高位言語デバッギングシステム SOLDA†

春原 猛†† 大井房武††
関本彰次†† 中村敏行†††

SOLDA は PL/I 型の制御用高位言語 ESPRIT で記述されたプログラムを対象としてテストおよびデバッグを実行，支援する高位言語デバッギングシステムである．SOLDA は制御用実時間システムや計算機基本ソフトウェア等のような大規模で複雑なプログラムを開発する職業的なプログラマによって使用されることを考慮して設計され，効率的なテストおよびデバッグの諸機能を総合的に包含している．システムの特徴として，次の点があげられる．

（1） 実システムの構成との差異を模擬する機能
（2） 単体テスト・デバッグのための支援機能
（3） 会話またはバッチ形態での運用
（4） プログラムの外部からのテスト・デバッグ指示
（5） ソース・レベルの用語によるテスト・デバッグ指示および結果の出力
（6） プログラムの一時修正機能
（7） プログラムの実行プロフィール・データ収集機能

本論文では高位言語デバッギングの道具の各種実現方式の比較を行い，SOLDA システムの適用範囲と特徴，システム構成，テスト・デバッグ機能，被デバッグプログラムとのインタフェース，プログラム構成および使用経験から指摘された今後の課題について述べる．

1. まえがき

プログラミング言語の高位化により，プログラム作製の生産性，信頼性，保守性は著しく向上し，特にコーディングの効率化，プログラムの均質化および読解性の向上が指摘されている．しかし，プログラム作製工程の3～4割を占めるといわれるテストおよびデバッグ段階においては，機械に依存した手段，すなわち，オブジェクト・プログラムの情報または機械語の知識を必要とする道具等を使用せざるを得ない実情にあり，高位言語使用の環境が十分でなく効率の低下をもたらしている．

本論で紹介する高位言語デバッキング・システムは，その問題を解決する一つの手段を提供するもので，高位言語プログラムに対するテストおよびデバッグの指示や，その結果の情報出力をソース・プログラムで使用されている用語や名前，またはそれらと同じレベルの用語を使って行う．したがって，オブジェクト・プログラムの情報および機械語レベルの知識とソース・プログラムの間の情報変換をすることなく，テストおよびデバッグが可能になる．これによりテストおよびデバッグの生産性向上と標準化の促進，テスト内容の均質化，テスト結果のドキュメント性の大幅な向上が期待される．これまでにも高位言語デバッギングを支援する道具として，Conway ら[1]による診断機能をもった PL/C コンパイラ，Bull や原田らによる BASIC[3]や会話型 PL/I[2),4)]に対するコンパイラの付属機能，Grishman による AIDS[6]や TSS による FORTRAN プログラムに対するデバッグ・ツール[5]，迫田[7]や Panzl[8]らによるテスト手続き記述言語がある．この他にも，PL/I チェックアウト・コンパイラ，BUGTRAN, EXDAMS, ALGOL W デバッグ・ツール等がある．これらの道具の分類とその評価について後述するが，ここで対象としている制御用実時間システム等の大規模なソフトウェアの開発に適用する道具としては，次のような問題点がある．

a．制御用実時間システムの開発では，プログラム開発に使用する計算機システムと，目的とする実システムの構成（制御システムに特徴的な入出力機器構成および多量のシステム共通データの内容と構成等）が異なる場合が多い．したがって，テスト・デバッグの道具としてそれらのシステム構成の差異に起因する間隙を補完し，模擬する機能が要求される．

b．従来の道具は，おおむね小規模プログラムが対象であった．しかし，複雑かつ大規模なプログラムに

† SOurce Level Debugging Aids: SOLDA by TAKESHI SUNOHARA, FUSATAKE OOI, SHOOJI SEKIMOTO (Mitsubishi Electric Corp.) and TOSHIYUKI NAKAMURA (Ikutokiu Institute of Technology).

