ソフトウェア工学 51－3
（１９８６．１１．２６）

インタプリタ方式によるプログラム解析・評価システム

羽田野 尚登、 永田 淳次、 武内 惇

沖電気工業株式会社 情報処理事業本部 ソフトウェアセンタ

本稿では、クロス開発環境のホストマシン上で、ターゲットマシンのプログラムを実行させ、その動作特性を評価・解析するためのツール：ＰＩ２システムについて述べる。

ＰＩ２は、ソースプログラムをインタプリティブに実行する対話型デバッガの標準的な機能の他に系統的に効率良くテストを進めるための、カバレッジの測定機能、アサーションのチェック機能、再テスト自動化機能等を有する。

更に、組込み型システムではプログラムの一部をアセンブラ言語で記述することがあるため、ＰＩ２は、アセンブラプログラムをＰＬ／Ｍプログラムと連結し、同様に実行する機能（ＯＭＦ実行機能）を有する。

本稿では、このＯＭＦ実行機能を中心に、ＰＩ２システムの機能について述べる。

PROGRAM ANALYSIS SYSTEM
BASED ON SOURCE CODE INTERPRETER (in japanese)

Hisato HATANO, Junji NAGATA, Atsushi TAKEUCHI

(Information processing division, Software Center,
OKI electric industry co.
1-16-6 chuou, warabi-shi, 335, japan)

PI2 is an interpretive execution system of PL/M source programs for a target microprocessor system.

PI2 supports test coverage, assertion checking, etc., in addition to functions provided by ordinary cross development environments.

This paper describes functions in PI2, focusing on an execution function which executes assembly object modules and PL/M source programs, binding them dynamically.

1．　はじめに

近年、マイクロプロセッサは機能及び性能の高度化とコストの低下によりその適用範囲は拡大の一途をたどっている。その中で金融端末などの情報処理装置，機械やプラントの制御装置などのようにハードウエア装置の中に組み込まれて使用されるもの（組込み型システム）も多い。組込み型システムでは装置機能の複雑化に伴いそれに要求されるプログラムは大規模化、複雑化の傾向にある。（金融端末では，１ＭＢｙｔｅのメモリを実装しているものもある．）

ターゲットマシン上での組込み型システムのプログラム開発では、次の点で問題がある。

・テスト／デバッグ用の入出力装置が接続されていない（できない）。
・テスト／デバッグ用のメモリ領域を確保しにくい。
・開発段階でのプログラム管理を行なうためのファイル容量が少ない。
・ＯＳ等の基本ソフトウエアの構成がシステム毎に違い，高級なプログラム開発用ツールを装備できない。

このため一般には、スーパミニコンピュータもしくは大型コンピュータ上に別途プログラム開発システムが構築され使用されている。この環境のことをクロス開発環境と呼び、プログラム開発用のコンピュータのことをホストマシン，実行対象のコンピュータのことをターゲットマシンと呼ぶ。

クロス開発環境には、次の利点がある。

・大容量の補助記憶装置が利用可能－－－ファイル等の管理が容易
・高速の印字装置が利用可能－－－必要な情報を即時に入手できる
・高速のプロセッサが利用可能－－－システム構築が高速で行なえる
・豊富なツールが利用可能－－－ホストマシン上のツールが利用できる

しかし、クロス開発環境では、コーディングしたプログラムの動作確認までに時間がかかるという欠点を持つ。プログラムはホストマシン上でコンパイル、リンクされ、ターゲットマシンに転送される。その後テスト・デバッグが可能になる。

これらの環境を改善させるために、ホストマシン上でデバッガを用意し、ターゲットマシンにプログラムを転送することなく、ホストマシン上でテスト・デバッグ作業を可能にした。

クロス開発環境でのホストマシン上のデバッガは、ホストマシン用のデバッガと比較して、次の機能が必要であろう。

(1) ターゲットマシンの環境
　全てのプログラムを用意して動作させることは難しいため、ターゲットマシンの環境をホストマシン上に構築　することを支援する機能が望まれる。
(2) テスト支援
　ターゲットマシンでのテスト作業を極力減少させるために、再テスト支援、網羅率の測定等の機能が望まれる　。
(3) 実行モニタ
　ターゲットマシンでの、障害の原因解析は難しい面が多いため、予想される問題点を可能な限り抽出しておくことが望まれる。
(4) 対象範囲
　プログラムは、高級言語だけでなくアセンブラでも記述されるため、全ての言語を対象にしたものが望まれる。

