

プロセス制御用マイコンのテスト記述言語

竹中一起、横井玉雄

住友金属工業(株) システムエンジニアリング本部

プロセス制御用マイコンの結合テスト段階以降のテストや調整を自動化することを目的としたテスト記述言語(TPL)を開発した。本システムは、単一計算機内でのテスト機能のみならず、ネットワークを介して結合された他の計算機に対してテスト方案ファイルを自動送信し、かつ解釈・実行を制御(リモート起動及び同期)する機能も有しているので、大規模分散型システムの場合でもシステム全体規模の総括テストが効率的に行えるのが特徴である。

本稿では、まずTPL実行のベースとなる当社のマイコン・セミシステムSRMXについて概説した後、TPLの言語仕様と機能、及び処理系の実現方法について述べた。

Test Programming Language for Process Control Computer

(in Japanese)

Kazuki TAKENAKA and Tamao YOKOI

Sumitomo Metal Industries, Ltd.

1-3, nishinagasu-hondori, Amagasaki, 660, JAPAN

We developed the test programming language(TPL) in order to test real-time software automatically and efficiently on the micro-computer based process control computer. TPL is mainly designed to use at the stage from assembly test to field test. Our TPL can transmit test description files to other computers through network and can control the execution on them, so it can also be applied to the large-scale distributed system easily.

In this paper, we first give the outline of our semi-system software SRMX on which TPL executes, and then describe the language specification, function and implementation(interpreter) of our TPL.

I. はじめに

鉄鋼業界では、他業種に先駆けて古くから製造工程の自動化に取組み、製造工期の短縮や製造コストの合理化に大きな成果を上げてきた。当社では従来、このような製造ラインの計算機制御には国産の制御用ミニコンを導入し、プロセス制御システムを構築してきたが、メーカー間・機種間に互換性がなく、開発や保守の生産性や要員育成の面で大きな問題となっていた。一方、近年のVLSI技術の急激な発達により、マイクロプロセッサが目ざましい進歩をとげ、性能的には従来の制御用ミニコンを遙かにしのぎ、最新のミニコンとも肩を並べるまでになってきた。しかもマイクロプロセッサは、価格が非常に安く、全世界で2～3系列に統合・標準化されている。このような状況の下、当社ではプロセス制御用の計算機（以下「プロコン」と称す）を価格性能比に圧倒的に優れたマイクロコンピュータで標準化するというマイコン戦略（ARMS: Advanced Realtime Industrial Micro-computer of Sumitomo Metal's Standard）を昭和58年にうちたて、同時に国内プロコンメーカーの協力も得てARMS路線を展開してきた。ARMSでは、インテル系マイコン（OSはiRMX, 言語はPL/MまたはFortran）とマルチバスを標準と定め、当初はマイコン1～2台の小規模システムから取組み、数年前には6～10台のマイコンを分散配置した工場全体を制御する大規模システムの構築技術も確立し、最近では新規に導入されるプロコンは、すべてマイコンというまでに定着している。

ところで、鉄鋼プロセス制御のような実時間制御システムでは、機能分割された複数のタスクが、外部からの割込信号やオペレータの要求、あるいは上位や下位に位置する別の計算機からの伝文等により非同期に起動され、お互いに通信しあいながら複雑に絡み合っ

て処理を進めていく。このような大規模かつ複雑な鉄鋼応用システムに対処すべく、国内のプロコンメーカー各社は、長年のシステム経験を基に、OSとアプリケーションソフトの中間に位置する鉄鋼専用セミシステムを開発し、応用システムの開発効率化を図っていたが、マイコンの場合は汎用のリアルタイムOSが提供されるだけの為、大規模システムの場合には結合テストから総合調整・現地調整の工程が長引くと言う事態を招いていた。そこでCPUが8086から80286に移行し、処理速度が高速になりメモリ空間も拡大されたのを機会に、当社に蓄積されたプロコン技術を結集したマイコン用のセミシ

ステムSRMXを独自に開発し、結合テスト以降の効率化に大きな成果を上げてきた。今回、結合テスト以降のテストや調整を更に効率化することを目的として、SRMXをベースにシステムテストを自動化するテスト記述言語（TPL: Test Programming Language）とその処理系を開発したので報告する。今回開発したTPLは、ネットワーク結合された大規模分散型システムの場合も、総括的なテストが行なえるのが特徴である。本稿ではまずSRMXについて概説した後、TPLの機能と実現方法について述べる。