†† 三菱電機(株)
††† 徳島工業大学

おいては多数の部分に分割して，単体テストから始めることが基本的手順となる．プログラムの一部分である単体モジュールをテスト・デバッグするために，実行に必要な環境を設定する支援機能が要求される．

c．また，複雑かつ大規模なプログラムにおけるテストの網羅性を達成するためには，テストデータの多数の組み合せの実行が必要であり，テストを連続して効率的に実施し得ることが要求される．従来の道具は一部の例を除いてデバッグ支援を主体とし，テストの効率化の配慮が少ない．

これらの問題点の解決を目的として，高位言語デバッギング・システム SOLDA を開発した．

SOLDA (SOurce Level Debugging Aids) は制御用として新しく開発された PL/I タイプの高位言語 ESPRIT で記述されたプログラムを対象としてテストおよびデバッグを実行，支援する高位言語デバッギング・システムである．SOLDA は制御用実時間システムや計算機基本ソフトウェア等のような，規模大にして複雑なプログラムを開発する職業的なプログラマによって駆使されることを考慮して設計され，効率的なテストおよびデバッグの諸機能を総合的に包含している．

以下では，2章で高位言語デバッギング* の道具の各種実現方式の評価を行い，3章で SOLDA システムについて詳しく述べる．

2. 各種方式とその評価

高位言語デバッギングの道具の実現方法は，次に示す4種類に分類できる．

a．デバッグ・コード挿入方式[1]
b．インタプリンタ／逐次型コンパイラ付属機能方式[2)~4)]
c．デバッグ・ライブラリ方式[5]
d．デバッギング・システム方式[6),7)]

方式aはソースコードの一部として挿入されたデバッグコードがプログラムの一部として実行され，そのコードによって指定されたデバッグ動作がなされる方式である．方式bは高位言語のインタプリンタ方式や会話型言語に対する逐次コンパイル方式のコンパイラあるいは教育用に使用されるロード・実行(load-and-go)方式のコンパイラの付属機能として，デバッギング機能が設けられている方式である．方式Cはコンパイラによりデバッグ・ライブラリへの割込みコードや，ソース・オブジェクト対応情報(シンボル情報等)がオブジェクト・コード中に挿入される．リンケージ・エディット時にデバッグ・ライブラリが組込まれ，実行時に会話形式またはバッチ形式でオペレータから指示されたデバッグ動作が，デバッグ・ライブラリによって行われる．方式dは方式cと類似のものであるが，被デバッグ・プログラムとデバッギング機能部分が独立しており，被デバッグ・プログラムはデバッギング・システムの管理下で実行される．

上述した各方式を次のような観点から比較評価する．

（1） デバッグ手順の設定

方式aでは，コーディングとデバッグ準備を同時にするので効率的なデバッグ指示が与えられ，プログラムのデバッグしやすさを向上させる．また，保守時に条件つきコンパイルでデバッグ指示を簡単に再利用できる．一方，ソース・コード中にデバッグ指示が混在するため，ソース・リストの読解性を低下させる．方式a以外では，デバッグ指示は被デバッグ・プログラムの外から与えられるのでデバッグ指示に柔軟性があるが，デバッグ指示の再利用は煩雑である．しかし，方式dではデバッグ指示をプログラムの形にして保存することが可能であり，その場合には，デバッグ指示の再利用も簡単である．

（2） デバッグ機能

方式a，bは道具の実現方式上の制約あるいは対象とするプログラムが比較的簡単であることから，トレース，ダンプ機能中心の比較的単純なデバッグ機能を持つ．方式c，dでは豊富なデバッグ機能の実現が可能である．

（3） デバッグの能率

バグの発見が難しいエラーのデバッグは会話形式で行うのが能率的であるが，方式aでは会話形式のデバッグは不可能である．また同一プログラムに対して多数のテスト・ケースをバッチ・モードでテストする場合，方式dが最も能率的な方法を提供できる．

（4） エラーの波及の防御

被デバッグ・プログラムのエラーにより，デバッギング・システムや他のプログラムの実行に支障を生じる場合があるが，方式a，cではエラーの波及の防御がしにくい．

（5） 道具の開発コスト

---

* テストとデバッグは本質的に異なる作業であるが，本稿で述べる種類の道具を考える場合は両者に共通な部分が多く，明確に区別することが難しい．したがって，本稿では特に断りがない限り，デバッグという用語をテストを包含した意味で使用する．

