する。

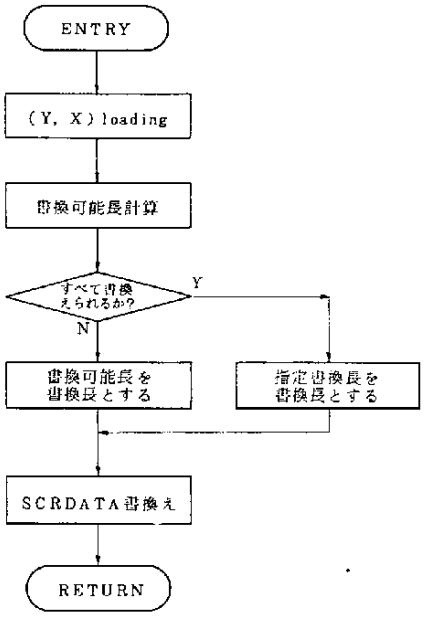
CODE

Y	X	

・STRING；文字列格納エリアの先頭番地。

・LENG；　文字数

3. ブロックチャート



SM.DDISPLAYルーチン

1. 機　能

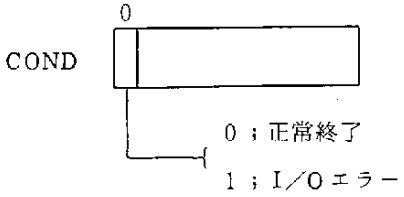
・「スクリーン」（第5行～第19行）に一度に表示する。

・表示データは，領域SCRDATA（80文字×15行＝1200文字分＝300語）に格納されている。

2. 呼び出し手順

SXJ, 7	SM.DDISPLAY
NOP	COND, , base

・COND；　I／Oエラー通知復帰情報



SM.HCOPYルーチン

1. 機　能

・領域SCRDATAの内容をラインプリンタに出力する。SCRDATAは「スクリーン」に一度に表示するデータが格納されている領域である。

2. 呼び出し手順

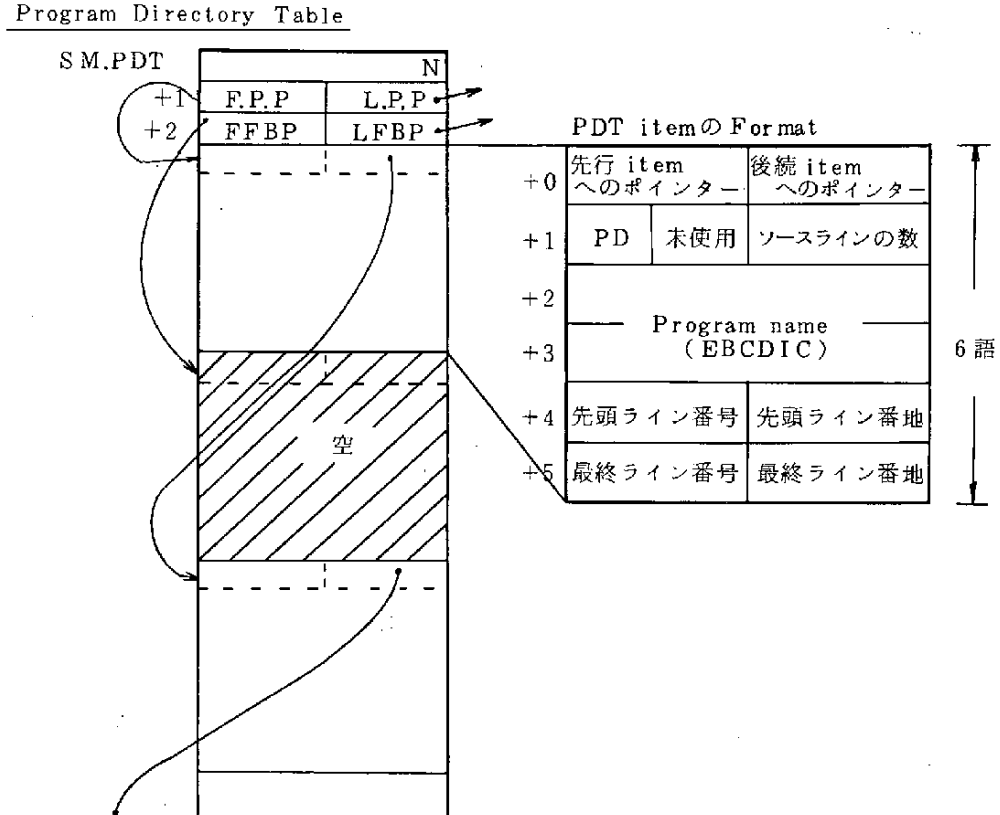
```
SXJ，7          SM.HCOPY
```

上記サブルーチン内で使用されるルーチンを以下の表に示す。

ルーチン名	機　　能
Exitパート	ルーチンではないが，個別ルーチンの出口になっている。
CRTREAD	CRTに対するREADマクロ命令発信。
CRTWRITE	CRTに対するWRITEマクロ命令発信。
LPWRITE	LPに対するWRITEマクロ命令発信。
CHECK	CHECKマクロ命令発信。
RDCLEAR	CRT　READ用データ領域をクリアーする。
LPCLEAR	LP用作業領域をクリアーする。
PREWRT	指定領域の第1語目に表示文字数，第2語目以降に表示文字列という決まったパターンのデータをCRTにWRITEする。
SUBWRT	文字数および文字列が格納されている語のアドレスを受けとって，指定文字列をCRTに表示する。
SUBLPWRT	文字数および文字列が格納されている語のアドレスを受けとって，指定文字列をLPに出力する。
CONVERT	2進表現で指定されたCRT画面上のアドレスをCRT用コードに変換する。
BUFINST	「スクリーン」に一度に出力するための領域SCRDATAの内容を書き換える。
RSDISP	SM.RDISPLYとSM.SDISPLYに共通のルーチン。R.OR.Sの0bit目がONならSM. SDISPLY指定。

3.4.3　ソース・プログラム管理用サブルーチン

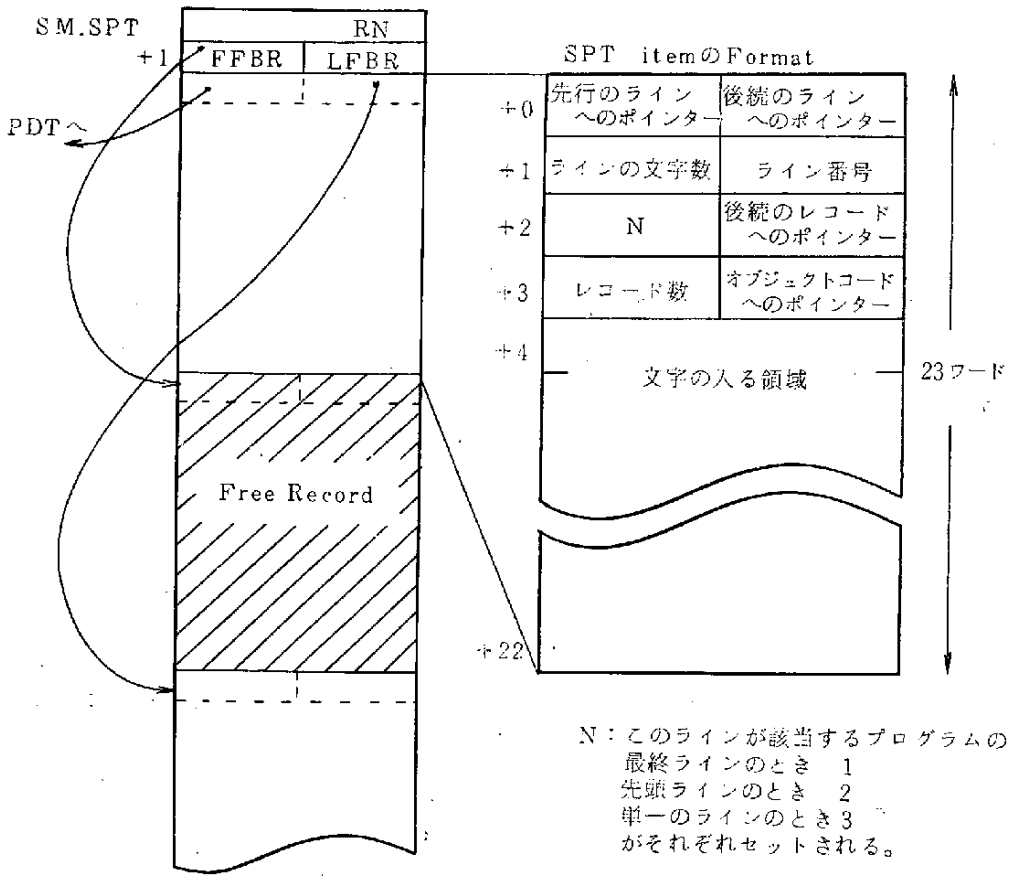
ここで述べるルーチン群は，ユーザがキーボードより入力したステートメントとそれから構成されているプログラムをセーブしておくための，ソース・プログラム・テーブル（SM. SPT，SPTと略記することもある。）およびプログラム・ディレクトリ・テーブル（SM.PDT，PDTと略記することもある。）を対象として，ステートメントやプログラムの登録，削除などの処理を行なうためのものである。



N　　：登録されているプログラム数
F.P.P ：プログラムチェインの先頭へのポインター
L.P.P ：プログラムチェインの先頭へのポインター
FFBP：フリーブロックチェインの先頭を示すポインター
LFBP：フリーブロックチェインの最後を示すポインター

図3－46　プログラム・ディレクトリー・テーブル

Source Program Table



N：このラインが該当するプログラムの
最終ラインのとき　1
先頭ラインのとき　2
単一のラインのとき3
がそれぞれセットされる。

※各ラインの2レコード目以降の0語目から2語目の前半語，および3語目の情報は無意味である。
また，後続レコードへのポインターは各ラインごとにレコードをチェイニングし各ラインの最終レコードのときは全ビット0になる。

RN　：まだ入りうるレコード数
FFBR：先頭の未使用レコードへのポインター
LFBR：最後の未使用レコードへのポインター

図3－47　ソース・プログラム・テーブル

SX.PSEARCHルーチン

1. 機　能

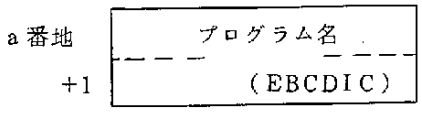
・プログラム名で，PDT内のアイテムを探し，その番地とそれに属する先頭のライン番号を返す。

2. 呼び出し手順

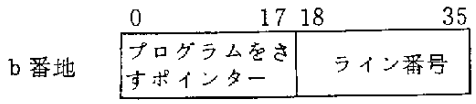
```
SXJ, 7        SX.PSEARCH
NOP           a, , base  ……①  利用者がセット
NOP           b, , base  ……②  SX.PSEARCHがセット
```

3. パラメータの説明

① プログラム名を指定する。



② 復帰情報をセットする。



※ 探し出したプログラムにラインがまったく登録されていないときには，ライン番号を0にして返す。

指定されたプログラムが未登録で発見できなかったときは，b番地をすべて0にして返す。

SX.ENTPルーチン

1. 機　能

プログラム名称とPDでPDTアイテムを生成し，PDTに登録する。

2. 呼び出し手順

```
LZI           n  ……①
SXJ, 7        SX.ENTP          } 利用者がセット
NOP           a, , base  ……②
NOP           b, , base  ……③
NOP           c, , base  ……④  } SX.ENTPがセット
```

3. パラメータの説明

① 登録するプログラムのPDを指定する。

② プログラム名を指定する。

a番地	プログラム名	
+1	(EBCDIC)	

③ 同じ名称のプログラムがあったとき，SX.PSEARCHのときのb番地と同じ。

b番地	プログラムをさすポインター	ライン番号

同じものがなかったときには，PDT内のアイテムの番地をb番地の前半語に返す。

④ コンディションコードを返す。

	0	1	
c番地			

※c番地　第0 bit：ONならば同じプログラム名がすでにPDT内にあった。

第1 bit：ONならばPDTがいっぱいで，このプログラムを登録できない。

SX.DLTPルーチン

1. 機　能

・指定されたプログラムに属するPDT内のアイテムおよびラインを，各々のPDTおよびSPT内から削除する。

2. 呼び出し手順

```
SXJ,7        SX.DLTP
NOP          a,,base ……①  利用者がセット
```

3. パラメータの説明

① 前半語にPDTのアイテムへのポインターを指定する。

a番地	PDTアイテムへのポインター	

SX.INSERTLルーチン

1. 機　能

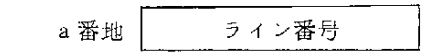
・指定されたラインをSPTに登録する。

2. 呼び出し手順

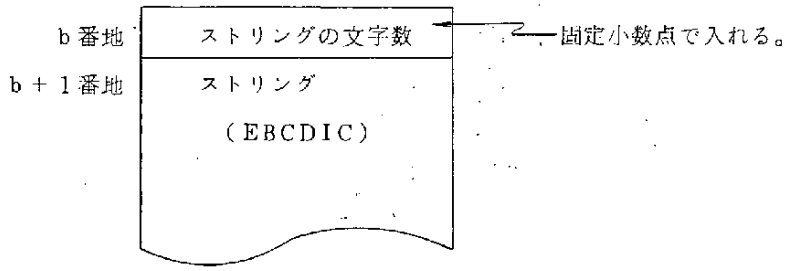
```
SXJ, 7        SX.INSERTL
NOP           a, , base  ……①
NOP           b, , base  ……②   } 利用者がセット
NOP           c, , base  ……③   SX.INSERTLがセット
```