ホストマシン上のデバッガの実現方式には、ソースインタプリタ方式、ロードモジュール実行方式の二種類ある。ソースインタプリタ方式は、ソースプログラムを完全に表示できる等プログラマの思考にそくした表現方法をとることができるという長所がある。また、ターゲットマシンのプロセッサに依存しないという特徴もある。ロードモジュール実行方式は、ソースインタプリタ方式と比較して高速で実行できるという長所がある。我々はプログラマの思考にあわせることができるということを重要視し、ソースインタプリタ方式を採用した。またプログラムだけでは、言語毎にデバッガが必要となるため，オブジェクトレベルの実行機能（ＯＭＦ実行機能）を備えることとした。このように、ホストマシン上のデバッガはターゲットマシンに依存しない機能と依存する機能とを融合したものとした。

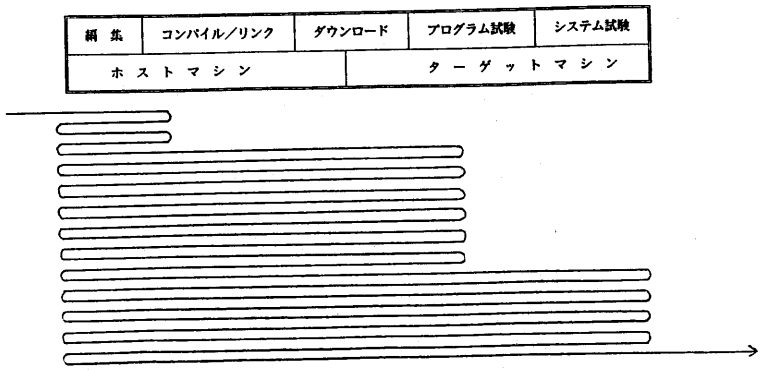
本報告では、ホストマシン上のデバッガであるＰＩ２の機能概要を示し、特にＯＭＦ実行機能について詳述する。

2．設計方針

ＰＩ２［１］はＰＬ／Ｍ言語で記述されたプログラムを実行することが基本的な機能である。ＰＬ／Ｍ言語はシステム記述言語であり、ＰＬ／Ｉ言語のサブセットである。

(1) 作業モデル

クロス開発環境では、プログラムの作成（修正）をしてから、その動作確認までのサイクル（図－１参照）が長いという欠点を持つ。



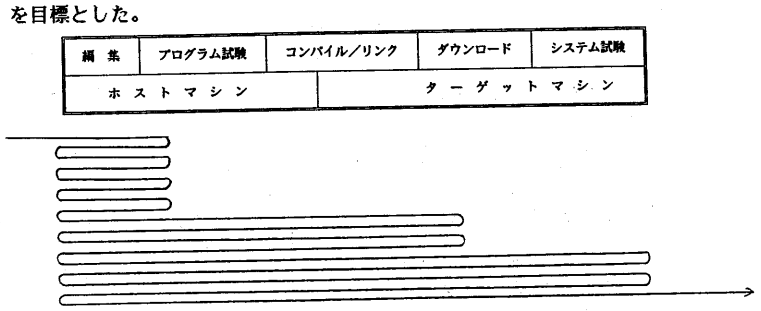
図－１　動作確認までのサイクル

そこでＰＩ２では

① ホストマシンでの動作確認を充実し、ダウンロードターゲットマシンでの動作確認を軽減する。

② ホストマシン上での動作確認、プログラム修正のサイクルを早めること。

を目標とした。



図－２　ＰＩ２利用時の動作確認までのサイクル

(2) ＰＩ２への要求機能

「(1)の作業モデル」の改善を図るため、一般にデバッガに要求される機能を［４］に加えてＰＩ２では、クロス開発環境のデバッガとしてターゲットマシンの環境、テスト評価、マルチ言語のサポートなどの実現を考慮し、表－１に示す要求機能を設定した。