デバッギングの道具の開発コストは，実現されるデバッグ機能の種類に大きく依存するが，一般に，方式aは開発コストは低く，方式dは高い．

3. SOLDA システム

3.1 SOLDA の適用範囲と特徴

SOLDA は PL/I 型制御用言語 ESPRIT[9),10)] で書かれたプログラムを対象とする高位言語デバッギング支援システムである．SOLDA はプログラムのテスト・デバッグを支援するツールであり，SOLDA と被デバッグ・プログラムは同一タスクとして実行される．被デバッグ・プログラムからサブタスクが起動されてもよいが，そのタスクに対してテスト・デバッグ指示を与えることはできない．

まえがきで述べた従来のシステムの問題点に対処するため，SOLDA は次のような特徴を持っている．

（1） 実システムの構成との差異を模擬する機能

制御用システムのプロセス入出力を模擬する機能により，プロセス入出力装置が結合されてない環境でプログラムのデバッグが可能である．

（2） 単体デバッグのための支援機能

プログラムの一部が未完成であっても，デバッグ環境設定機能により，単体デバッグ等を行える．

（3） テスト・デバッグ作業の効率化

- プログラムのテストとデバッグ機能を総合的にまとめて一体化したため，多数のテスト・ケースおよび複数個のプログラムのテストはバッチ処理で，複雑なバグの追求は会話処理で能率的に行える．
- デバッグ指示はプログラムの外部から計画的に，また状況に応じて自由に指定できる．
- デバッグ指示とデバッグ結果はソース・レベルの用語で出力され，保守のドキュメントとして役立つ．
- プログラムに対する一時的修正機能により，推定されたバグが誤りの真の原因であるかを，再コンパイルすることなく確認できる．
- 実時間システムでは，プログラムの実行速度に対する要求が厳しいが，プログラムの実行状況（実行プロフィール）データ収集機能により得られる実行頻度の分布を調べ，性能改善を図るべき場所を的確に指摘できる．また，その機能により未実行パスの確認ができ，テストの網羅性のチェックも可能となる．

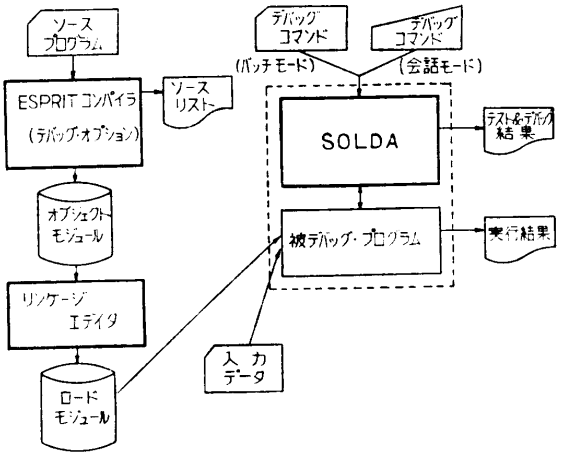


図 1 SOLDA のシステム構成
Fig. 1 System organization of SOLDA.

3.2 システム構成

複雑なデバッグ機能の実現し易さ，拡張可能性およびデバッグ作業の効率化の観点から SOLDA はデバッギング・システム方式（2章の方式d参照）を採用した．SOLDA は**図1**に示されるシステム構成になっており，SOLDA を使用したデバッグ作業は次の三つのジョブからなる．

（1） コンパイル・ジョブ

デバッグの対象となるモジュールをデバッグ・オプション指定でコンパイルする．

（2） リンケージ・エディット・ジョブ

(1)でコンパイルされたモジュールの集合をテスト済みのモジュールと共にリンケージして，プログラムのロード・モジュールを作成する．

（3） デバッグ・ジョブ

(2)で作成された被デバッグプログラムを，(1)で得たソース・リストを参照しながらデバッグする．SOLDA は被デバッグプログラムを実行させながら，外から与えられたデバッグ・コマンドによって指定されたデバッグ動作を行う．

3.3 デバッグ機能

SOLDA のデバッグ機能の体系を**図2**に示す．また，デバッグ・コマンドの構文と意味を**図3**，**表1**に示す．デバッグ・コマンドおよびデバッグ情報の出力はソース言語レベルの用語で表現される（**図4**参照）．会話モードの場合にも，出力情報のハード・コピーが生成されるので，デバッグのドキュメント性の向上が図れる．