3. パラメータの説明

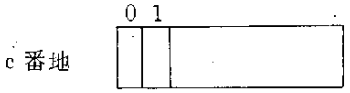
① ライン番号を指定する。



② 挿入するラインのストリングを指定する。



③ コンディションコードを返す。



※c番地　第0 bit：ONならSPTがいっぱいで，このラインを登録することができない。
第1 bit：ONなら同じライン番号を持つラインがすでにSPT中にあった。
今指定されたラインのストリングで置きかえる。

4. 一般規則

指定されたプログラムに属するライン群の中に，同じライン番号を持つラインがすでに登録されていたときには，このラインを削除し，今指定されたストリングで置きかえる。このとき，前のラインと番地は異なる。

SX.LSEARCHルーチン

1. 機　能

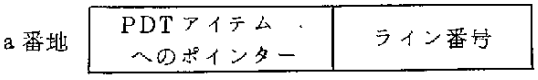
・PDTの番地とライン番号（LDW）を指定し，SPTを探し，ラインの番地を返す。

2. 呼び出し手順

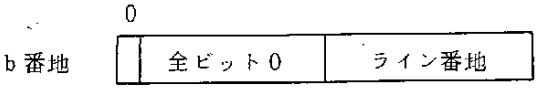
```
SXJ, 7        SX.LSEARCH
NOP           a, , base  ……①  利用者がセット
NOP           b, , base  ……②  SX.LSEARCHがセット
```

3. パラメータの説明

① 対象とするプログラムの，PDT上のアイテムを指すポインターとライン番号（LDW）を指定する。



② 復帰情報を返す



※b番地　第0ビット：ONなら対象のラインが見つからなかった。

SX.SETNXTPルーチン

1. 機　能

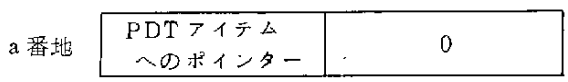
指定したプログラムに対して，PDT上での次のプログラムのアイテムを示すポインターと先頭ラインの番地を返す。

2. 呼び出し手順

```
SXJ, 7        SX.SETNXTP
NOP           a, , base  ……①  利用者がセット
NOP           b, , base  ……②
NOP           c, , base  ……③  } SX.SETNXTPがセット
```

3. パラメータの説明

① プログラムのPDTアイテムを指すポインターを指定する。



② 指定のプログラムの次のプログラムを示すアイテムのポインターと先頭ラインの番地を返す。

b番地

次のPDTアイテムへのポインター	ラインの番地

※もし指定したプログラムがPDTにおけるチェインの最後である時は，b番地を全て0にして返す。また，次のプログラムにラインがまったく登録されていない時はラインの番地を0に返す。

SX.REMVLルーチン

1. 機　能

・指定したラインをSPTから削除する。

2. 呼び出し手順

```
SXJ，7          SX.REMVL
NOP             a，，base ……①  利用者がセット
```

3. パラメータの説明

① 削除するラインの番号を指定する。

a番地

PDTアイテムへのポインター	ライン番号

4. 一般規則

指定したラインを削除した結果，そのラインの属するプログラムが持つラインが一つもなくなってしまった時は，該当するPDTアイテムも削除する。

指定したラインがSPT中に存在しなかったときは，何もしない。

SX.GETLNルーチン

1. 機　能

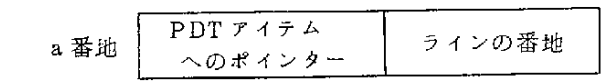
・指定したラインをSPTから取り出し，指定された領域に格納する。

2. 呼び出し手順

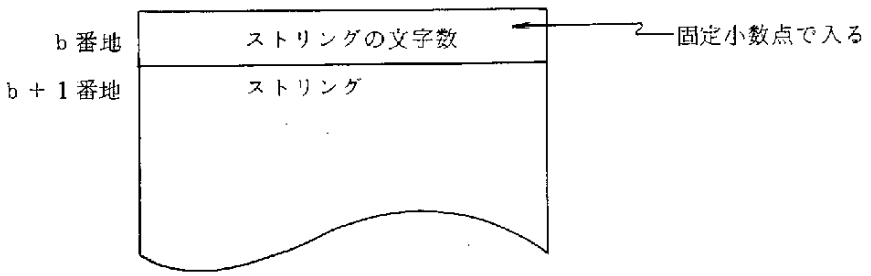
```
SXJ, 7        SX.GETLN
NOP           a, , base  ……①  利用者がセット
NOP           b, , base  ……②  SX.GETLNがセット
```

3. パラメータの説明

① PDTアイテムへのポインターおよびラインの番地からなるLAWを指定する。



② ラインのストリングおよびその文字数を返す。



SX.GETLNS ルーチン

1. 機　能

・指定したラインをSPTから取り出し，指定された領域に格納した後，LAWの値を次のラインに合わせる。

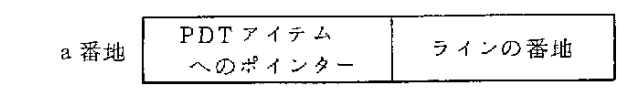
2. 呼び出し手順

```
SXJ, 7        SX.GETLNS
NOP           a, , base  ……①  利用者およびSX.GETLNSがセット
NOP           b, , base  ……②
NOP           c, , base  ……③  } SX.GETLNSがセット
```

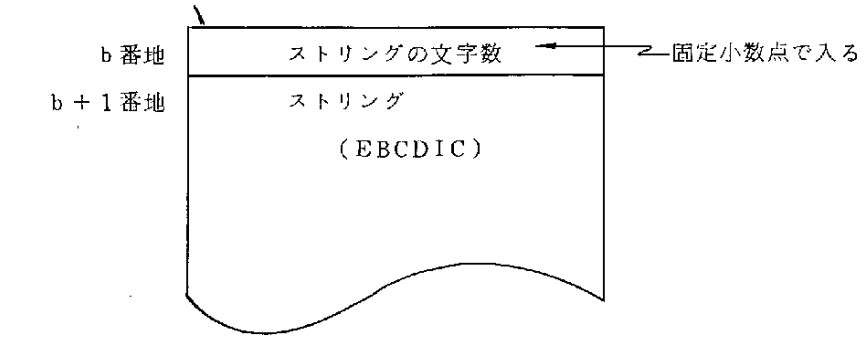
3. パラメータの説明

① PDTアイテムへのポインター，およびラインの番地からなるLAWを指定する。

このLAWは，処理を実行した後では次のラインに合わせて返す。



② ラインのストリングおよびその文字数を返す。



③ コンディションコードを返す。

c番地	固定小数点が入る。

※ 0：まだこのプログラムに続きのラインがある。
LAWを次のラインの値に合わせて返す。
1：今取り出したラインは，このプログラムの最終ラインである。
LAWの値は，次のプログラムの先頭ラインに合わせて返す。
2：今取り出したラインは，このプログラムの最終ラインであり，かつPDT中に次のプログラムがない。
LAWを0にして返す。

SX.NEALNLOWルーチン

1. 機　能

・指定した番号より大きいライン番号を持つラインのうち，最小のライン番号を持つものを探す。

2. 呼び出し手順

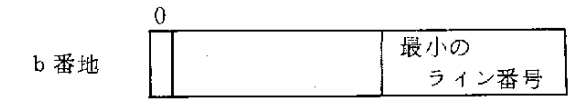
```
SXJ，7        SX.NEALNLOW
NOP           a，，base ……①  利用者がセット
NOP           b，，base ……②  SX.NEALNLOWがセット
```

3. パラメータの説明

① PDTアイテムへのポインター，およびライン番号からなるLDWを指定する。

a番地	PDTアイテムへのポインター	ライン番号

② 復帰情報を返す。



※ b番地 第0 bit：ONなら指定のラインが最大のラインであった。

SX.CNGLNSルーチン

1. 機 能

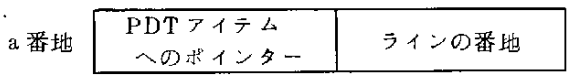
・ラインの番地を指定し，そのラインの番号を変更する。

2. 呼び出し手順

```
SXJ. 7        SX.CNGLNS
NOP           a,, base  ……①
NOP           b,, base  ……②   } 利用者がセット
NOP           c,, base  ……③   SX.CNGLNSがセット
```

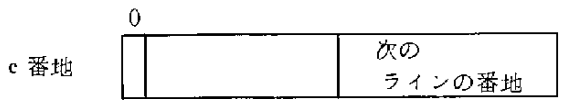
3. パラメータの説明

① PDTアイテムへのポインター，およびラインの番地からなるLAWを指定する。



② 書き込むべきライン番号を指定する。

③ 次のラインの番地，およびコンディションビットを返す。



※c番地 第0 bit：ONなら次のラインがない。

4. 一般規則

指定された番号を持つラインが最終ラインのときは，c番地の第0bitを"1"にして第1bitから第35bitに"0"をつめる。

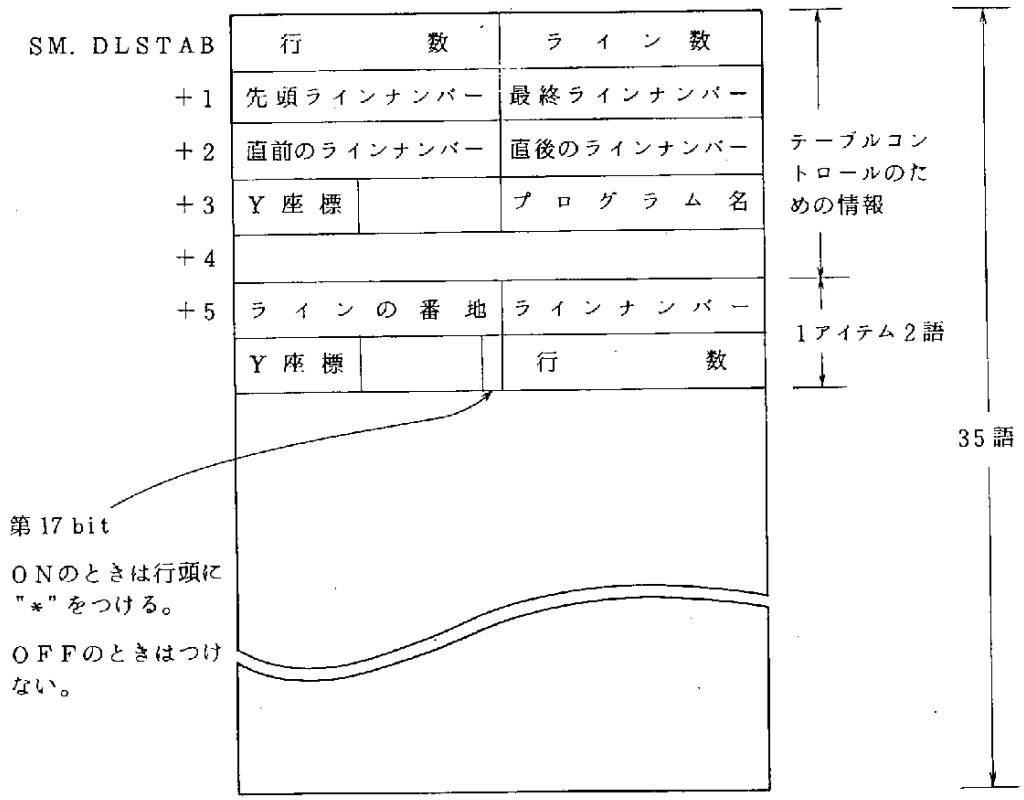
変更するラインが，該当するプログラムの先頭または最終ラインのときは，PDTアイテムも更新する。

3.4.4　ソース・プログラム表示用サブルーチン

ここで述べるルーチン群はＤＬＳテーブル（SM. DLSTAB）を対象として，ソースプログラム内のステートメントの登録，削除等に関した画面の変化を操作するためのものである。例えば，ステートメントを何行かずらして表示する場合，あるいは削除されたステートメントを画面上から消去する場合などに用いられる。

ＤＬＳテーブル

ディスプレイ画面上に表示されているラインを集めてリストしたテーブル



※　Y座標はディスプレイされたものだけについてセットされる。
　　従って，ここを見ればディスプレイされたものかどうかがわかる。

図3－47　ＤＬＳテーブル

SX. CRTDISPルーチン

1. 機　能

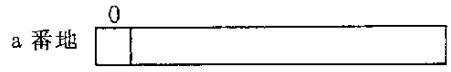
DLSテーブルに登録されているラインを，キャラクタディスプレイに表示する。

2. 呼び出し手順

```
SXJ, 7          SX. CRTDISP
NOP             a, , base ..... ① SX. CRTDISPがセット
```

3. パラメータの説明

① コンディションコードを入れる。



※ a番地 第0 bit ： ONなら　I／Oエラー
OFFなら　正常

4. 一般規則

- 表示したものについてのみY座標をセットする。したがってY座標が0であるということは，まだこのラインが表示されていない意味である。

SX. DSHIFTルーチン

1. 機　能

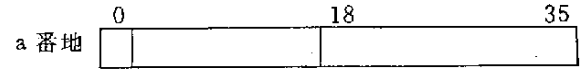
DLSテーブルの内容を指定したライン数だけずらす。

2. 呼び出し手順

```
SXJ, 7          SX. DSHIFT
NOP             a, , base ..... ① 利用者がセット
```