II. SRMX

（1）機能概要

SRMXは、RAS機能の充実による開発期間の短縮と生産性の向上を目的として開発したもので、iRMXと応用ソフトの中間に位置する。一般プログラマはiRMXの存在を全く意識しないでよく、応用ソフトの製作に専念できる。SRMXの開発にあたっては、特に以下の機能の実現に重点を置いた。

①システム資源の利用方法の簡略化

iRMXでは、タスクやセマフォ等のシステム資源を、ユーザが動的に生成し管理する為、プログラムが複雑になり、単体テストもやりづらい。そこでこれらシステム資源は、イニシャル時にシステムが自動的に生成するようにし、応用ソフトからは資源に割り付けた一連の番号によりアクセス可能とする。

ファイルやデバイスについては、イニシャル時にシステムがアタッチ、オープンし、応用ソフトからは読み書き命令のみで利用可能とする。さらにデバイスについては、通信プロトコルが異なっても統一したインターフェイスを提供する。

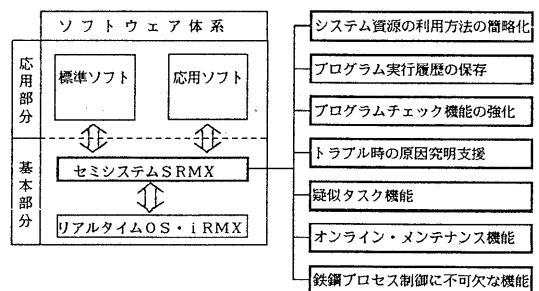


図1 SRMXの位置づけと機能

②プログラムの実行履歴の保存

マルチタスクシステムでは、複数のタスクが通信しあいながら並列に動作する。このようなシステムでトラブルが発生した場合、どのタスクがどういう順番で動作したかという情報が得られないと、原因の究明に非常に時間がかかる。そこで応用ソフトがSRMXに対して命令を発行する度に、実行履歴を保存するようにする。なお履歴の保存の可否は、タスク単位に指定できる。これにより挙動が不審なタスクだけに焦点を当てることができる他、オーバーヘッドの軽減にも有効である。

③プログラムチェック機能の強化

システムコールのパラメータチェックを強化することにより、複雑なトラブルを未然に防止する。またメッセージ受信時に於ける、受信バッファサイズがメッセージ長より短いことによるメモリ破壊や、何らかの不具合により動作停止中のタスクに対し、メッセージを送り続けることにより発生するメモリの欠乏等を、システムレベルで検知し、不具合の早期発見を目指す。

④トラブル時の原因究明支援機能の充実

システムコールの実行履歴や応用ソフトウェアのトレースを可能とし、リセット後でも前回の実行状況を参照可能とする。

⑤疑似タスク機能

端末を1つのタスクとみなして、任意のタスクを起動したりメッセージを受け取ったりできる機能で、タスク単位のテストや漸増的なプログラム開発を可能とする。

⑥資源やデータのオンラインメンテナンス機能

システム資源やファイル内容を、オンライン時にも参照・更新可能とし、プログラム開発やテストの効率化を図る。

⑦鉄鋼プロセス制御に不可欠な機能の提供

ファイル関係では、レコード単位のファイル入出力やレコード単位・ファイル単位の排他制御、インデックス付ファイル等。タスク関係では、時刻指定や一定周期毎のタスク起動、他タスクの実行禁止等、鉄鋼プロセス制御に不可欠な機能を数多く提供することにより応用ソフトの負荷を軽減する。

(2) サブシステム

SRMXは、以下の5つのサブシステムから成る。

①タスク管理サブシステム

タスク起動・タスク間通信、タスク状態管理、実時刻管理等を司る。SRMXでは、相手方のタスク番号と機能コード、及び可変長のメッセージデータにより、タスク起動やタスク間通信を行なう。タスク番号は1ワード長で、そのうち上位1バイトはCPU番号を表す。応用ソフトからは、起動先のタスクが自CPU内の場合も、他CPU内の場合も全く同じ処理を実行すればよく、起動先タスクが他CPU内の場合は、SRMXが自動的にネットワークにメッセージを送り出す。