環境設定機能として，単体デバッグのためのパラメ

- デバッグ指示入力機能
- デバッグ情報出力機能
- 環境設定機能
  - システムデータ設定機能
    - リンケージの模擬機能
      - アーギュメント生成機能
      - 未完成モジュールスキップ機能
  - プロセス入出力の模擬機能
    - 実／疑似入出力の選択機能
    - 入出力の模擬機能
- 被デバッグプログラムの実行制御機能
  - 実行の開始／終了機能
    - プログラムの実行開始／終了機能
    - 文単位の実行開始／終了機能
  - 実行再開機能
    - 実行中断点からの再開機能
    - 過去の時点からの再開機能
    - 中断点からスキップした点からの再開機能
  - 異常状態処理機能
- デバッグ動作機能
  - デバッグ動作の実行時点指示機能
    - 中断点の指定機能
    - 論理条件の指定機能
    - ブレークキーによる中断機能
  - デバッグ動作指示の修正機能
  - 値設定機能
    - 変数の値設定機能
    - 繰返し設定機能
  - 値／メッセージ出力機能
    - 変数の値出力機能
      - 属性に応じた形式による出力
      - 16進形式出力
    - インターバル出力機能
  - ダンプ機能
    - スナップ・ショット・ダンプ機能
    - ポスト・モーテム・ダンプ機能
    - ダンプ内容指示機能
      - ブロック起動／文実行経過
      - 変数領域ダンプ
      - 異常状態情報ダンプ
  - トレース機能
    - トレース制御機能
      - トレース開始／終了指示
      - トレース種別指示
    - フロートレース機能
      - ブロックトレース
      - ブランチトレース
    - 変数トレース機能
  - パッチ機能（プログラムの一時修正）
    - 文の追加／変更／削除機能
    - 変数の追加宣言機能
  - 実行プロフィール情報収集機能
    - 文／手続きの実行頻度収集機能
- 補助機能
  - 作業モード指示機能
  - デバッグコマンド問合せ機能
  - デバッグ出力制限機能
  - 複数テストケースに対するテスト機能

図 2 SOLDA の機能

Fig. 2 Functions of SOLDA.

```
<ADD コマンド>::=ADD {<代入文> | <GO TO 文> | <DECLARE 文>} BEFORE <文指定子>;
<BATCH コマンド>::=BATCH;
<CHANGE コマンド>::=CHANGE <文指定子> FOR {<代入文> | <GO TO 文>};
<CONTINUE コマンド>::=CONTINUE [FROM <開始点文番号>]
                      [UNTIL <停止点文番号リスト>];
<DEBUG コマンド>::=DEBUG <プログラム名> [<オプション>];
<DELETE コマンド>::=DELETE <文指定子>;
<DIALOG コマンド>::=DIALOG [<ポイント指定子>];
<END コマンド>::=END;
<EXCUTE コマンド>::=EXECUTE
                    [UNTIL <停止点文番号リスト>];
<HELP コマンド>::=HELP;
<IF コマンド>::=IF <式> <ポイント指定子> THEN
              <then-else ユニット> [ELSE <then-else ユニット>];
<PISET コマンド>::=PISET <プロセス入力セット指定リスト>
<PRINT コマンド>::=PRINT [<メッセージ>] [<名前リスト>]
                    [[EVERY <インターバル数>] <ポイント指定>];
<PROFILE コマンド>::=PROFILE {<文番号リスト> IN <手続き名> | <手続き名リスト>};
<SET コマンド>::=SET <セット指定リスト> [<ポイント指定子>];
<SNAP コマンド>::=SNAP [<ポイント指定子>];
<SUSPEND コマンド>::=SUSPEND
                    {ALL | <無効コマンドキーワードリスト>}
                    FOR <文指定子> [<ポイント指定子>];
<TRACE コマンド>::=TRACE {<名前リスト> | FLOW | ALL | OFF} [<ポイント指定子>];
```

図 3 デバッグ・コマンドの構文

Fig. 3 Syntax of the SOLDA debug command.

表 1 デバッグコマンドの意味

Table 1 Semantics of the SOLDA debug command.