3. パラメータの説明

① シフトに関する指定をする。



※ a番地 第0 bit ： ONなら　逆方向シフト
OFFなら　順方向シフト

Lower half ： シフトするライン数（個定小数点数）．DLSテーブルに登録されているライン数を越えた場合，改ページを示す。

4. 一般規則

- ＤＬＳテーブルには最大15行まで登録可能である。
- １つのラインの一部分だけを表示することはしない。もし，はみだすときはそのラインは表示しない。
- 指定されたシフトするライン数が，ＤＬＳテーブルに登録されたライン数よりも大きいときには，ＤＬＳテーブルに登録されているライン数だけシフトし，それ以上のシフトは行わない。

ＳＸ．ＣＮＳＴルーチン

1. 機　能

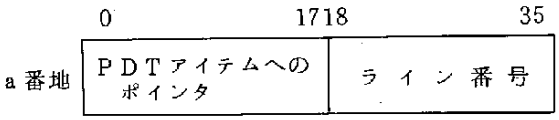
先頭ラインのＬＤＷを指定し，そのラインから表示画面に入りきれるだけのラインをＤＬＳテーブルに登録する。

2. 呼び出し手順

```
SXJ，7          SX．CNST
NOP             a，，base ……… ① 利用者がセット
```

3. パラメータの説明

① ＰＤＴアイテムへのポインタ，およびライン番号からなるＬＤＷを指定する。



4. 一般規則

- ＤＬＳテーブルには最大15行まで登録可能である。
- １つのラインの一部分だけ表示することはしない。もし，はみだすときはそのラインは表示しない。

ＳＸ．ＲＣＮＳＴルーチン

1. 機　能

SM．DLSTABに登録されているラインのうち＂＊＂印のついているものを，SM．DLSTABおよびSTPから取除き，SM．DLSTABを再編成する。

2. 呼び出し手順

```
SXJ，7          SX．RCNST
```

3. 一般規則

- 削除してできた部分は前につめる。
- 前につめた結果できたあきには，ＳＰＴから新たにラインを取り出して埋めて行き，空白がないようにする。

ＳＸ．ＳＤＩＳＰルーチン

1. 機　能

ディスプレイ画面に表示されているラインの先頭に，アスタリスク"＊"をつけるか，またはついている"＊"を消す。

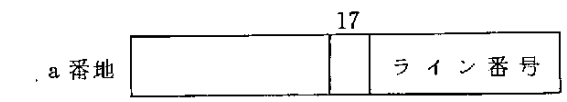
ＳＭ．ＤＬＳＴＡＢアイテム中のビットも操作する。

2. 呼び出し手順

```
SXJ，7        SX．SDISP
NOP           a，，base ……… ① 利用者がセット
NOP           b，，base ……… ② ＳＸ．ＳＤＩＳＰがセット
```

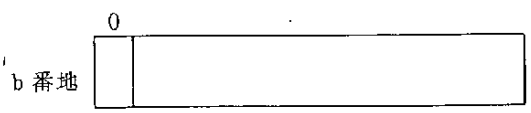
3. パラメータの説明

① "＊"に関する指定，およびライン番号を指定する。



※　a番地　第17 bit ：ＯＮなら　"＊"をつける。
　　ＯＦＦなら "＊"を消す。

② コンディションコードを返す



※　b番地　第0 bit ：ＯＮなら　Ｉ／Ｏエラー

SX. DADDLルーチン

1. 機　能

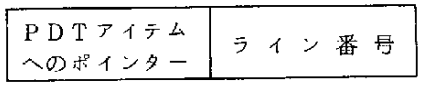
SM: DLSTABに1ライン追加，登録する。

2. 呼び出し手順

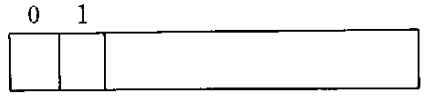
```
SXJ, 7        SX. DADDL
NOP           a , , base ..... ① 利用者がセット
NOP           b , , base ..... ② SX. DADDLがセット
```

3. パラメータの説明

① PDTアイテムへのポインター，およびライン番号からなるLDWを指定する。



② コンディションコードを返す



※ b番地　第0 bit ： ONなら指定されたライン番号を持つラインがSPT中にない。

第1 bit ： ONなら指定されたライン番号がSM. DLSTAB に登録されている範囲外である。

4. 一般規則

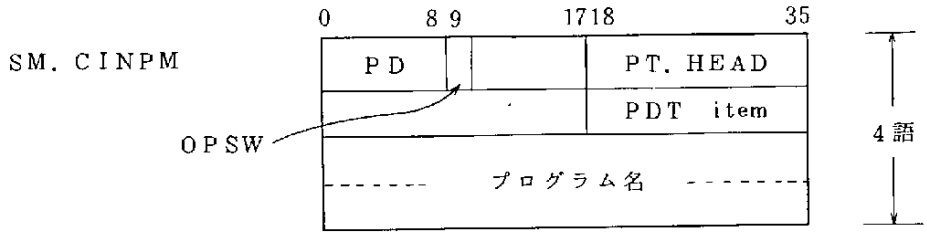
- ディスプレイ画面上で1番上が空白なことはない。
- ディスプレイ画面上の最後の行が一部しかDLSテーブルに表示できないときは，この行は表示しない。従って，ディスプレイ画面上の下部に空白ができることはある。
- 今，DLSテーブルに登録したラインは，必ず表示される。
- 先頭の直前のライン番号より小さいか，最後のラインの直後のラインのライン番号より大きなライン番号が指定されたときには，コンディションコード（b番地）の第1ビットを"1"にして何もしない。
- 指定されたライン番号がSPT中に存在しなかったときには，コンディションコード（b番地）の第0ビットを"1"にして何もしない。
- DLSテーブルがいっぱいで，なお新しくラインを追加するときは，原則として下から行を削除していく。

3.4.5　テーブル管理用サブルーチン

ここで述べるルーチン群は，表３－４に示すように各種のテーブルあるいはエリアを対象として，オープン，クローズ，登録，サーチなどの処理を行うためのものである。

表３－４　テーブル管理用サブルーチン

ルーチン名	対象	機能
SM. IDOPEN	SM. IDTABL	オープン処理
SM. IDCLOSE	（IDテーブル）	クローズ処理
SM. GETBLOCK	SM. IDTABL Global area Virtual area	ブロックの確保
SM. TSCHL	SM. IDTABL	現在の処理モードによるサーチ
SM. TSCHX	（ローカルテーブル）	Activity名によるサーチ
SM. ENTRY		登録
SM. THDSEARCH	Table Head	サーチ
SM. TABOPEN		開設
SM. LABSEARCH	Label Table	サーチ
SM. LABENTRY		登録
SM. TSCHC	Constant Table	サーチ兼登録
SM. LITENTRY	Literal Table	登録
SM. PRDOPEN	Procedure Record	オープン処理
SM. PRDPUT		組み立て処理
SM. PRDGET		分解処理
SM. PRDCLOSE		クローズ処理



- SM. CINPMはGLOBAL名とする。
- PD　：　現在処理中のPD（Program Descriptor）が入る。
- OPSW　：　ONならINPUT MODEの変更があって，しかもまだ未登録プログラムである事を示す。この場合，PD，PDT item は意味を持たない。
  OFFなら通常のケース。
- PT. HEAD：　IDTABLE内のTABLE HEADを示すポインター。
  （SM. IDTABLからの相対番地）
- PDT item　：　PDTテーブルのitemを示すポインター。
  （B 2 area，absolute）

※　SM. CINPMは現在処理中のプログラムに関する情報をセットするためのエリア。

図3－48　SM. CINPMの仕様

表3－5　SM. TABSTATEのビットの意味

bit 位置	(bit, bite)	bitがONの場合の意味
0	(0, 0)	TABLE HEADが開設された
1	(1, 0)	SYM. TABに登録があった
2	(2, 0)	SYM. TABのRWLISTがある
3	(3, 0)	LAB. TABに登録があった
4	(4, 0)	LAB. TABのRWLISTがある
5	(5, 0)	
6	(0, 1)	
7	(1, 1)	CONST TABに登録があった
8	(2, 1)	Literal TABに登録があった
9	(3, 1)	Virtual area の get block があった
10	(4, 1)	
11	(5, 1)	
12	(0, 2)	
13	(1, 2)	
14	(2, 2)	
15	(3, 2)	

SM. IDOPENルーチン

1. 機　能

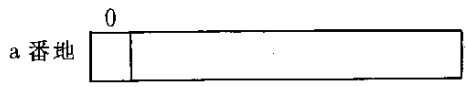
　各エリアの初期設定を行う。

2. 呼び出し手順

```
SXJ, 7        SM. IDOPEN
NOP           a , , 1  ..... ① SM. IDOPENがセット
```

3. パラメータの説明

① エラー情報を返す。



※　a番地　第0 bit　：　ONなら　　オープンエラー

　　OFFなら　正常

SM. IDCLOSEルーチン

1. 機　能

　ID TABLE処理のクローズを行う。

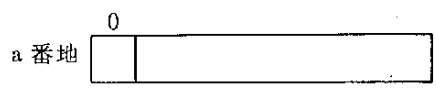
　指定によって，オープン後登録したものを正式に登録するか，破棄するかを行う。

2. 呼び出し手順

```
SXJ, 7        SM. IDCLOSE
NOP           a , , 1  ..... ① 利用者がセット
NOP           b , , 1  ..... ② SM. IDCLOSEがセット
```

3. パラメータの説明

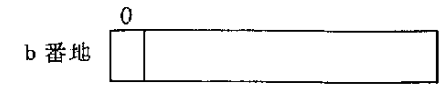
① 正式に登録するか否かを指定する。



※　a番地　第0 bit　：ONならば　　エラークローズ

　　OFFならば　正式登録

② Close Status を返す。



※ b番地 第0 bit : ONならばエラー、OFFならば正常

ID Close 処理

	エラー・クローズ	ノーマル・クローズ
SYM. TAB D chain	なにもしない。	登録処理。
SYM. TAB RWLIST	書き換え処理。	なにもしない。
LAB. TAB D chain	なにもしない。	登録処理。
LAB. TAB RWLIST	書き換え処理。	なにもしない。
CONST. TAB	なにもしない。	Temp → Regular
LITERAL. TAB	なにもしない。	Temp → Regular
TAB開設	なにもしない。	登録処理
ALLOCATION関係	なにもしない。	V, G, IDについて Temp → Regular 処理
WORK		

SM. GETBLOCKルーチン

1. 機 能

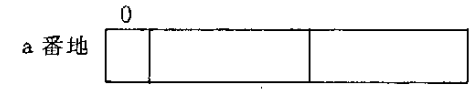
SM. IDTABL, Global area, Virtual area から適当なサイズのBlockを取り出す。

2. 呼び出し手順

```
SXJ, 7      SM. GETBLOCK
NOP         a,, 1  ..... ①  } 利用者がセット
NOP         b,, 1  ..... ②  }
NOP         c,, 1  ..... ③  } SM. GETBLOCKがセット
NOP         d,, 1  ..... ④  }
```

3. パラメータの説明

① BlockをGETする対象エリアを指定

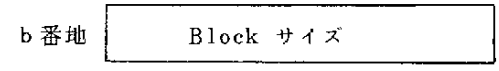


※　a 番地　第 0 bit ：ONなら　アドレス部はLOCAL TABLEのHEADをさすポインター

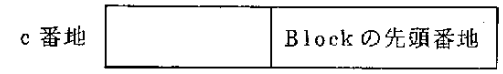
OFFなら　アドレス部はエリアを示す番号で

0…IDTABLE AREA, 1…GLOBAL AREA

② Block サイズを Binary で示す。



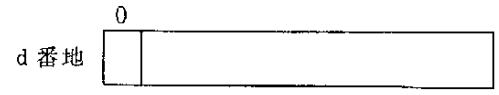
③ Block の先頭アドレスを返す。



※　番地部の意味は，対象エリアにより異なる。

対象エリア	番地部の意味
SM. IDTABL	テーブル内相対番地
Global area	Domain 先頭からの絶対番地
Virtual area	エンティティ内の相対番地

④ エラー情報を返す。



※　d 番地　第 0 bit ：ONならエラー，OFFなら正常終了

エラーは Block が取れない時にだけ出される。

SM. TSCHL ルーチン

1. 機　能

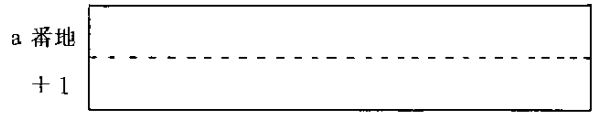
ID TABLE を，現在の処理モードに従って search して，目的の SYMBOL の実効Location を返す。

2. 呼び出し手順

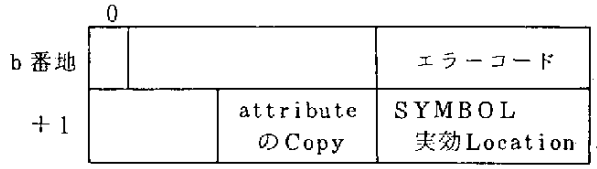
```
SXJ, 7        SM. TSCHL
NOP           a ,, 1  ..... ① 利用者がセット
NOP           b ,, 1  ..... ② SM. TSCHLがセット
```