機能項目	概要
プログラム実行	ターゲットマシンのプログラムをホストマシン上で実行する。
実行モニタ	実行状態（プロシジャの活性状態、プログラム履歴等）を監視、表示する。
テスト支援	網羅率の測定、アサーションのチェック等のテストを行う。
デバッグ支援	シンボリックにデータの参照・変更を行う。
ドライバ／スタブ	ドライバ・スタブを簡易に生成し、テスト環境生成を簡単化する。
ユーザ・インタフェイス	馴じみやすいマンマシンインタフェイスの提供 繰り返し作業の自動化
対象モジュール	コーディングされたプログラムの全てを、テスト・デバッグ対象とできる。
実行環境	ターゲットマシンと同様の環境をホストマシン上に構築する。

表－１　ＰＩ２への要求機能一覧表

(3) インタプリタ実行方式の採用

ＰＩ２の実現方式には、ソースインタプリタ方式［２］、［３］とロードモジュール実行方式の二種類が考えられる。それぞれの特徴を表－２に示す。

項目	ソース・インタプリタ方式		ロードモジュール実行方式	
プログラマの思考に沿う。	実行対象がソースプログラムであるため、 ・シンボリックなデータの扱い ・注釈も含め完全なソースプログラムの表示。 ができ、コーディングしたプログラムとのテストデバッグが実施できる。	○	アドレスをシンボルとして扱うことも可能であるが、完全にシンボリックに扱うことはできない。 またデータを１６進表示にたよることになる。	×
対象範囲	言語仕様が変わらない限りターゲットマシンのプロセッサに依存しない。	○	ターゲットマシンのプロセッサに依存するため、プロセッサ毎に開発が必要となる。	×
パフォーマンス	ソースプログラムの解釈が必要とされる。	×	コードの解釈は単純である。	△
リアルタイム性	ホストマシン上のためリアルタイム性には欠ける。	×	ホストマシン上のためリアルタイム性には欠ける。	×

表－２　ホストマシンでの実行方式

我々は、「プログラム思考に沿う」ことに注目し、ソースインタプリタ方式を採用することとした。

3．PI2

3．1　機能概要

ＰＩ２の機能一覧を図－３に示す。

(1) プログラム実行機能

ＰＩ２の基本的な機能はＰＬ／Ｍソースプログラムを入力し、それを実行させることである。実行形態としては、連続実行、ステップ実行等があり、デバッグの状況に応じてその種類を選択することができる。

(2) プログラム実行モニタ機能／デバッグ支援機能

プログラムの実行を制御することができ、コマンド内からプロシジャ呼出しや強制的に分岐させる機能をもつ。ブレークポイントやウォッチポイント等の実行モニタ機能とを組合せることによって柔軟なプログラム実行を行うことができる。ブレークポイントやウォッチポイントでは、その条件が成立したときの動作を指定することができ、中断の変わりにある動作を行わせることができる。

(3) テスト支援

テスト作業を支援するために，Ｃ０カバレッジ、Ｃ１カバレッジを収集する機能を持つ。オペレーションログを利用することにより、再テストも容易にしている。

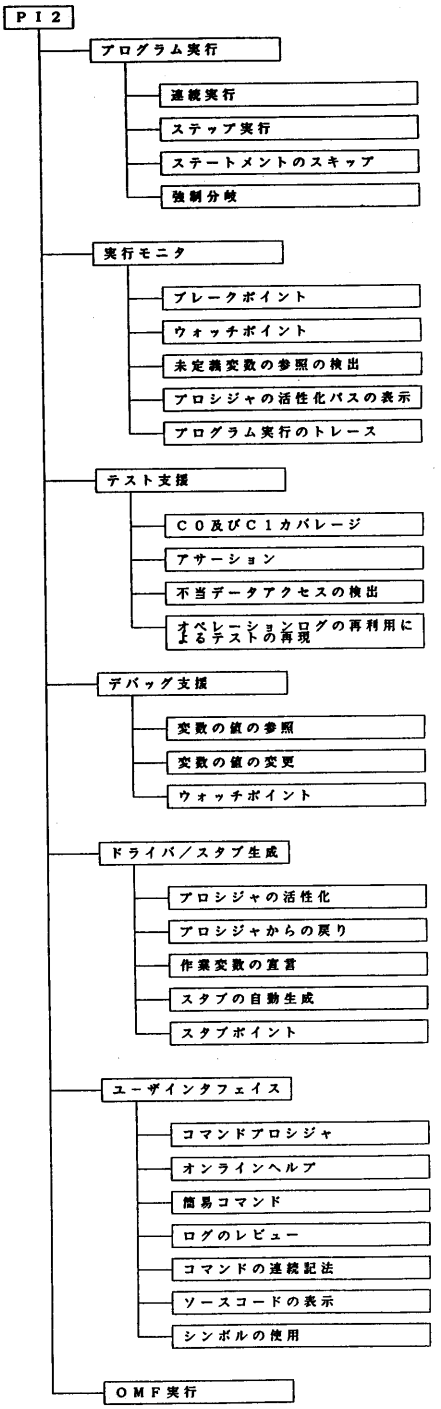
プログラムの検証のためには、アサーションチェック機能を利用することができる。注釈もしくはコマンドの中に範囲等のアサーションを定義することができる。