コマンド	意味
ADD	被デバッグプログラムに文や変数宣言の追加
BATCH	会話モードからバッチモードに作業モードを変更
CHANGE	被デバッグプログラムの文の変更
CONTINUE	〃 の実行再開と，一時停止点指定
DEBUG	〃 の指定
DELETE	〃 の文の削除
DIALOG	バッチモードから会話モードに作業モードを変更
END	デバッグ作業の終了
EXECUTE	被デバッグプログラムの実行開始と，一時停止点指定
HELP	デバッグコマンドについての問合せ
IF	条件分岐によるデバッグ動作の選択
PISET	プロセス入力模擬のための入力データ準備
PRINT	変数の値およびメッセージの出力
PROFILE	文の実行回数カウント
SET	変数（パラメタ）への値設定
SNAP	スナップ・ショット・ダンプの指示
SUSPEND	デバッグ動作を無効にする
TRACE	変数トレース，フロートレース，トレース中止の指示

注）コマンド・キーワードの省略形として，上位3文字が使われる．

```
MELCOM SOLDA VERSION-A 00  77-04-06  PAGE  1
*  MELCOM SOLDA                                  デバッグ開始メッセージ
*  :IN: DEBUG SEARCH;
*  :IN: SET M=5 AT 1;
*  :IN: SET TABLE=(7,28,
*  :IN:                5,31,
*  :IN:                8,10,
*  :IN:                6,60,                    バッチ・モードで入力
*  :IN:                2,23) AT 1;              された一連のデバッグ
*  :IN: SET X=5 AT 1;                           コマンドの出力
*  :IN: PRINT '----X: KEY,
       Y: DATA----' X,Y AT 17;
*  :IN: IF Y=31 AT 17
       THEN PRINT 'O.K.'
*  :IN: ELSE DIALOG;
*  :IN: EXECUTE;
*  DEBUGGED PROGRAM NAME: SEARCH
                                    プログラム実行開始メッセージ
*  CASE 1 START
*  AT   1 OF SEARCH
*   * SET *    M=5
*   * SET *    TABLE(1).KEY=7
*   * SET *    TABLE(1).DATA=2              デバッグ出力情報
*   * SET *    TABLE(2).KEY=5
       :
*   * SET *    X=5
*  AT   17
*   * PRINT *  ----X: KEY,                  デバッグ出力情報Yの
                Y: DATA----                 値が正しくない
*   * PRINT *  X=5                             (エラーの発見)
*   * PRINT *  Y=10
*   * IF *     FALSE                        DIALOG コマンドにより会話モードに移る
```

図 4 デバッグの出力例

Fig. 4 Sample debug listing.

ータに対するアーギュメント生成機能や，未完成モジュールの呼出しスキップ機能がある．プロセス入出力の模擬は PISET コマンドで指定されたデータを入力時点で設定し，出力時点では出力データを印字する．

プログラムは直接実行方式で実行されるが，会話モードの場合は EXECUTE, CONTINUE コマンドの出発点，停止点の指定により，プログラムの実行を制御する．

デバッグ動作の実行時点は文番号で指定されるか，あるいは会話モードの場合はコマンド入力直後に動作が行われる．条件判定によるデバッグ動作の選択機能は IF コマンドにより指定する．また，デバッグ動作は SUSPEND コマンドにより無効にされる．SET, PRINT コマンドは値設定および値出力を指示する．プログラム実行中に異常状態が発生した場合はブロック起動経過，起動中のブロックの変数領域のダンプおよび異常状態情報からなるポスト・モーテム・ダンプが行われ，次のプログラムのデバッグに移る．プログラム実行途中でのダンプは SNAP コマンドにより指示する．TRACE コマンドは変数およびフロー・トレースを制御する．ADD, DELETE, CHANGE コマンドにより，プログラムの一時修正が行われる．プログラムの実行頻度データを得る場合には PROFILE コマンドを使う．

作業モード（会話／バッチ）の指定は SOLDA のジョブ・コントロール・コマンドで指定されるが，デバッグの途中で BATCH, DIALOG コマンドでモードを変更することが可能である．また，バッチ・モードでは複数プログラムのデバッグや同一プログラムに対する複数個のテスト・ケースについてのテスト等が行われる．