3. パラメータの説明

① SYMBOL格納エリア先頭番地



② 復帰情報のエリアを指定する。



※ b番地 第0 bit ： ONならエラー; OFFなら正常

b＋1番地 1バイト：SYMBOL TABLE属性部のCopy この部分のみ

b＋1番地 Lower half ： SM. IDTABLからの相対番地

SM. TSCHXルーチン

1. 機 能

IDTABLEのLocal Table をActivity 名に従ってsearch して，目的のSYMBOL の実効Location を返す。

2. 呼び出し手順

```
SXJ, 7        SM. TSCHX
NOP           c ,, 1  ..... ③ ┐
NOP           a ,, 1  ..... ① ┘ 利用者がセット
NOP           b ,, 1  ..... ② SM. TSCHXがセット
```

3. パラメータの説明

①
②
} SM. TSCHLと同形式

③　Activity 名格納エリア先頭番地

SM. ENTRYルーチン

1. 機　能

変数名をSYMBOL　TABLEに登録する。

2. 呼び出し手順

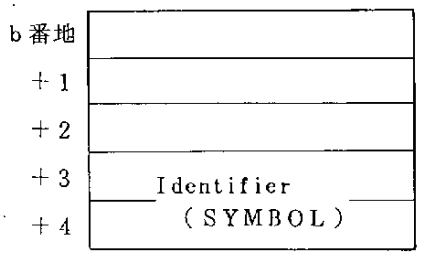
```
SXJ, 7        SM. ENTRY
NOP           a , , 1  ..... ①
NOP           b , , 1  ..... ②   } 利用者がセット
NOP           c , , 1  ..... ③
NOP           d , , 1  ..... ④   } SM. ENTRYがセット
NOP           e , , 1  ..... ⑤
```

3. パラメータの説明

①

※　a番地　第0 bit　：　ONなら定義

②　item

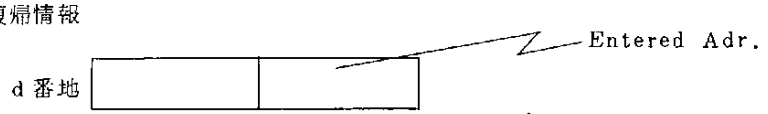


③　テーブルヘッドを示すポインター

※　PD　　：　Program Descriptor

　　THD　：　テーブルヘッドポインター

④　復帰情報



⑤ エラー情報

※ e 番地 第0 bit ： ONならエラー，OFFなら正常

SM. THDSEARCHルーチン

1. 機 能

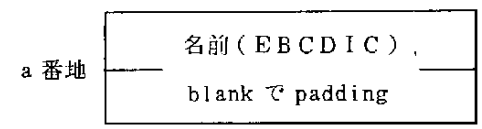
テーブルヘッドを search し，同じ item が登録されていればそのアドレスを返す。 無い場合はその旨知らせる。

2. 呼び出し手順

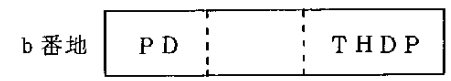
```
SXJ, 7      SM. THDSEARCH
NOP         a,, 1  ..... ①  利用者がセット
NOP         b,, 1  ..... ②  SM. THDSEARCH
NOP         c,, 1  ..... ③    がセット
```

3. パラメータの説明

① ローカルテーブル名



② search した結果を返す。

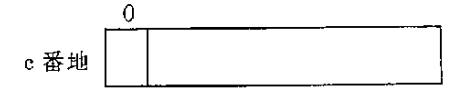


※ PD ： Program Deseriptor

THDP ： テーブルヘッドを示すポインター

（SM. IDTABLEからの相対番地）

③ エラー情報を返す。



※ c 番地 第0 bit ： ONならば正常，OFFならばエラー

SM. TABOPENルーチン

1. 機　能

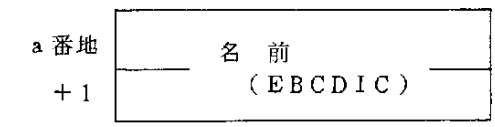
新しくテーブルヘッドを開設する。

2. 呼び出し手順

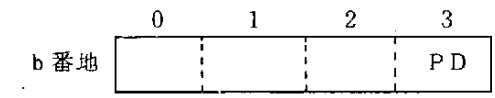
```
SXJ, 7      SM. TABOPEN
NOP         a,, 1  ..... ①  } 利用者がセット
NOP         b,, 1  ..... ②  }
NOP         c,, 1  ..... ③  } SM. TABOPENがセット
NOP         d,, 1  ..... ④  }
```

3. パラメータの説明

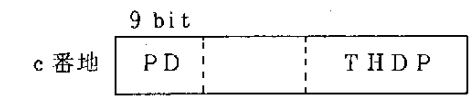
① 開設するテーブルヘッドの名前



② 開設するテーブルヘッドの属性



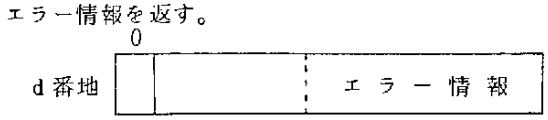
③ 開設したテーブルのポインターを返す。（SM. IDTABLからの相対番地）



※ PD　：Program Descriptor

THDP：テーブルヘッドを示すポインター

④ エラー情報を返す。



※ d番地　第0 bit ：ONならばエラー，OFFならば正常

SM. LABSEARCHルーチン

1. 機　能

ラベルテーブルをsearchし，同じラベルが登録されていればそのアドレスを返す。

無い場合は，その旨知らせる。

2. 呼び出し手順

```
SXJ, 7      SM. LABSEARCH
NOP         a , , 1  ..... ①
NOP         b , , 1  ..... ②   利用者がセット
NOP         c , , 1  ..... ③   SM. LABSEARCHがセット
```

3. パラメータの説明

① ラベル名を指定する。

a番地	EBCDIC （8文字）
＋1	blank で padding

② 対象とするテーブルを指定する。テーブルヘッドを示すポインターで指定する。

b番地		THDP

※ THDP ： 対象テーブルヘッドを示すポインター

（SM. IDTABLからの相対番地）

③ 復帰情報のエリアを指定する。

	0	
c番地		
＋1		

※ c番地　第0 bit ： ONなら，ラベルが見つからなかった事を示す。この場合c＋1番地の情報は意味を持たない。

OFFなら，ラベルが見つかった事を示す。

c＋1番地Lower half ： ラベルに対応するエントリーのアドレスを返す。

（SM. IDTABLからの相対番地）

SM. LABENTRYルーチン

1. 機　能

ラベルテーブルへアイテムの登録を行う。

ラベル処理に対しては，①単なる登録と②定義，を区別して扱い，これをuserの指定に依り行う。

ラベルテーブルのサーチ結果と，①②の指定によって処理が異なり，次の処理行列にもとづいて行う。

指定＼サーチ結果	な　い	有った(def. bit OFF)	有った(def. bit ON)
登　録	登　録 (def. bit OFF)	アドレスを返す	アドレスを返す
定　義	登　録 (def. bit ON)	itemにvalueを書き込み，アドレスを返す	2重定義でエラーとする

2. 呼び出し手順

```
LRI       n  ..... ⓪
SXJ, 7    SM. LABENTRY
NOP       a,, 1 ..... ①
NOP       b,, 1 ..... ②  } 利用者がセット
NOP       c,, 1 ..... ③
NOP       d,, 1 ..... ④  SM. LABENTRYがセット
```

3. パラメータの説明

⓪　登録，定義の別を指定する。

n＝0　　登録

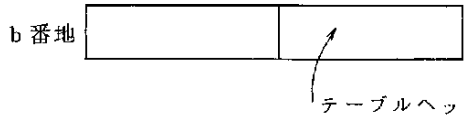
n＝1　　定義

※　それ以外を指定した場合はエラーとする。

①　ラベル名を指定する。

a番地	EBCDIC
＋1	8文字

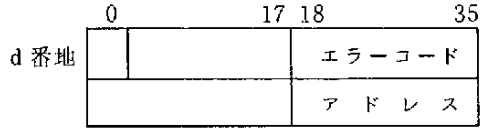
② 対象とするテーブルを指定する。



③ ラベルのvalue を指定する。これはn＝1即ち定義の場合のみ意味を持つ。

※ n＝0のときには，この内容は何であってもかまわない。

④ 復帰情報を返す。



※ d 番地の0 bit ： ONならエラー，OFFなら正常

d＋1 番地のLower half ： 登録したアドレスが入る。

（SM. IDTABLからの相対番地）

SM. TSCHCルーチン

1. 機能

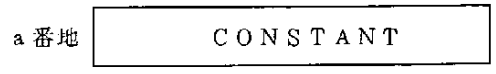
CONSTANT TABLEをsearchして，同じitemがなければ，そのCONSTANTを登録し，そのアドレスを返す。もし同じitemが登録済みであれば，そのアドレスを返す。

2. 呼び出し手順

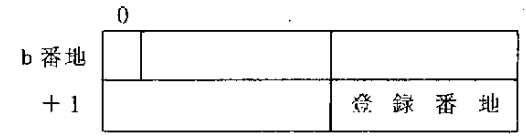
```
SXJ, 7        SM. TSCHC
NOP           a , , 1  ……① 利用者がセット
NOP           b , , 1  ……② SM. TSCHCがセット
```

3. パラメータの説明

① 登録するCONSTANTのある番地を指定する。

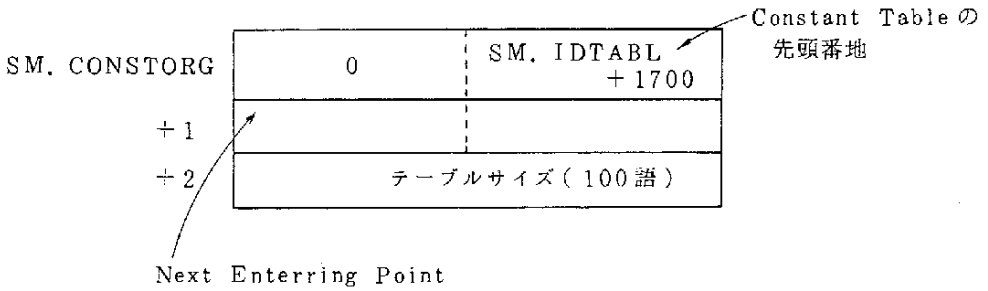


② CONSTANTを登録したアドレスと，エラー情報を返す。

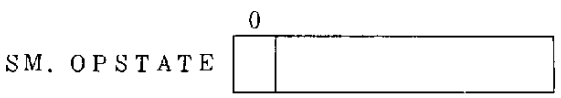


※　b番地　第0 bit ： ONならエラー，OFFなら正常

b＋1番地Lower half ： アドレスが入り，SM. CONSTORGの内容を加えるとabsolute valueが求まる。



※ SM. TSCHCルーチンで更新していく。



※　0 bit ： ONならOpenされた。

： OFFならOpenされていない。

※　SM. IDOPENルーチンでsetする。

SM. LITENTRYルーチン

1. 機　能

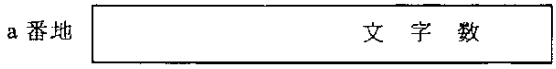
リテラルを登録し，そのアドレスを返す。

2. 呼び出し手順

```
SXJ, 7      SM. LITENTRY
NOP         a , , 1  ……①  }利用者がセット
NOP         b , , 1  ……②  }
NOP         c , , 1  ……③  SM. LITENTRYがセット
```

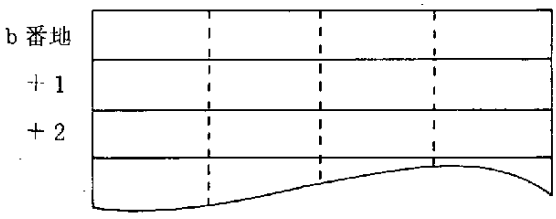
3. パラメータの説明

① 文字数を指定する。



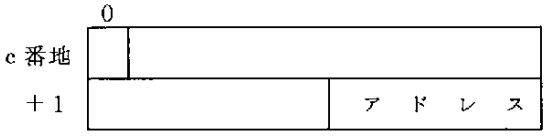
文字数 ＞ 0
0のときはエラーとなる。

② 登録エリアの先頭番地を指定する。



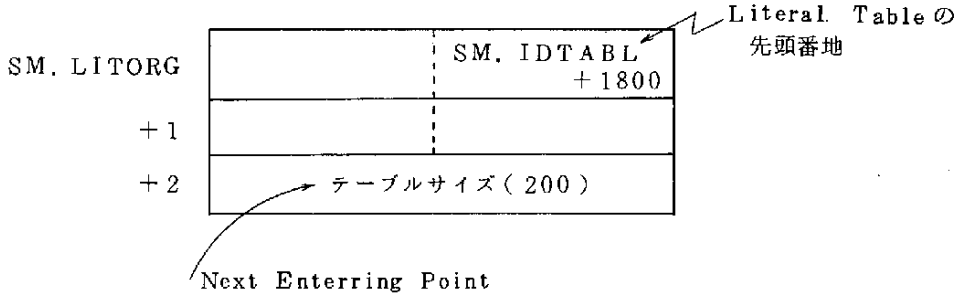
※ 余りはblankが入る。

③ 復帰情報を返す。



※ c番地 第0 bit ： ONならばエラー，OFFならば正常

c＋1番地 Lower half ： アドレスが入り，SM. LITORGの内容を加えると absolute value が求まる。



SM. PRDOPENルーチン.