②ファイル管理サブシステム

IRMXの管理下で、レコード単位の入出力やファイル／レコード単位の排他制御を実施する。ファイルの種類としては、「ダイレクトアクセス」「インデックス付き」「サイクリック」の3種類をサポートする。

③デバイス管理サブシステム

各種の標準端末への入出力やプリンタへの出力と通信をサポートする。通信方式としては、当社標準の中規模ネットワークS-NETや、BSC、レベル2B等、各種の通信プロトコルをサポートしており、通信コントローラボードとともに、ビルディング・ブロック方式で、所望のシステムを構成できる。

④プロセスデータ入出力管理サブシステム

応用ソフトとプロセス信号入出力装置間のインターフェイスを総括的に管理し、応用ソフトのハードウェアからの独立性を高める。また各種ユーティリティ機能の提供により、ソフトウェアテストやハードウェア調整の効率化を図る。

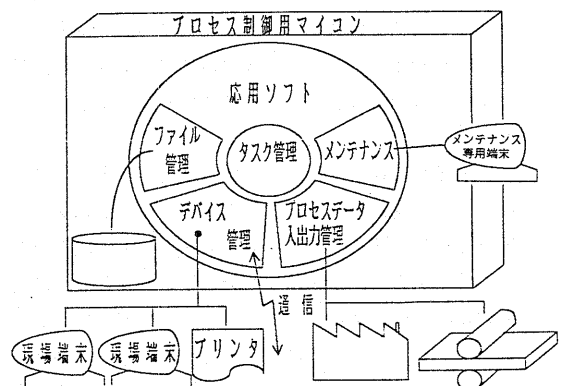


図2 SRMXの構成

オペレータの要求により、タスクやファイルの状態表示、疑似タスク機能、タスク実行履歴の表示、ファイルやテーブルの参照・更新等を行なう。本サブシステムはTPLと深く結び付いているので、次節で詳しく述べる。

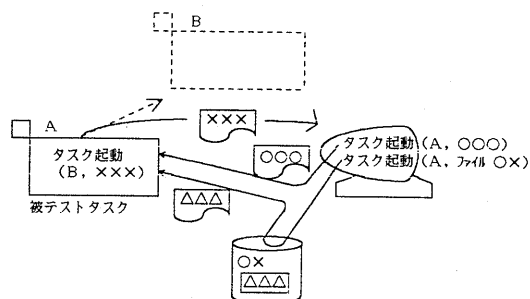
メンテナンス専用の端末から対話型で実行でき、システムテストやトラブル時の原因究明を強力に支援する機能である。ここでは、メンテナンス機能のうち、TPLと特に関係の深い4つの機能について述べる。

各タスクが、現在どのような状態(Running, Waiting, Suspended等)なのか、あるいはメッセージキュー(メールボックス)の中には、何個のメッセージが未処理状態でたまっているか、そのメッセージ内容はどういうものかを表示する。また時刻指定や周期指定によるタスク起動用のメッセージを管理する内部テーブルも表示できる。さらにイベント待ちをしているバターンの待合せ内容、待合せ条件(AND, OR等)や、イベントの現在値等を表示したり、あるいはファイルやタスクの排他制御情報を表示したり等、各種の内部管理情報の表示が行える。

動作中のタスクを停止させたり再開させたり、タスクの優先度を変更したり、あるいはデッドロックに陥ったタスクを回復させたり、イベントを発生させたり取消したりして、テストを効率的に支援する。

メンテナンス専用端末を1つのタスクとみなして、任意のタスクを起動したり、タスクからメッセージを受け取る機能で、タスク単体の結合テストからシステム全体規模の結合テストに至るまでを強力に支援するものである。

メンテナンス端末からメッセージ内容をキーインするか、あるいは、メッセージを格納したファイル番号とレコード番号を対話形式で指定することにより、メッセージが生成されタスクを起動することができる。特に後者のメッセージ生成法は、テスト方案に基づき一連のメッセージをファイル化しておけば、何度でも繰り返しテストできるため、非常に強力なテスト手段となる。