3.4 被デバッグ・プログラムとのインタフェース

被デバッグ・プログラムのオブジェクト・コード中に ESPRIT コンパイラにより，次の2種類のコードが挿入される．

（1） 割込みコード

割込みコードは被デバッグ・プログラム中の次に示す個所に生成され，プログラム実行中に SOLDA に制御が渡る．

- 実行文の先頭
- ブロックの入口，出口
- 手続きの呼出し地点
- 変数の代入直後

（2） 情報コード

SOLDA がオブジェクトとソース・レベルの情報の対応づけをするために使用する次のコードが生成される．

- シンボル情報（属性，アドレス）
- 文のアドレス情報
- ブロックのスコープ情報

SOLDA と被デバッグ・プログラムは同一のタスクとして動作する．**図5**に SOLDA と被デバッグ・プログラム間のインタフェースを図示する．SOLDA によって実行が開始された被デバッグ・プログラム中で割込みコードに遭遇すると SOLDA に実行制御が渡る．SOLDA はデバッグ動作テーブルをみて動作が指定されていれば，被デバッグ・プログラム中に挿入されている情報コードを参照しながらデバッグ動作を行う．会話モードの場合，割込み地点（被デバッグ・プログラム中で割込みコードによって，実行制御が SOLDA に移る場所）があらかじめ指定された一時停止点ならば，オペレータからのデバッグ・コマンドを受けつけることになる．割込み地点に指定されたデバ

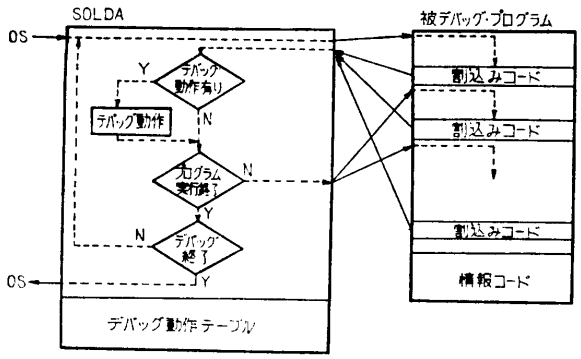


図 5 SOLDA と被デバッグ・プログラムのインタフェース

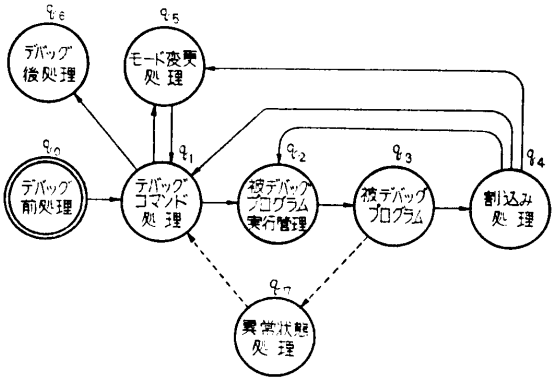
Fig. 5 Interface between SOLDA and a debugged program.

ッグ動作を終了した後，プログラムに実行制御を渡し，実行を再開させる．

現在，被デバッグプログラムに上記の挿入コードが入るとプログラム・サイズは2倍ないし3倍となる．また，挿入コード中，割込みコードの占める割合の方が大きい．挿入コードのサイズ縮少は今後の課題の一つである．

3.5 プログラム構成

SOLDA は次に述べる8個の基本的なプログラムから構成され，「SOLDA ドライバ」が他のプログラムを制御する．したがって，SOLDA における制御の流れは，ドライバ以外の7個のプログラムと被デバッグ・プログラム間の制御の流れとして捉えることができる（**図6参照**）．



(q_0, SOLDA ジョブ開始) $\rightarrow q_1$
(q_1, EXECUTE/CONTINUE コマンド) $\rightarrow q_2$
(q_1, DIALOG/BATCH コマンド) $\rightarrow q_5$
(q_1, END コマンド) $\rightarrow q_6$
(q_2, プログラム実行開始／再開) $\rightarrow q_3$
(q_3, 割込みコードに遭遇) $\rightarrow q_4$
(q_3, 異常状態発生) $\rightarrow q_7$
(q_4, 実行終了／一時停止点) $\rightarrow q_1$
(q_4, デバッグ動作終了) $\rightarrow q_2$
(q_4, 作業モード変更指定) $\rightarrow q_5$
(q_5, モード変更処理終了) $\rightarrow q_1$
(q_7, 異常状態処理終了) $\rightarrow q_1$

図 6 SOLDA における状態遷移

Fig. 6 State transition in SOLDA.