1. 機 能

PROCEDURE RECORD作成ルーチンのオープン処理を行う。

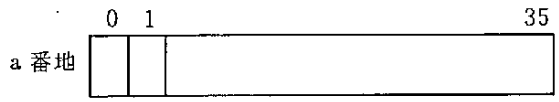
パラメータ指定でGET用，PUT用を使い分ける。

2. 呼び出し手順

```
SSJ       SM. PRDOPEN
NOP       a , , 1  ..... ① 利用者がセット
```

3. パラメータの説明

① オープンの仕方を指定する。



※ a番地 第0 bit ： ONならば 全体のオープン
OFFならば 各ステートメント単位のオープン
第1 bit ： ONならば GETオープン
OFFならば PUTオープン

注） 尚，オープンはSTPが呼び，ESPは呼ぶ必要はない。

SM. PRDPUTルーチン

1. 機 能

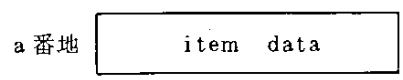
1 itemごとにPROCEDURE RECORD用のデータをもらい，完全なRecordに組み立てる。

2. 呼び出し手順

```
LR        a  ..... ① 利用者がセット
SSJ       SM. PRDPUT
```

3. パラメータの説明

① item data



注） このルーチンでは，itemのFormatに関するチェックは一切行わない。

SM. PRDGETルーチン

1. 機 能

呼ばれるごとに，PROCEDURE RECORDのitemを1つ返す。

データが無くなった場合には，EOR（End of Record）を知らせる。

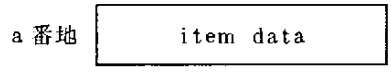
2. 呼び出し手順

```
SSJ       SM. PRDGET
NOP       a,, 1  ..... ①
NOP       b,, 1  ..... ②
```

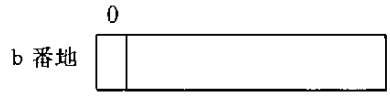
SM. PRDGETがセット

3. パラメータの説明

① item data



② ＥＯＲを知らせる



※ b番地 第0 bit ： ONならEORである。この場合パラメータ①の内容は保障しない。

注） 最終itemを取り出し，次にこのルーチンが呼ばれるとＥＯＲとなる。

SM. PRDCLOSEルーチン

1. 機 能

PROCEDURE RECORD作成ルーチンのクローズ処理を行う。

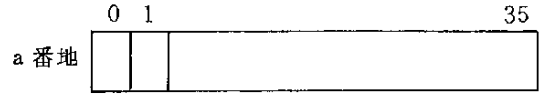
パラメータ指定でＧＥＴ用，ＰＵＴ用を使い分ける。

2. 呼び出し手順

```
SSJ       SM. PRDCLOSE
NOP       a,, 1  ..... ① 利用者がセット
```

3. パラメータの説明

① クローズの仕方を指定する。



※ a番地 第0 bit ： ＯＮならば 最終クローズ
ＯＦＦならば 各ステートメント単位のクローズ
第1 bit ： ＯＮならば ＧＥＴクローズ
ＯＦＦならば ＰＵＴクローズ

注） 尚，クローズはＳＴＰが呼ぶ。

3.4.6　スケジューリング・ステートメントの処理

(1)　SCHEDULEステートメント

（機能）

このステートメントは，パッシブ状態にある離散型プロセス，もしくはスケジュールド状態にある連続型プロセスをスケジュールする。

（一般形式）

SCHEDULE　＜プロセス＞　＜スケジュール句＞　；

（スケジュール方法の指定の仕方）

スケジュール方法の指定の仕方として，SIMBOL－Ⅱでは大きく分けて二通りの方法がある。第一の方法は，時間を指定する方法で，この場合，直接実行すべき時刻を指定する方法と現在の時刻からの相対時間を指定する方法の二形式に分けられる。第二の方法は，プロセスを指定する方法で，既にスケジュールされているプロセスの直前もしくは直後を指定する方法である。この場合の実行予定時刻ETは，その指定されたプロセスのETと等しくなる。以上の様なスケジュール方法の指定をSIMBOL－Ⅱでは＜スケジュール句＞と呼んでいる。ただし，スケジューリングの対象となるプロセスが連続型の時は，第二の方法は許さない。というのは，前に連続型プロセスの処理の概略の所でふれたように，連続型プロセスのイベントの発生は，スケジュール・テーブルにつながっているイベント・ノーティスが毎回スキャンされ，SC＝1のものに対してそのプロセスのイベントが発生される。即ち，SCの値は昇順に並んでいるわけではなく，連続型プロセスのイベント・ノーティスはプライオリティ順に並んでいるのである。もし，この第二の方法を許す事にすれば，プライオリティの順序が崩されてしまうことになる。ある特定のプロセスのイベントの前後に，あるプロセスのイベントを発生させたければ，プロセスのプライオリティを利用するか，或いは，あらかじめそれらのプロセスをスケジュールしておき，SCに大きな値をセットしておく。そうして，しかるべき時にそれらのプロセスをスケジュールし直すという方法を用いれば，第二の方法と同様な操作を行なうことができる。つまり，離散型プロセスのスケジューリングは実行予定時刻ETの値をセットし，それをETの順，プライオリティの順にリンクし直すことにより成されるが，連続型プロセスのスケジューリングは，プロセスの発生と同時にスケジュールド状態におかれ，SCの値を変更していく事により離散型プロセスのスケジューリングと同様な機能を果たすように作られている。従って，スケジュール・テーブルとリンクされているイベント・ノーティスは，プロセスの発生時にプライオリティ順にリンクされ，既にリンクされているイベント・ノーティスの順序関係は崩されることがない。

所で，第一の方法はさらにPRIORの指定が許されている。このPRIORの指定をすると，今スケジュールしようと思っているプロセスのイベント・ノーティスと同じET，同じプライオリ

ティをもつイベント・ノーティスがある場合，それらの先頭にスケジュールされる。このPRIORの指定がない時は，同じET，同じプライオリティをもつイベントがある場合には，それらの最後にスケジュールされる。このPRIORの指定は，スケジュールの対象となるプロセスが連続型の時には，上で述べたと同様な理由で許されていない。

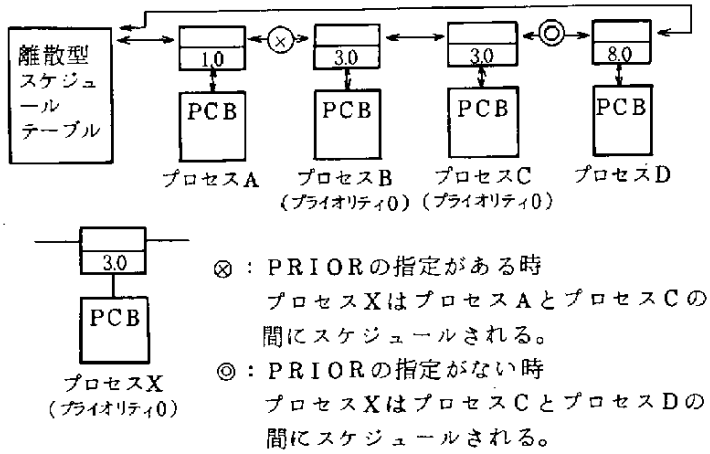


図3－49　PRIORの指定

以上のスケジュール句についてまとめると次のようになる。

① AT　　<時　刻>　〔，PRIOR〕

② DELAY　　<時　間>　〔，PRIOR〕

③ AFTER　　<プロセス>

④ BEFORE　　<プロセス>

形式①　：<時刻>は現在時刻以上でなければならない。
　　PRIORの指定は離散型プロセスだけに許される。

形式②　：<時間>は「0．0」以上でなければならない。
　　PRIORの指定は離散型プロセスだけに許される。

形式③，④：<プロセス>はスケジュールされていなければならない。
　　この形式は離散型プロセスだけに許される。

（ステートメントの処理）

対象となるプロセスが離散型の時は，まだスケジュールされていないので，まずイベント・ノーティスをフリー・ストレッジからとってきて対象となるプロセスのPCBとリンクする。そして，ETをスケジュール句に従ってセットし，スケジュール句によって指定された位置に入れる。又，スケジュール・テーブルのチェイン・アイテム個数は「1」増やされる。

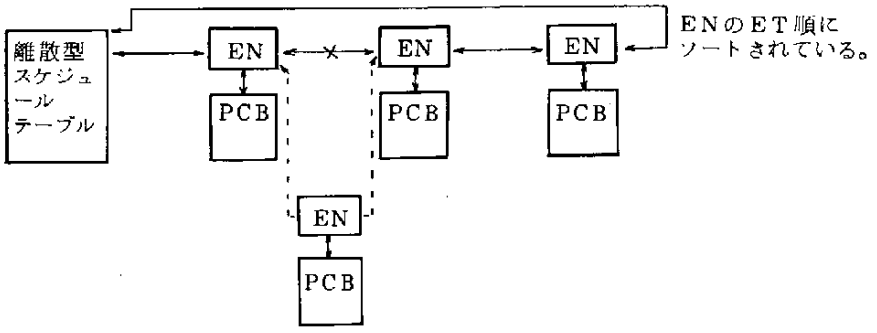


図３－50　SCHEDULEステートメントの操作（離散型）

一方，対象となるプロセスが連続型の時は，既にスケジュールされているので，ＳＣをスケジュール句に従ってセットする。例えば，あるプロセスＣＰ１のイベントの発生をしばらく停止しておきたいとすれば，

SCHEDULE　CP1　DELAY　30.0 ；

といった様な記述を他のプロセスの中で行なえばよい。このようなスケジューリングは，ＣＰ１のプロセスのＥＮのＳＣ部に 30.0／⊿Ｔをセットする事によって成される。ここで⊿Ｔは最小ステップサイズの事である。⊿Ｔ指定の時は，（指定時刻－現在時刻）／⊿ＴがＳＣ部にセットされる。

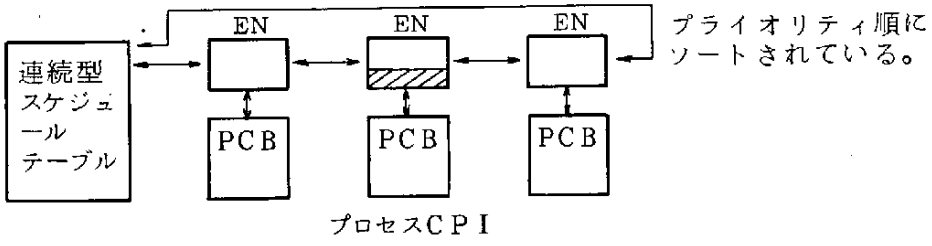


図３－51　SCHEDULEステートメントの操作（連続型）

既に述べてきたように，連続型プロセスのスケジューリングは，イベント・ノーティスＥＮのＳＣ部を処理することによって成されるわけである。従って，ある条件を満足した時にプロセスのイベントを発生，あるいは延期させたい時は，ＩＦステートメントを用いて，その条件を満足したら，スケジューリング・ステートメントを実行するように記述すればよいわけである。例えば，グローバル・データ（ＧＤＡＴＡ）の値が「2.0」になったら，連続型プロセスＣＰを「30.0」の間発生しないようにするには，次の様に記述すればよい。

IF（GDATA＃2.0）　SCHEDULE　CP　DELAY　30.0 ；

なお，従来の離散型システムSIMBOLに連続型システムの機能を追加して開発されたＳＩＭＢＯＬ－Ⅱにおいて，連続型プロセスのスケジューリング・ステートメントは，離散型プロセスのスケジューリング・ステートメントと同じ記述ができるようにしてある。従って，従来の離散型スケジューリングの概念で連続型プロセスのスケジューリングが行なえる。しかし，実際の連続型プロセスのスケジューリングの操作は，ＳＣ部の書き換えが主な仕事であるので，SCHEDULE，RESCHEDULE，DELAYの３つのステートメントの差異はほとんどないと言える。

（ランタイムの呼び出し手順）

1. 呼び出し手順

```
LZI          Schedule code
SXJ, 7       SM. SCHED
NOP          a, *, base
NOP          b, *, base
```