また、プログラムが全部出来上がった後の総合調整時においても、SRMXでは、端末や通信、プロセス信号入出力等、すべてのI/Oは、SRMXの入出力管理タスクがI/Oをハンドリングした後、アプリケーションの指定タスクへ入力をメッセージ型式で振り分ける構造をとっているため、現場端末からのオペレータ入力や上位計算機からの伝文、プロセス割込信号等によるタスク起動メッセージをファイル化しておけば、実際の入出力がなくても、応用ソフトレベルでは実際のシステム運用時と同様のテストを行なうことが可能である。

高度なプロセス制御システムでは、制御対象の挙動を数式化した制御モデルを組み込み制御している。このような制御モデルは、理論式あるいは経験式により、式の骨格は固まっているが、パラメータは当初確定していないものが多く、システムを運用しながら個別システムにあわせてパラメータをチューニングしていく。また制御モデル以外のソフトウェアについても、しばしば変わる部分については、可変部をデータ域におき、プログラムは固定するのが通例である。

SRMXでは、ファイル内あるいはテーブル上に置かれたこれらのパラメータやデータを容易に参照・更新可能なよう、フルスクリーン型式とコマンド型式の2種類のメンテナンス方法を提供している。

III. TPLの機能

(1) 開発の背景

SRMXではメンテナンス機能の充実により、漸増的なプログラム開発やシステム全体規模の結合テストが比較的容易に行えるようになり、テストの効率は大きく向上したが、それでもなお、オペレータが全てのメンテナンス・コマンドを対話型で入力しなければいけないとか、タイミングにクリティカルなテストは難しい等の問題点があった。そこで今回、これらの問題点を一掃し、結合テスト以降のテストや調整をより一層効率化することを目的として、メンテナンス専用端末から対話的に行っているプログラムテストや管理情報/ファイルのメンテナンス操作を、自動的に連続実行するためのTPL言語とその処理系を開発した。

(2) TPL文法

ARMSではPL/MとFortranを標準言語と定めているが、科学技術用大型計算機上で開発され移植された制御モデル以外は、実行効率及び記述力を重視して全てインテル社のシステム記述言語PL/Mでプログラム開発している。そこで今回のTPLも、プログラマが熟練しているという意味からPL/Mライクなシンタックスとした。但し、実行コマンドのみをファイル化し、一括実行するバッチ型式の記述・実行を可能とするため、最外側のdoブロック（モジュール名：DO；～END；）は省略することとした。

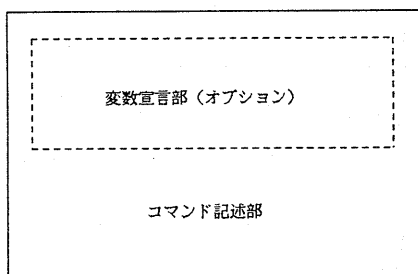


図4 TPLファイルの基本構成

①変数宣言

コマンド記述部でWORK変数を使用する場合、あるいは主記憶上のテーブルを参照する場合に宣言する。

a. WORK変数

BYTE, WORD, INTEGER, REAL, DWORD 等の単純データ型のみサポートする。初期値の設定も可能である。

配列型や構造型は、TPL実行の制御変数やWORK変数としては、通常必要ないと判断し、今回は対象外とした。

b. システムテーブル参照（絶対アドレスも含む）

メンテナンス専用端末からオペレータが対話型でテストを行う場合は、コマンドを実行する度にタスクの挙動を確認し、その結果に応じて逐次テストを進めていく。これらの情報は全てシステムの内部管理テーブルに格納されているので、このテーブルを参照する手段を提供する必要がある。またプログラムがアクセスするアドレスは論理番地化されているが、複数台のCPUからなるマルチCPUシステムでは、CPU間のデータ交換用として共有メモリを使用することから、絶対番地の内容を直接参照・更新する機能も不可欠となる。そこでTPLでは、システムの内部管理テーブルや絶対番地にアクセスする手段として、変数のAT属性宣言を提供している。