（1） SOLDA ドライバ

プログラム間の制御の流れの管理を行う．また，被デバッグ・プログラムからの割込みおよび異常状態発生による割込みを受けつける．

（2） デバッグ前処理

デバッグ動作テーブルの初期設定や，異常状態割込みを OS から SOLDA に取りこむための処理をし，開始メッセージを出力する．

（3） デバッグ・コマンド処理

デバッグ・コマンドを解析し，結果をデバッグ動作テーブルに登録する．会話モードの場合は一部のコマンドを除いて，デバッグ動作は登録されずに，ただちに実行される．

（4） 被デバッグ・プログラムの実行管理

被デバッグ・プログラムの指定された地点や割込み地点への制御の移行（すなわち，被デバッグ・プログラムの実行）を行う．その際に，割込み地点に指定されたソース・プログラムの修正動作，プロセス入出力模擬，手続きのロード／スキップ処理等を行う．

（5） 割込み処理

割込み地点にデバッグ・コマンドで指定された動作があれば実行する．トレースおよびプログラム実行プロフィール情報収集はここで行う．

（6） デバッグ後処理

デバッグ前処理で行った異常状態割込みを SOLDA に取り込むための処置を解消し，デバッグ作業の後処理をした後，終了メッセージを出力する．

（7） モード変更処理

バッチ・モードから会話モード（または，その逆）への作業モードの変更を行う．

（8） 異常状態処理

ポスト・モーテム・ダンプをして，次のプログラムのデバッグ作業に移る．

4. むすび

SOLDA は MELCOM 350-50 TSOS システムで稼動する．SOLDA のプログラム自身は FORTRAN 型の高位言語で記述されており，総サイズは 88.2 k バイトである．実行に必要な主記憶容量は 31.2 k バ

イトと被デバッグ・プログラムのサイズを加えたものである.

SOLDA の使用実験から，次のような点が今後の課題として指摘されている.

(1) 被デバッグ・プログラム中のデバッグ用挿入コードのサイズ縮少 (3.4 節参照)

(2) ソース・プログラムの修正に影響されないデバッグ動作位置の指定方法

(3) 主記憶常駐データ参照および補助記憶域参照の模擬

(4) 被デバッグ・プログラムの実行時間測定機能

(5) テスト手続き記述言語の開発

謝辞 SOLDA チームのメンバであり，本稿について有益な助言を頂いた紺田茂實，小林博，臼井澄夫 (コンピュータ・システム工場) および大室隆 (開発本部計算機研究部) の諸氏に感謝します.

参 考 文 献

1) Conway, R. W. and Wilcox, T. R.: Design and Implementation of a Diagnostic Compiler for PL/I, Commun. ACM, Vol. 16, No. 3, pp. 169-179 (1973).
2) IBM System/360 OS (TSO), Interactive Terminal Facility: PL/I, GC 28-6827-0.
3) Bull, G. M.: Dynamic Debugging in BASIC, Comput. J., Vol. 15, No. 1, pp. 21-24 (1972).
4) 原田: インタラクティブ PL/I システムの設計と作成，情報処理，Vol. 16, No. 2, pp. 85-92 (1975).
5) IBM FORTRAN Interactive Debug for OS (TSO) and VM/370 (CMS) Terminal User's Guide, SC 28-6885-1.
6) Crishman, R.: The Debugging System AIDS SJCC, pp. 59-64 (1970).
7) 迫田，霜田，小島，桑原，小崎: プログラムテストシステム TPL(1), (2)，昭 49 情処学会第 15 回大会，pp. 661-664 (1974).
8) Panzl, D. J.: Test Procedures: A New Approach to Software Verification, Proc. 2nd International Conference on Software Engineering, pp. 477-485 (1976).
9) 太細，定松，向井，竹田，武藤，居原田: PL/I 型制御用システム記述言語 ESPRIT の特徴について，昭 51 信学会誌総合全国大会，p. 6-171 (1976).
10) Dasai, T., Iharada, K., Sunohara, T., and Nakamura, T.: High Level Process Control Language "ESPRIT" and Its Source Level Debugging System "SOLDA", Proc. 1977IFAC-IFIP Workshop on Real-Time Programming (1977).
11) Gaines, R. S.: The Debugging of Computer Programs, Ph. D. Thesis, Princeton Univ. (1969).

(昭和 52 年 8 月 31 日受付)
(昭和 54 年 4 月 19 日採録)