2. パラメータの説明

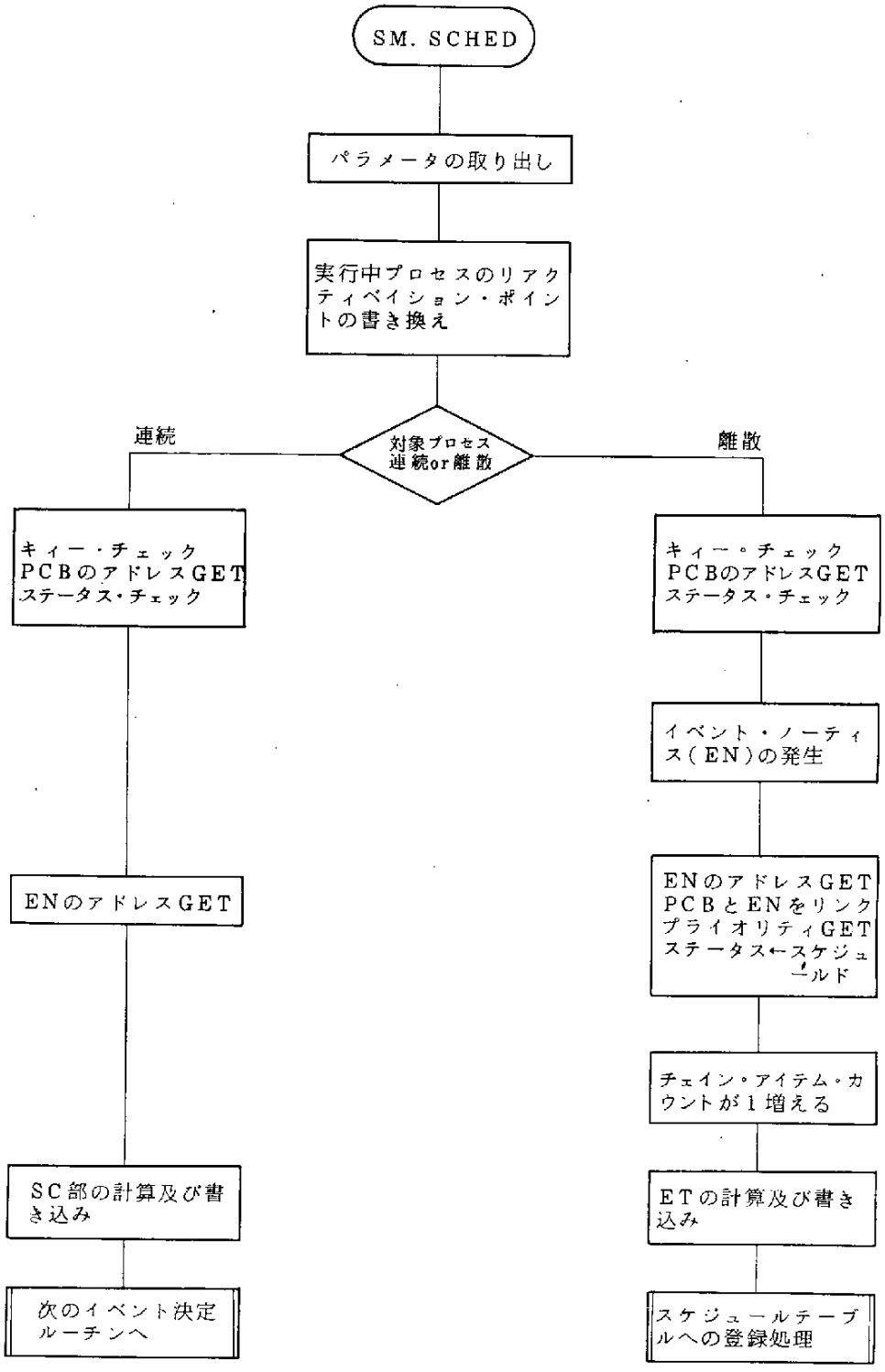
Schedule code ： ＡＴ指定なら　0
ＤＥＬＡＹ指定なら　1
ＢＥＦＯＲＥ指定なら　2
ＡＦＴＥＲ指定なら　3

※　PRIOR指定があれば上記の値に10を加えておく。

a ： スケジュールの対象となるプロセス名の入っているareaのアドレス。SELE 指定はa番地の内容をzeroにする。

b ： AT，DELAY指定の場合はTimeの入っているアドレス，AFTER，BEFORE指定の場合は，プロセス名の入っているアドレス。

（処理概略フロー）


SM. SCHED
パラメータの取り出し
実行中プロセスのリアクティベイション・ポイントの書き換え
対象プロセス
連続or離散
連続
離散
キィー・チェック
PCBのアドレスGET
ステータス・チェック
ENのアドレスGET
SC部の計算及び書き込み
次のイベント決定ルーチンへ
キィー・チェック
PCBのアドレスGET
ステータス・チェック
イベント・ノーティス（EN）の発生
ENのアドレスGET
PCBとENをリンク
プライオリティGET
ステータス←スケジュールド
チェイン・アイテム・カウントが1増える
ETの計算及び書き込み
スケジュールテーブルへの登録処理

(2) RESCHEDULEステートメント

（機能）

現在アクティブ，またはスケジュールド状態にあるプロセスをスケジュールしなおす。

（一般形式）

RESCHEDULE　＜プロセス＞　＜スケジュール句＞　；

（処理）

対象となるプロセスが離散型の時は，このステートメントで指定されるプロセスは既にスケジュールされているので，イベント・ノーティスENが存在する。そこで，まず始めにこの対象となるプロセスのENを捜し出し，スケジュール句に従ってET部をセットする。次に，このENを一旦スケジュール・テーブルから切り離す。切り離す時の先行EN，後続ENのポインター部の書き換えは，対象プロセスのENの位置，即ち，ENが先頭にあった時（スケジュール・テーブルとENの間），中間にあった時（ENとENの間），最後にあった時（ENとスケジュール・テーブルの間）により異なる。次にスケジュール句に従って指定された位置に入れる。

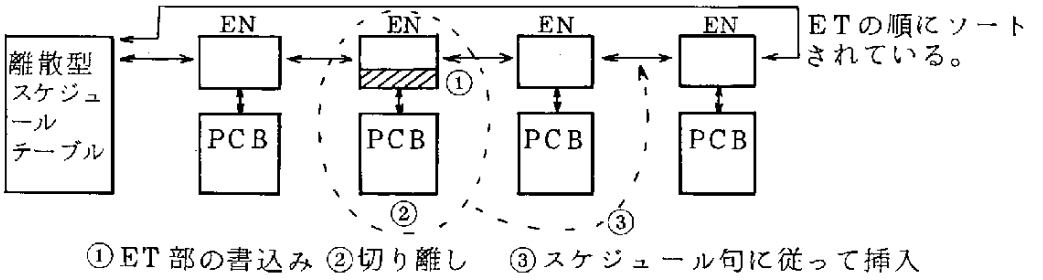


図3－52　RESCHEDULEステートメントの操作（離散型）

対象プロセスが連続型の時の操作はSCHEDULEステートメントとほとんど同様である。

（ランタイムの呼び出し手順）

1. 呼び出し手順

```
LZI          Schedule code
SXJ, 7       SM. RESCHED
NOP          a, *, base
NOP          b, *, base
```

2. パラメータの説明

Schedule code ： AT指定なら　0

DELAY指定なら　1

BEFORE指定なら　2

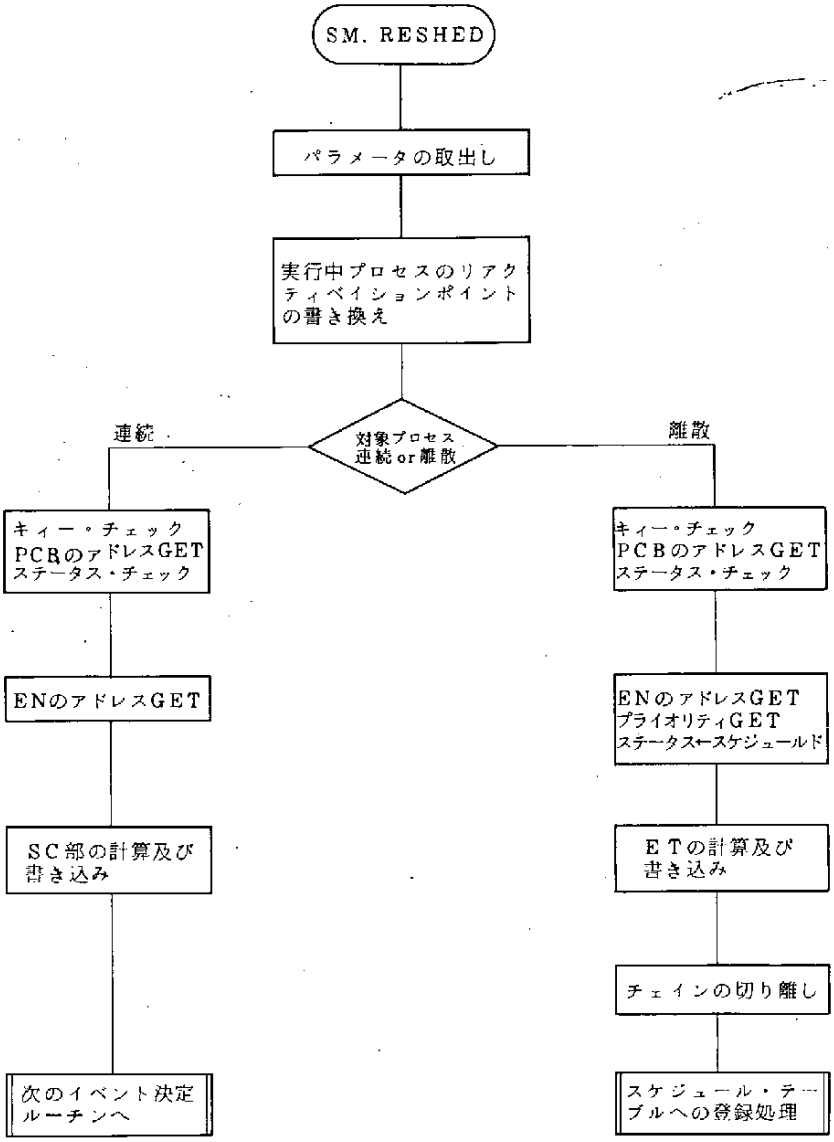
AFTER指定なら　3

※　PRIOR指定があれば上記の値に10を加えておく。

a ： スケジュールの対象となるプロセス名の入っているareaのアドレス。SELF指定はa番地の内容をzeroにする。

b ： AT，DELAY指定の場合はTimeの入っているアドレス，AFTER，BEFORE指定の場合は，プロセス名の入っているアドレス。

（処理概略フロー）



(3) DELAYステートメント

（機能）

現在アクティブなプロセスの実行を指定時間だけ中断し，スケジュールド状態にする。

（一般形式）

DELAY　＜時間＞　；

（処理）

対象となるプロセスが離散型の時は，現在離散型処理中であるわけで，その対象プロセスのイベント・ノーティスENは離散型スケジュール・テーブルとつながっている。そこでまず，このENのET部を書き込み，スケジュール・テーブルとENのリンクを切り離す。次に，今書き込んだETに従って，しかるべき位置にENをセットする。

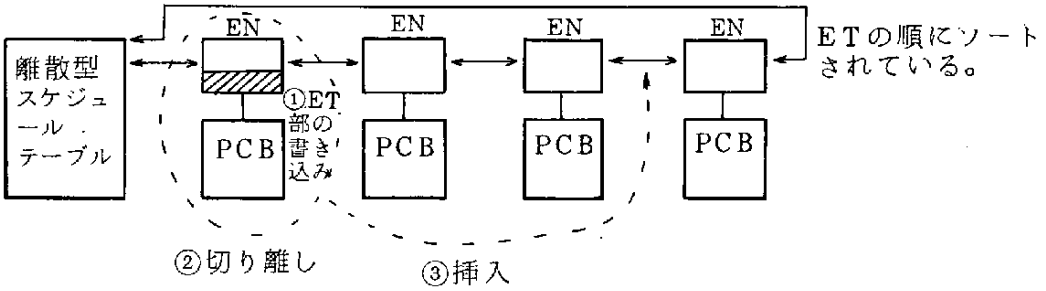


図3－53　DELAYステートメントの操作（離散型）

なお，このステートメントはGPSSのADVANCEブロックに相当するステートメントで，プロセスの時間消費を表わすのに使われる。従って，このステートメントをプロセデュアー・プログラムの中で使うことはできない。

対象プロセスが連続型プロセスの時の操作はSCHEDULEステートメントとほとんど同じである。

（ランタイムの呼び出し手順）

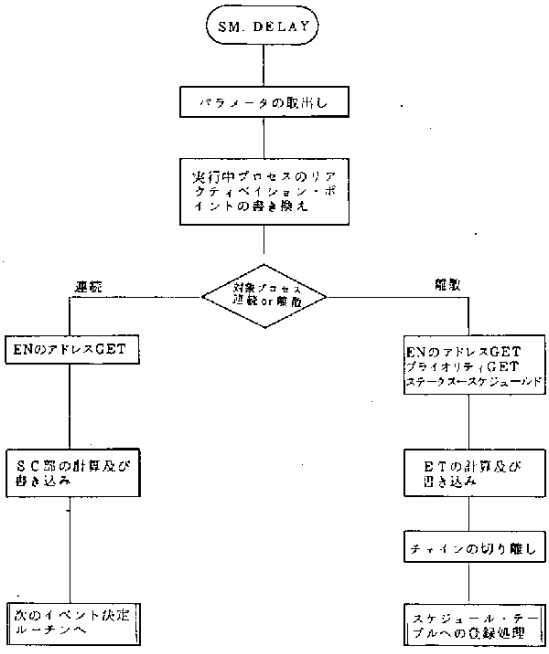
1. 呼び出し手順

SXJ，7	SM．DELAY
NOP	Z，，base

2. パラメータの説明

Z ： 時間の入っているアドレス。

（処理概略フロー）



ここで，以上の３つのステートメントSCHEDULE，RESCHEDULE，DELAY をふり返ってみると次の事がわかる。

(i) 自分（Called プロセス）を自分（Calling プロセス）でスケジュール ……DELAY

(ii) 自分（Called プロセス）を他のプロセス（Calling プロセス）がスケジュール

…… SCHEDULE

(iii) (i), (ii)いずれでもよい …… RESCHEDULE

従って，例えばプロセスＰ１の中で

```
DELAY   30.0 ;
RESCHEDULE   SELF   DELAY  30.0 ;
```

という記述は同じ処理がなされる。

又，プロセスＰ１の中で，

```
SCHEDULE      P 2   DELAY （or AT）  10.0 ;
RESCHEDULE   P 2   DELAY （or AT）  10.0 ;
```