変数宣言の例を図5に示す。イはI, JというWORK変数をBYTE型で宣言したもの、ロはKというWORD型のWORK変数を初期値10で、またハはTBLTCBという名のテーブルをAというBYTE型の配列で、ニは70000H番地からをBという同じくBYTE型の配列でアクセスするための宣言である。

```
DECLARE (I, J) BYTE ,      .... イ
        K      WORD  INITIAL (10);  .... ロ
DECLARE A (128) BYTE AT (@TBLTCB),  .... ハ
        B (32)  BYTE AT (70000H);  .... ニ
```

図5 変数の宣言例

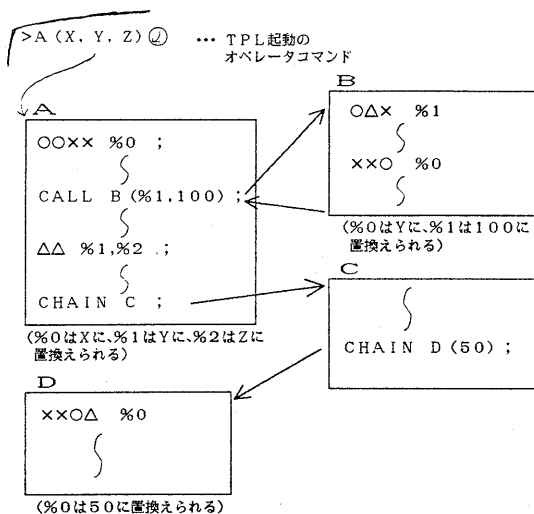
②コマンド記述

単純なコマンド実行文、代入文の他、do while文、do case文、do 繰返し文、単純do文、if～then～else～等、PL/M言語と同一の制御構造を提供している。

各表現部に於ては、算術演算、論理演算、関係演算をサポートする。またPL/Mと同様、厳密なタイプチェックを行う。型変換には組込関数を使用する。

TPLファイルのモジュール化あるいはライブラリ化を可能とするため、'CALL TPLファイル名'や、'CHAIN TPLファイル名'により、TPL実行のファイルネスティ

ングや実行ファイルの移行を実現している。CALLあるいはCHAINによるTPL実行ファイル移行時のパラメータ引渡し、あるいはTPLを起動するオペレータコマンドからのパラメータ引渡し用としてフォーマルパラメータをサポートしている。これはTPLファイル内で、引渡されたパラメータを使用する箇所に、「%n」(n:0～9)の型でパラメータを記述しておけば、TPL実行時、%0,%1,%2…の順で、実際のパラメータと置き換えられて解釈・実行される。パラメータが省略された場合は、ヌル・ストリングに置き換えられる。



(3) TPLコマンド

メンテナンス専用端末から対話型で実行するメンテナンス機能の殆どの操作を、TPLコマンドとして利用可能としている。現在サポートしているTPLコマンドの分類と機能・コマンド数等を、表1にまとめる。

(4) 分散型システムへの対応

中・大規模プロセス制御システムでは、各機能単位に計算機を配置し、それらをネットワークで結合した分散型システムの形態をとるのが通例である。このような分散型システムの場合でも、システム全体規模の総括テストが効率的に行えるよう、今回開発したTPLシステムでは、単一CPU内でのテスト機能(TPL実行)のみならず、ネットワークを介して結合された他CPUに対してTPLファイルを送信しつつTPL実行を制御(リモート起動及び同期)する機能も有している〔他CPU内のタスクに対するタスク起動は、SRMXのシステムコールレベルで実現済〕。

これは図7に示すように、システム全体を統括する統括CPU内で、各個別CPUのテスト方案をTPL文法に従って記述・一括管理し、更にシステム全体としてどのようにテストを進めるかというテスト・シナリオをTPLファイル化して実行すれば、統括CPUから各CPUに対してTPLファイルが自動的に送信され、TPLがリモート起動される。この機能を利用すれば、統括CPUはあたかもオーケストラの指揮者の如く振舞い、システム全体規模のテストでも効率的に進めることができる。