(4) ドライバ／スタブ生成機能

実行プログラムに必要なドライバやスタブは、ＰＩ２のコマンドで定義することができるため、新たにプログラミングする必要がない。

(5) ユーザインタフェイス

ＰＩ２のプログラム表示では、ソースファイルをそのまま表示するため、コーディングしたソースコード（コメントも含めて）を見ながら、作業を進めることができる。また、コマンドには、プログラム内のシンボル名を用いることができ、その型に合せてシンボルの内容を表示する。



図－３　ＰＩ２の機能一覧

３．２　ＯＭＦ実行

マイクロプロセッサが実装された組込み型システムの場合、性能向上やメモリの節約のために、全てを高級言語で記述するわけにはいかないことが多く、クリティカルな処理部分はアセンブリ言語で記述する。ＰＩ２はＰＬ／Ｍ言語のソースプログラムを入力して、それをインタプリタ方式で実行するため、一部にＰＬ／Ｍ言語以外のコードがある場合、実行することができなかった。これらの部分に対しては、スタブモジュールを用意すれば実行することが可能である。しかし、システムインタフェースやパッケージとして提供されるコードについては、用意するスタブモジュール量が多く、実際には面倒なことが多い。このため、アセンブリ言語で記述されたモジュールとＰＬ／Ｍ言語で記述されたモジュールの両方を混在させて実行する機能が必要となった。

混在させて実行する方式には、

- オブジェクト実行方式
- アセンブラプログラムのインタプリタ方式
- ロードモジュール実行方式

がある。ロードモジュール実行方式はすでに検討しているため、ここではオブジェクト実行方式と、アセンブラプログラムのインタプリタ方式とを比較評価する。それぞれの方式の特徴を表－３に示す。

項　　目	オブジェクト実行方式		アセンブラプログラムのインタプリタ方式	
プログラマの思考	オブジェクトモジュール内の情報に依存する	△	データの参照、プログラムの表示等ソースプログラムと全く同一にできる。	○
対　象　範　囲	ターゲットマシンのプロセッサに依存する。	×	ターゲットマシンのプロセッサに依存する。（アセンラは機械語と１対１）	×
既存機能との親和性	ＰＬ／Ｍソースプログラムと共存が可能である。	○	同　　左	○
パフォーマンス	コードの解釈は単純に行える。	○	ソースプログラムの解釈が必要なため遅い。	×
結　合　編　集	結合編集用の情報をもつ。	○	同　　左	○

表－３　オブジェクト方式とアセンブラプログラム方式の比較

我々は、以上の点からオブジェクトを実行させる機能をＰＩ２に付け加えることとした。

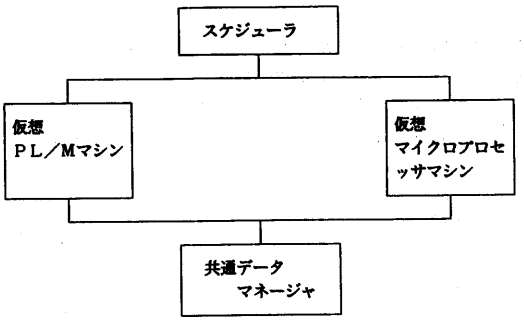
(1)　ＯＭＦ実行

ＯＭＦは，Ｏｂｊｅｃｔ　Ｍｏｄｕｌｅ　Ｆｏｒｍａｔ　の略であり，ＯＭＦ実行機能はオブジェクトモジュールを実行させる機能である。

ＯＭＦ実行機能は、ＰＩ２の機能の一部として実現されており、ＰＩ２の利用者は、特に意識せずに、ＰＬ／Ｍソースプログラムとオブジェクトモジュールを混在して実行することが可能である。