という記述も同じ処理がなされる。

(4) CANCELステートメント

（機能）

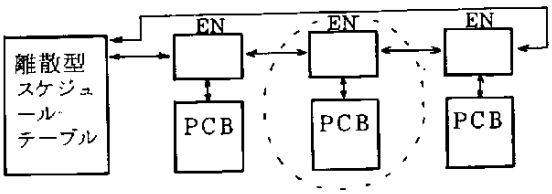
スケジュールド状態にある離散型プロセスをパッシブ状態にする。連続型プロセスの時は，スケジュールド状態のままであるが，その発生が起こらないようにされる。

（一般形式）

CANCEL　＜プロセス＞　；

（処理）

対象となるプロセスが離散型の時は，そのイベント・ノーティスENを探し出し，ENとPCBのリンクを切り離し，ENをフリー・ストレッジに返す。



①切り離し（スケジュール・テーブルのチェイン）
②ENとPCBの切り離し
③ENをフリー・ストレッジに返却

図3－54　CANCELステートメントの操作（離散型）

対象となるプロセスが連続型の時は，SC部に大きな値（SM. TSTOP＋1）をセットし，再びスケジュールされるまでそのプロセスを発生しないようにする。

（ランタイムの呼び出し手順）

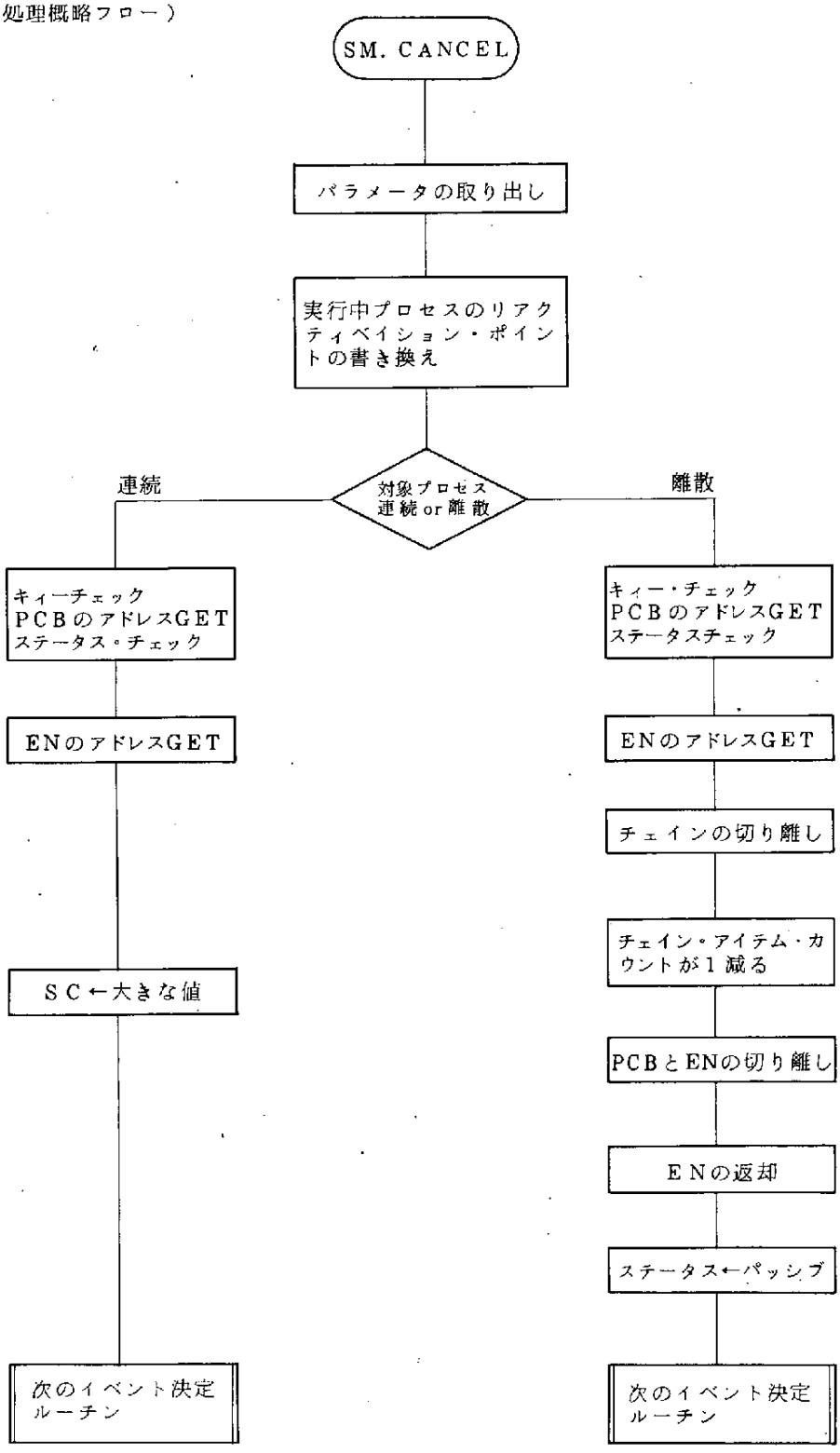
1. 呼び出し手順

```
SXJ, 7          SM. CANCEL
NOP             a , *, base
```

2. パラメータの説明

a ： スケジュールの対象となるプロセス名の入っているareaのアドレス。SELF指定はa番地の内容をzeroにする。

（処理概略フロー）


SM. CANCEL
パラメータの取り出し
実行中プロセスのリアクティベイション・ポイントの書き換え
対象プロセス
連続 or 離散
連続
離散
キィーチェック
PCBのアドレスGET
ステータス・チェック
ENのアドレスGET
SC←大きな値
次のイベント決定ルーチン
キィー・チェック
PCBのアドレスGET
ステータスチェック
ENのアドレスGET
チェインの切り離し
チェイン・アイテム・カウントが1減る
PCBとENの切り離し
ENの返却
ステータス←パッシブ
次のイベント決定ルーチン

(5) TERMINATEステートメント

（機能）

アクティブ，スケジュールド，パッシブのいずれかの状態にあるプロセスをターミネイティド状態にする。

（一般形式）

TERMINATE　〔＜プロセス＞〕　；

（処理）

プロセスを省略した場合にはＳＥＬＦ指定とみなされる。対象となるプロセスが離散型の時は，その対象となるプロセスのＰＣＢを探し，ステータスをターミネイティドにした後，イベント・ノーティスＥＮの有無を確め，ＥＮが有れば（スケジュールド or アクティブ状態の時）離散型スケジュール・テーブルのチェインから切り離し，その後，ＰＣＢとのリンクも切り離してフリー・ストレッジにＥＮを返却する。ＥＮがない時（パッシブ状態）は，ＰＣＢのステータスをターミネイティドにするだけでよい。

対象となるプロセスが連続型の時は，ＥＮの有無を確めない（必ずＥＮはある）点を除けば離散型の処理とほとんど同様である。

（ランタイムの呼び出し手順）

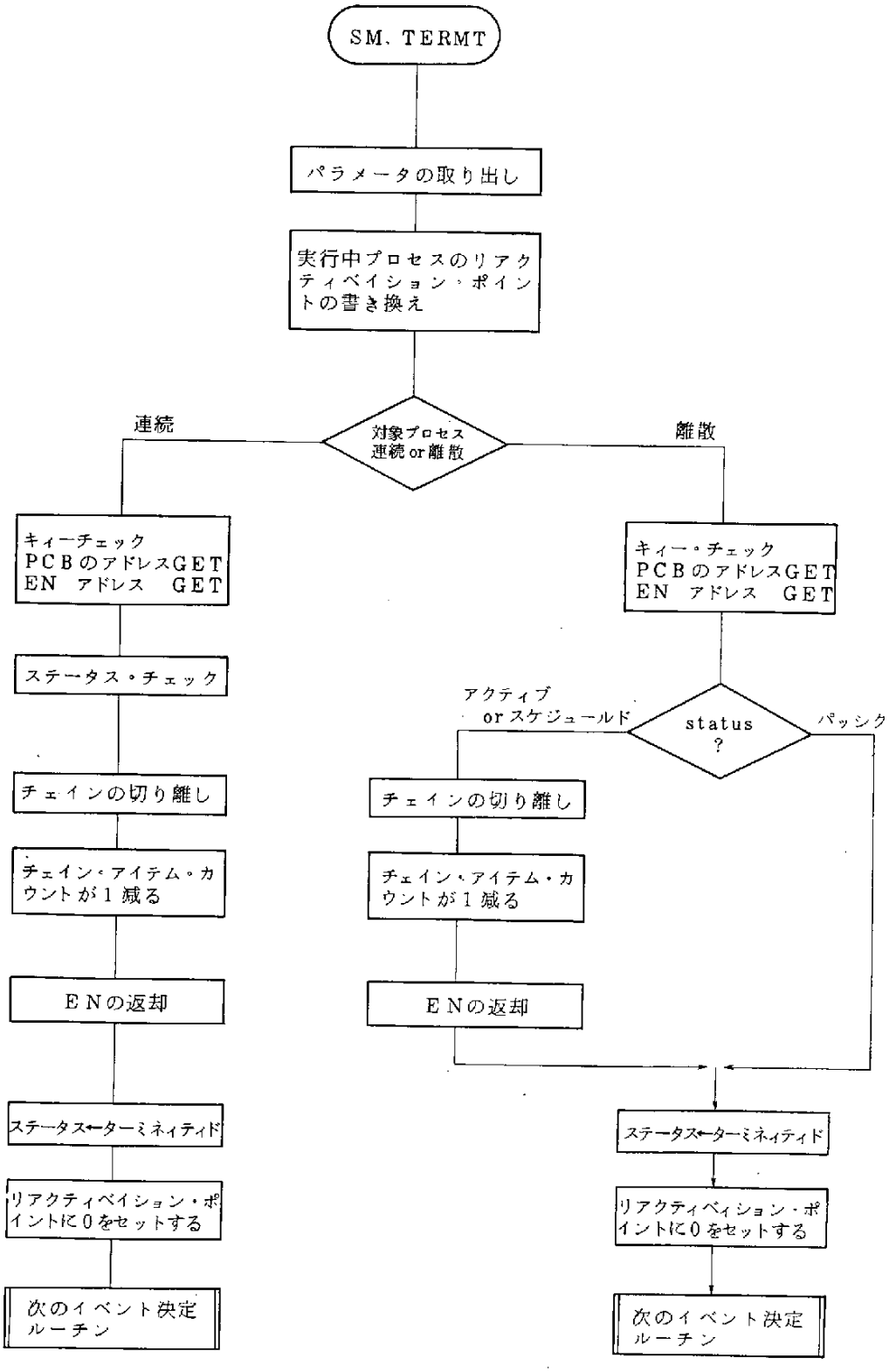
1. 呼び出し手順

SXJ，7	SM. TERMT
NOP	a，*，base

2. パラメータの説明

a ： スケジュールの対象となるプロセス名の入っているareaのアドレス。SELF指定はa番地の内容をzeroにする。

（処理概略フロー）


SM. TERMT
パラメータの取り出し
実行中プロセスのリアクティベイション・ポイントの書き換え
対象プロセス 連続 or 離散
連続
離散
キィーチェック PCBのアドレスGET EN アドレス GET
ステータス・チェック
チェインの切り離し
チェイン・アイテム・カウントが1減る
ENの返却
ステータスターミネィティド
リアクティベイション・ポイントに0をセットする
次のイベント決定ルーチン
キィー・チェック PCBのアドレスGET EN アドレス GET
status ?
アクティブ or スケジュールド
パッシク
チェインの切り離し
チェイン・アイテム・カウントが1減る
ENの返却
ステータスターミネィティド
リアクティベイション・ポイントに0をセットする
次のイベント決定ルーチン

(6) DESTROYステートメント

（機能）

パッシブまたはターミネイティド状態にあるプロセスをシステムの場から消滅する。

（一般形式）

DESTROY　＜プロセス＞　；

（処理）

対象となるプロセスのＰＣＢを探し，これをフリー・ストレッジに返却する。このステートメントは単にＰＣＢに使われているメモリーを他の用途に使える状態（フリー・ストレッジへの返却）にするだけである。なお，対象プロセスは，その時点でグループに属していてはならない。又，対象となるプロセスが離散型，連続型のいずれであっても同様な処理がなされる。