表1 TPLコマンド一覧

分類		主なコマンド機能	コマンド数
実行制御	実行制御	タイミング制御、中断(外部からの再起動待) TPLファイルネスティング、TPLファイル移行	7
	他CPUリンク	他CPUでのTPL起動、 TPLプログラム間での同期	2
タスク管理	内部管理 情報表示	タスク状態表示、リアルタイムタスク管理情報表示、 イベント管理・排他制御管理情報の表示	5
	タスク起動	非同期型・同期型タスク起動 時刻指定・遅延時間後・周期型タスク起動	8
	システム・メンテナンス	イベント発生・取消、タスクの停止・再開、優先度変更	10
	実行履歴 の保存	タスク単位の実行履歴の保存可否、トレースの可否	4
ファイル管理	ファイル内容 メンテナンス	ファイル内容の表示・変更、プリント出力、ファイル間のデモタビ- (表示型式は、10進、16進、Eタビ、ASCII、EBCDIC等豊富に準備)	7
	ファイル管理 内容メンテナンス	ファイルのオプション・加- ファイルの管理情報の表示	5

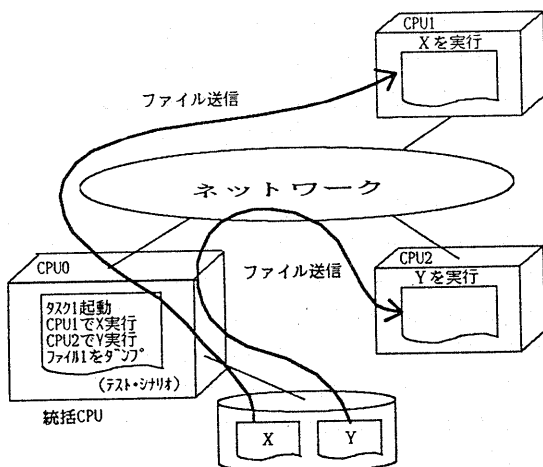


図7 TPLのリモート起動

他CPUへのTPLファイルの送信とTPLのリモート起動は、TESTコマンドで一括して実行する。このとき、他CPU内でのTPL実行が終了するまで起動元TPLが待機するか（同期型）、あるいはリモート起動した後、お互いが並列にTPLを実行するか（非同同期型）を指定できる。非同同期型で起動した場合は、起動元側がWAITコマンドを発行して相手CPUのTPLが終了するのを待つ。またTESTコマンドで、TPLファイル名の頭に:REMOTE:とつけると、ファイルの送信は行わずリモート起動のみを実施する。:REMOTE:指定の有無はフォーマル・パラメータを利用すれば簡単に切替えられるので、テスト方案を変更したときのみファイル送信を行うように運用すれば、ファイル送信に伴うオーバーヘッドを削減できる。

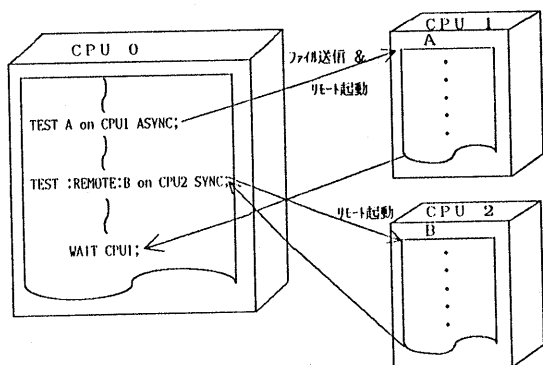


図8 TPLのリモート起動と同期

(5) TPLの運用

TPLは、現場オペレータ端末あるいはメンテナンス専用端末から

>TPLファイル名 [(パラメータ)] ②

と入力することにより起動される。またTPLの実行を中止したい場合には

>STOP ②

を、PAUSEコマンドにより中断しているTPLファイルを再開させる場合には

>CONT ②

を入力する。なおリモート起動はTPLファイル中からのみ起動可能で、端末からの直接起動はサポートしていない。

図9に、TPLの実行例を示す。実行結果は通常、システムプリンタに出力される

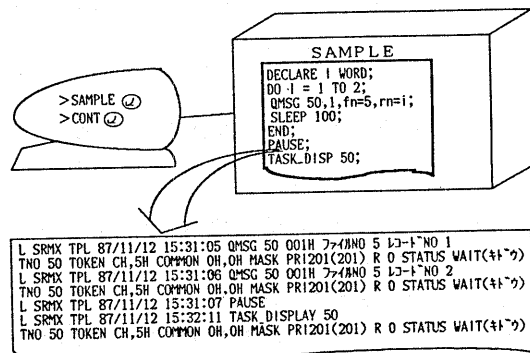


図9 TPLの実行例

IV. TPL処理系の実現

(1) TPLインタプリタ

テスト方案の変更やテストバスのバリエーションに柔軟かつ迅速に対応できるように、TPLの処理系はインタプリタ型とした。従って、TPLファイルの内容を変更した場合でも、即座にテストを実行できる。