(2) 実現方式

ＰＩ２での，ＯＭＦ実行機能の実現方式を図－４　ＰＩ２の実現方式に示す。



図－４　ＰＩ２の実現方式

ＰＩ２は、仮想ＰＬ／Ｍマシンと仮想マイクロプロセッサマシンの二つの仮想マシンから構成されている。仮想ＰＬ／Ｍマシンは，ＰＬ／Ｍソースプログラム（実行容易な形式に変換した内部コード）を実行する。仮想マイクロプロセッサマシンは，ＯＭＦを実行させる。ＯＭＦの命令コードはターゲットマシンの命令コードと同一であり、仮想マイクロプロセッサマシンはターゲットマシンと同様の実行環境を実現するために各種レジスタもシミュレートしている。

二つのマシン間に生じる問題点は

・　実行モードの区別
・　仮想メモリへのアクセス

がある。

これらの問題は、スケジューラと共通データマネージャにより解決されている。

スケジューラは、実行対象のモジュールを判断して仮想マシンを選択する。スケジューラにより各仮想マシンが制御されている。

共通データマネジャーは、各仮想マシンから呼びだされ、データのアクセスを司る。データをアクセスする方法は、仮想マシン毎に異なる。仮想ＰＬ／Ｍマシンは、主にシンボル（変数名）によりアクセスされる。仮想マイクロプロセッサマシンは、レジスタ相対によりアクセスされる。共通データマネージャはこれらのアクセスに対して変換を施して、割り当てられた仮想メモリへのアクセスを行なう。

４．おわりに

ホストマシン上で、ターゲットマシンのテスト・デバッグを行なう時、次の二種類のツールが考えられる。

- ターゲットマシンに依存しないツール
- ターゲットマシンを意識したツール

ＰＩ２の初期の思想は、ターゲットマシンに依存しないツールを開発することであり、ＰＬ／Ｍの言語仕様を論理的に実行していた。ＯＭＦ実行は、逆にターゲットマシンを意識した機能である。
ＰＩ２は、ターゲットマシンのプロセッサに依存する機能と依存しない機能とを融合したものである。
ＰＩ２のこれからの課題はそれぞれの機能を高度化していくことである。プロセッサに依存する部分では，

- ターゲットマシンの環境の提供
- 各種のプロセッサに対応したＯＭＦの実行

があり、プロセッサに依存しない部分では、

- テスト管理機能の強化
- プログラム検証機能の提供

がある。

ＰＩ２の最終目標は、ターゲットマシン上でのテスト作業を無くすことである。ＯＭＦ実行機能を実現したことにより、ＰＩ２はその対象が、ＰＬ／Ｍソースプログラムとオブジェクトモジュールとになった。メインプログラムがＰＬ／Ｍ言語、サブルーチンがアセンブリ言語で記述されている場合や、その逆の場合も、ホストマシン上で実行することができるようになった。そのため、ホストマシン上でのテスト・デバッグ可能な範囲（対象のプログラム）が、大きく広がった。

ＰＩ２の欠点は、リアルタイムにプログラムを実行していないところにある。

プログラムの動作環境は、本来ターゲットマシン上である。そこでターゲットマシンとホストマシンをネットワークで接続し、ターゲットマシンの動作状況をホストマシンマシンから監視する。この事により、リアルタイムに動作しているターゲットマシンをホストマシンから制御することが可能である。ホストマシン上にはシンボル情報、ソースプログラムが存在するため、シンボリックにデバッグ環境を実現することができる。これらの機能を実現するために、ＰＩ２とは別に実現し、現在評価中である。

参考文献

[1] 羽田野、永田、武内：「対話型デバッグシステムＬＳＤにおけるデータ履歴の保存機能」
情報処理学会第２９回全国大会, 4R-11

[2] 立岡、大矢：「プログラム開発用支援ツール　－　Ｃインタプリタ」
沖電気研究開発125, Vol 52, No 1 pp. 9-14

[3] John D. Johnson and Gary W. Kenney: "Implementation Issues for a Source Level Symbolic Debugger "
ACM Software Engineering Notes　Vol. 8, No. 4 (August 1983) pp. 149-151

[4] Rich Seidner and Nick Tindall: "Interactive Debug Requirements"
ACM Software Engineering Notes　Vol. 8, No. 4 (August 1983) pp. 9-22