（ランタイムの呼び出し手順）

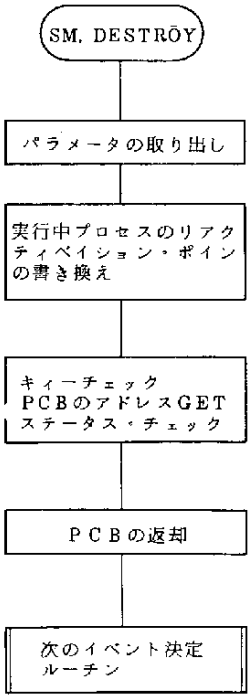
1. 呼び出し手順

SXJ, 7	SM. DESTROY
NOP	a , * , base

2. パラメータの説明

a ： スケジュールの対象となるプロセス名の入っているareaのアドレス。

（処理概略フロー）



(7) WAITステートメント

（機能）

アクティブなプロセスをキューに入れて，パッシブ状態（待ち状態）にする。

（一般形式）

WAIT　->　<キュー>　[, { FIRST / LAST / AFTER <メンバー> / BEFORE <メンバー> }] ;

（処理）

位置指定が省略されるとＬＡＳＴ指定と同様な扱いがされる。現在実行中のプロセス（アクティブなプロセス）のイベント・ノーティスＥＮをスケジュール・テーブルから切り離す。キューに挿入する為にそのＥＮをキュー・メンバー・エレメント（ＱＭＥ）とし，指定されるキューの位置指定される場所に挿入する。キューに挿入する処理は，セットやキューを扱う為のＩＮＳＥＲＴステートメントの処理ルーチンに任せている。ＥＮをＱＭＥとして使用できるように，それらの構造は同じになっている。

キィー	ＰＣＢへのポインター
先行ＱＭＥへのポインター	後続ＱＭＥへのポインター
キューに登録された時刻	

（ＱＭＥの構造）

キィー	ＰＣＢへのポインター
先行ＥＮへのポインター	後続ＥＮへのポインター
ＥＴ or ＳＣ	

（ＥＮの構造）

図３－55　ＱＭＥとＥＮの構造

従って，現在時刻を３ワード目に記入し，指定された場所に挿入し，先行，後続ＱＭＥへのポインター部の書き換えを行なえばよい。もちろん，スケジューリング・ステートメントにおいて，スケジュール・テーブルの該当項目の書き換え，ＰＣＢのステータスやポインターの書き換え等が必要であるのと同様に，キューヘッドの該当項目の書き換え，ＰＣＢのステータスの書き換え（アクティブからパッシブへ）なども必要である。

キューヘッドを示すＩＤ			
Q. MEM	： 現在登録されているメンバー数		
Q. FIRSTM	： 先頭メンバーのポインター	Q. LASTM	： 最終メンバーのポインター
Q. TMEM	： それまでに登録されたメンバーの総数		
Q. MAXWT	： 最大待ち時間		
Q. ZWT	： 待ち時間の総和		
Q. MAXQL	： 最大待ち数		
Q. ZEROE	： 待ちなしで出たメンバー数		
Q. LATIME	： 最新のアクセスタイム		
Q. ZQL	： Q. MEM×Δtの総和		

但し，Δtは現在の時刻－Q. LATIMEの意味

図３－56　キューヘッドの構造

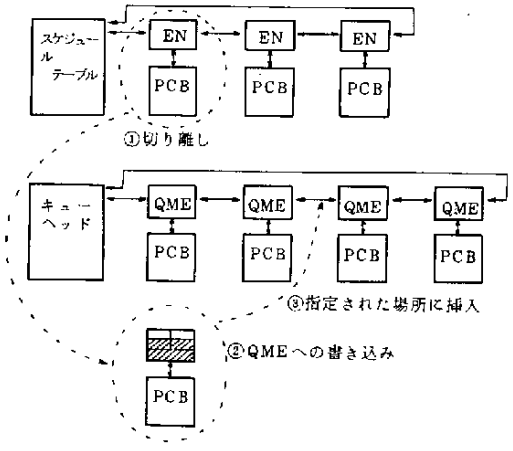


図３－57　ＷＡＩＴステートメントの処理

（ランタイムの呼び出し手順）

1. 呼び出し手順

```
LZI        Positioning code
SXJ, 7     SM. WAIT
NOP        Queue head address, *, base
NOP        β, *, base
```

2. パラメータの説明

Positioning code ： LAST指定　0　（省略時）
FIRST指定　1
BEFORE指定　2
AFTER指定　3

β ： codeが2と3の場合，メンバーの入っている番地，0と1の場合はサブルーチン側で無視する。

(8) WAKEステートメント

（機能）

キューに入って待ち状態（パッシブ）にあるプロセスを現在時刻で実行すべくスケジュールする。

（一般形式）

WAKE　〔＜メンバー＞〕　＜－　＜キュー＞　；

（処理）

WAITステートメントと対をなすステートメントである。メンバーにはプロセス関数が書けるので，キューからどのプロセスをスケジュールするかがダイナミックに指定できる。なお，メンバーを省略すると，キューの先頭メンバーがその対象プロセスとなる。

（ランタイムの呼び出し手順）

1. 呼び出し手順

```
LZI          Schedule code
SXJ, 7       SM. WAKE
NOP          mem , * , base
NOP          Queue head adress , * , base
NOP          α , , base
```

2. パラメータの説明

Schedule code ： AT指定なら　0
DELAY指定なら　1
BEFORE指定なら　2
AFTER指定なら　3

※ PRIOR指定があれば，上記の値に10を加えておく。

mem ： メンバーエレメント（プロセス）名の入っているアドレス。
もし，メンバー指定が省略された場合にはmen番地にzeroを入れておく。

α : Time or Process 名

4. コンバインド・シミュレータの問題

4. コンバインド・シミュレータの問題点

コンバインド・シミュレータとしては，コンバインド型のシミュレーションを行える事はもちろん，離散型だけ，あるいは連続型だけのシミュレーションも行えるシステムになっている事が好ましいと考えられる。実際，ＧＡＳＰ -Ⅳはその様なシステムに作られている。しかし，コンバインド型シミュレーション専用のシステムとした方が，実行効率やメモリー効率が上がる事を考えると，コンバインド型専用のシステムとする事は実用性の面からいって有効であると思われる。

ＳＩＭＢＯＬ－Ⅱはコンバインド型専用のシステムであるが，離散型処理だけ，あるいは連続型処理だけの実行も行えないことはない。ただ，離散型シミュレーションを行う時にも，ステップサイズずつシミュレーションを進めることになるので処理時間がかかり，専用のシミュレータを用いた方が効率的であると言える。ＳＩＭＢＯＬ－Ⅱを前述のような，離散型だけ，連続型だけの実行も効率よく行えるようなシステムとするには，離散型処理だけを行う場合に，離散用スケジュール・テーブルにつながれている先頭イベントから次々に実行していくようなコントロール・ルーチンを作っておき，そこに飛んでいくようにすればよいのではないかと考えている。

コンバインド・シミュレータＳＩＭＢＯＬ－Ⅱは，従来のＳＩＭＢＯＬに連続型プロセスを導入したことにより，いくつかの問題が生じ，一番大きなものとしてスケジューリングの問題がある。これには２つ問題があって，離散型プロセスと連続型プロセスが混在することによって引き起こされる，コンバインド・シミュレータとしての基本的な問題と，連続型プロセスが同時に複数個存在しうることによって引き起こされる問題である。

(1) 離散型プロセスと連続型プロセスが混在することによって起こる問題点

この問題は離散型シミュレーションと連続型シミュレーションではシミュレーション時間進行の管理が本質的に異っている点に起因している。つまり，離散型シミュレーションでは生起するイベントの生起予定時刻に合わせてシミュレーションクロックを動かして行くのであるが，イベントを実行しても時間は進行しない。一方，連続型シミュレーションでは時間の進行が積分器を中心に行われ，１ステップの実行は積分ステップサイズだけ時間の進行を伴う。つまりプロセスを実行すると時間が進んでしまう。そこで，積分ステップサイズの間で起こるイベントを問題にしなくてはならない。

実際には両タイプのプロセスが複数個ずつ存在するはずであるが，この問題を扱うについては離散型プロセスと連続型プロセスが各々１つの場合を考えれば充分である。２つのプロセスが実行される様子を図に表わすと，図４－１のようになる。

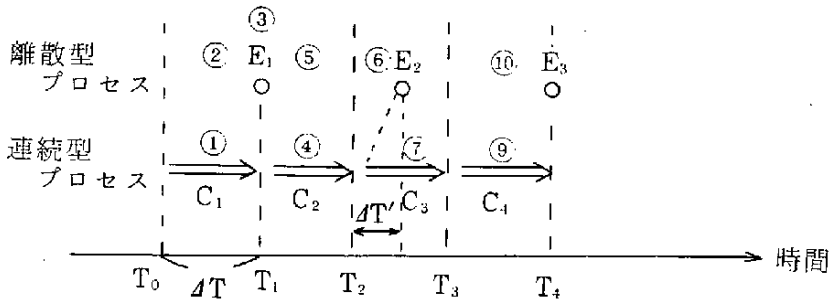


図４－１　離散型プロセスと連続型プロセスの混在における処理順序

２つのプロセスを単に時間の順に従ってうまく実行するだけならあまり問題は起こらず，離散型プロセスのイベントE_1，E_2，E_3などと連続型プロセスの実行サイクルC_1，C_2，C_3，……などについて，実行サイクルの終了時刻とイベントの生起時刻とを比較しながら早く起こる方を実行していけば良い。

問題は，離散型プロセスと連続型プロセスとがお互いにコミュニケートする場合にある。例えば，図中でイベントE_2が連続型プロセスのステータスを参照しようとする時，C_3の途中（即ち，$T_2+\Delta T'$）であるから，本来ならちょうどこの時点での連続型プロセスのステータスを知りたいにもかかわらず連続型プロセスの実行がΔT間隔でしか行われておらず，結局C_2終了時（即ち T_2）でのステータスしか見る事ができない。また，他のプロセスの実行を中断したり再開したりなど，いろいろとプロセス間の関係を複雑に扱う場合にも同様の問題が起きる。

この問題は，ステップサイズを"T_2"の時点で"$\Delta T'$"に変更できる様に，ステップサイズの自動調整ができれば解決できる。しかし，SIMBOL－Ⅱではそれを行っていないので，上記の様なことが起こってもそれを無視している。従って，ユーザは上記の様なことが起こり得るようなシミュレーションを行う場合には，プロセスのステップサイズを注意して決め，なるべくステップとステップの間に両型のプロセスがお互いにコミュニケートする様な処理が起こらないようにしなければならない。

(2)　連続型プロセスの同時現象の問題

この問題はコンバインド・シミュレータだから起こるといった問題ではなく，むしろ複数個の連続型プロセスが同時に実行されるために，これらプロセスの間で起こる問題であり，これも連続型プロセスが積分器を中心に時間が進められることに起因する。

図４－２からほぼ明らかであるが，連続型プロセスを順次実行して行く間で時間のズレが生じている。つまり，実行を終了してしまったプロセスは基準時刻TからΔTだけ進んだ時刻の状態に成っているにもかかわらず，まだ実行を終了していないプロセスは依然，時刻Tの状態のままである。この様に一時的な時間のズレは，プロセス間でお互いに他のプロセスと無関係に進行し

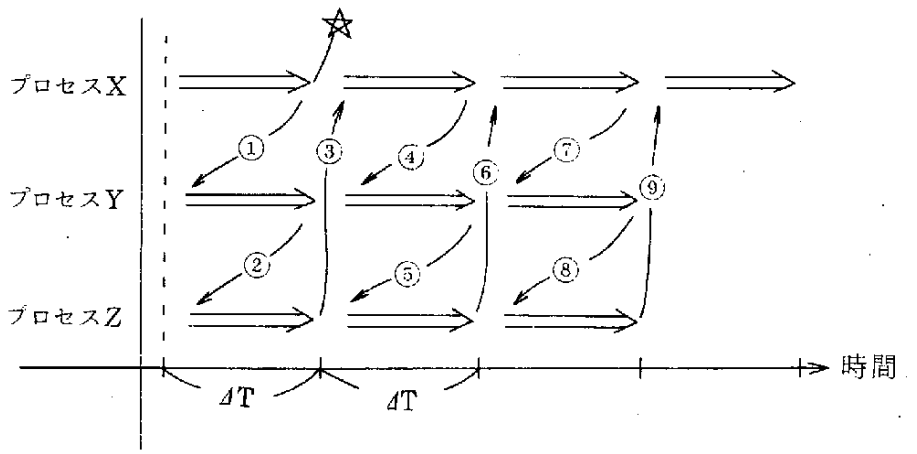


図4－2　複数の連続型プロセスを実行する時の様子

ている場合は特に問題とはならない。しかし，各プロセスがお互いにコミュニケートする場合，つまり他のプロセスのローカルデータをアクセスしたり，実行をコントロールしたりする場合には，ΔTだけ時間を進める間でプロセスの実行順序によってはまるっきり異った結果を生んでしまいかねない。

SIMBOL－Ⅱでは前述のような事が起こっても無視しているが，ローカルデータのアクセスに関してはメモリーがかなり必要となるが，各連続型プロセスのPCBにΔT前の値をセーブしておくエリアを設け，アクセスする側の時刻と同一時刻のものを参照できるようにすれば解決できる。また，連続型プロセスの実行順序は，プライオリティをつけるなどして，連続型スケジュール・テーブルへのプロセスの位置を任意に設定できるのでそれ程問題ではない。しかし，ユーザがスケジューリング・ステートメントの記述やプロセスの連鎖に対し，かなり注意をしなければならない。

昭和50年3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3丁目5番8号
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丁目11番11号
TEL (407) 7316

49－S002