(2) UNIXツールによるインタプリタの実現

TPLインタプリタの開発にあたっては、開発効率を上げるため、字句解析と構文解析にはUNIXのコンパイラ生成ツールlexとyaccを使用し、意味解析及びインタプリタの実行処理はyaccのactionとして記述することにした。変数は、高速検索が可能なよう開番地法のハッシュで管理した。またdo while文、do 繰返し文、

if文等の制御構造を実現するため、インタプリタ実行用の制御スタックを設け、制御構造に入ったときには図10に示すスタックフレームをpushし、制御構造から抜け出した時点でpopする処理を、各制御構造パターンのaction部に記述した。スタックフレームの「タイプ」は制御構造の種類を表す。また「実行フラグ」は、実際に処理を実行するか、あるいはシンタックスチェックのみを行うかを識別するために使用する。「パラメータ」部は、制御構造の種類に応じて取扱いが異なる。

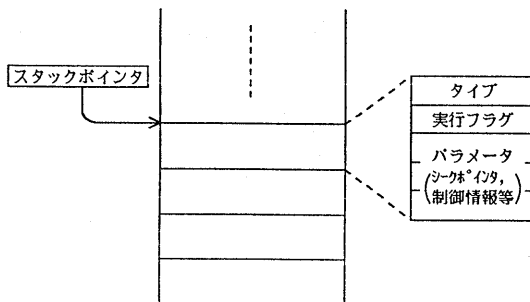


図10 インタプリタ実行用の制御スタック

ところで、lex及びyaccは元来コンパイラ生成ツールであり、従って解析対象のファイルを先頭から順に一度スキャンするだけであるのに対して、インタプリタではdo while文とdo 繰返し文については、TPLファイルの読込に関して後戻りの必要がある。そこで、lexの読込みルーチンinput()を、現時点（正確にはある制御構造の直前）のファイルのシークポイントを常に参照できるように書換え、上記2つの制御構造に入ったときには、現時点のシークポイントをスタックに積んでおく。その後、当該制御構造の終了をyaccが検出したところで、スタックフレームをpopすると同時に、TPLファイルのシークポイントを、当該制御構造の入口の位置（スタックに保持）に戻すという方法でこれらの制御構造を実現した。その他の制御構造についても、スタック内のパラメータ部に制御情報を保持し、適宜利用することにより、インタプリタの基本動作を実現した。

本インタプリタは、システム内に組み込まれ（常駐し）、オンライン使用に供される。またテスト方案によっては24時間以上連続動作するケースも十分考えられる。従って、文字列の処理には通常多量のヒープ

（自由メモリ）を必要とするが、ヒープを使い尽くすことによるシステムダウンを確実に回避すべく、字句解析や構文解析で使用するメモリ領域は、あらかじめ固定域として確保しておき、この領域をWORKエリアとしてマッピングすることにより対処した。

(3) エラー処理

TPLで検出するエラーには、文法エラーと実行時エラーの2種類がある。本処理系ではTPLファイル内の文法エラーの早期洗出しを図るため、文法エラーを検出した場合はその時点でコマンドの実行を中断するが、シンタックスチェックはファイルの最後まで続行することにした。実行時エラーの場合は、そのままコマンド実行を継続する。スタックフレームの「実行フラグ」は、これらの管理に使用している。

V. おわりに

本稿では、プロセス制御用マイコンの結合テスト段階以降のテストや調整を自動化・効率化することを目的として開発したTPLシステムについて、まずそのベースとなるセミシステムSRMXについて概説した後、TPLの言語仕様と機能、及び処理系の実現方法について述べた。

本システムは、中規模のプロセス制御システムの開発に既に適用し、その有効性を確認している。

謝辞

本システムの機能につき御討論いただき、有益な助言を賜った当社と歌山製鉄所計測制御システム室の川畑友明、浦本太郎の両氏に深く感謝します。