UNIXを利用した8086系交換ソフトウェア開発サポートシステム

石川惣一　森田隆治　清原伸一　中本幸一　岡田直美　高木利公（日本電気）
谷沢弘孝　清水泰二（日本電気通信システム）

1．はじめに

近年，電子交換システムは，増々高度化かつ複雑化し，情報処理社会の中枢を担う様になっている。これに対処する手段の1つとして，従来の大型機による集中化処理から，分散化処理へと進んでいる。さらに低価格の16ビット汎用マイクロプロセッサが市場に出回り，電子交換システムの経済的な分散処理システムの実現を可能とし，分散化処理指向の急速な展開を見せている。

このため，我々は8086系交換ソフトウェアのサポートシステムの開発が必要となった。この機会にソフトウェア開発の飛躍的な生産性向上を目的とした，UNIXをベースにミニコンによる小回りのきく，使い勝手の良い高能率のシステムを開発し，実用化した。

以下，本システムの特徴と概要について紹介する。

2．システムの特徴

本システムは以下の特徴を有している。

(1) ファイル化*環境とデバッグ環境の有機的な結合

従来は，ファイル化マシンとデバッグマシンが結合されていなかったため，デバッグ段階でバグを発見すると，その時点でパッチを作成し，検証を行い，その後別マシンでソースファイルを修正していた。そのため，非常に能率が悪く，しかもパッチによる変更情報管理が煩雑であった。これをファイル化環境とデバッグ環境を有機的に結合することにより，ターンアラウンドの大幅な改善とパッチデータの大幅な削減を可能とした。

(2) ハードウェアに独立なソフトウェアの開発の支援

移植性の高い交換用言語としてC[2]を採用した。これに伴い，Cクロスコンパイラ処理系をシステムに構築している。

(3) 既開発ソフトウェア資産の流用

既に開発済みの膨大な，ソフトウェア資産の有効利用を図るため，交換用言語変換機能をツールとして持っている。

(4) 小回りのきくシステム

モジュール単位にコンパイル，アセンブルができ，リンクが容易である。プログラムの修正，改造が迅速に可能である。オープンエンドで機能追加が容易である。

(5) 自席でのプログラム開発

従来，ファイル化作業とデバッグ作業とは離れた所で行われており，デバッグはターゲットマシンのそばでの作業を強いられていたが，机上からリモート端末を介してのデバッグおよびプログラム修正が可能となった。

(6) 強力なオンラインデバッグサポート機能

リアルタイムトレース，トリガトレースアナライザ等の強力なオンラインデバッグサポート機能を有している。

---

* ソースモジュールを作成し，コンパイルまたはアセンブルし，オブジェクト作成後，オブジェクト間の相互参照のアドレスリンケージをとり，ターゲットマシンでロード，実行可能なロードモジュールファイルを作成する一連の作業をファイル化と称す。

3．システム構成と機能

本システムは，VAX-11/750（OS：UNIX 4.1 bsd）をホストマシンとして，UNIX環境上にファイル化サポート（交換用言語変換ツール，Cクロスコンパイラ，クロスアセンブラ，リンカー等）を構築すると共に，市販のデバッグマシン（ICE：Inner Circuit Emulator）およびPROMライター，パーソナルコンピュータ等を光ネットワークでVAX本体と接続し，ファイル化環境とデバッグ環境との結合を図った8086用ソフトウェアのプログラム開発サポートシステムである。

以下に本システムの構成および機能概要を示す。

3.1 ハードウェア構成

サポートハードウェア構成を図1に示す。

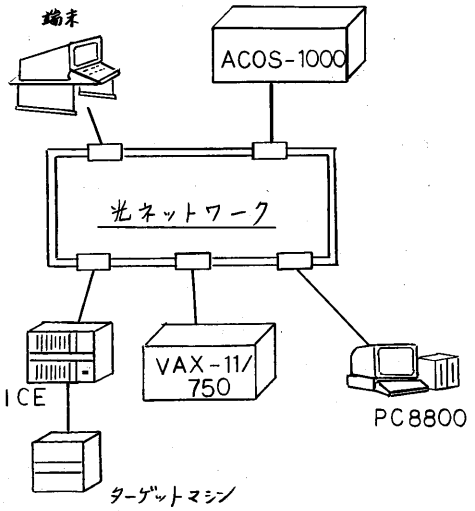

図1．サポートハードウェア構成

（ACOSは初期の大量なソースファイル作成に使用され，作成されたソースファイルはVAXに転送され，UNIX上のファイル構成に変換される。）

3.2 システムフロー

ファイル化からデバッグまでの一連の処理手順フローを図2に示す。

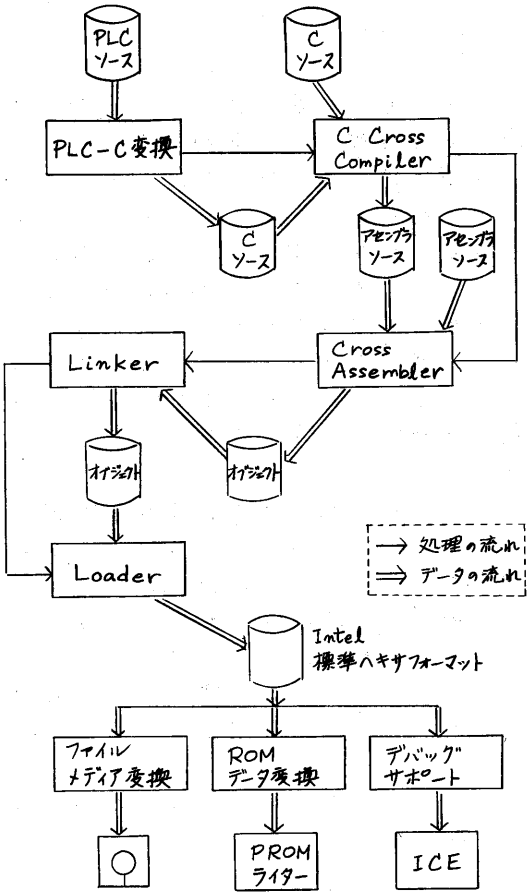

図2．処理手順フロー

3.3 機能概要

上記各ツールの機能について，簡単に説明する。

（1）PLC－C変換

以前は，交換用言語としてPLCを使用していた。PLC－C変換は，既存PLCソースプログラムの有効活用をはかるために，定められた変換ルールに基づいて，PLCソースをCソースに変換するものである。

（2）ファイル化サポート

Cソースを入力して，コンパイル，アセンブル，リンクし，最終的にイン

テル標準ヘキサフォーマットに変換する一連の処理である。

(3) RŌMデータ変換

PRŌMライタをコミュニケーションモードで制御し，ホストマシン上のデータをダイレクトにPRŌMにロードする。

(4) ファイルメディア変換

UNIX端末として使用しているPC-8800のフロッピーディスクにホストマシン上のデータをセーブする。また，この逆も行う。この機能は，交換システム納入現地でのデバッグに有効となる。

(5) デバッグサポート

ファイル化環境とデバッグ環境を結合し，ファイル化，デバッグのサイクルの時間的短縮を図るため，ホストマシンとICEとの間のインタフェースを確立し，ホストマシン，ICE間のデータ転送を可能としたものである。

次に，本システムの特徴的機能である，PLC－C変換，デバッグサポートについて特に記述する。

3.4 PLC－C変換

(1) PLCとC

PLC-C変換のソース言語であるPLCとターゲット言語であるCを説明する。

PLCは，PL/Iのサブセットであり，電子交換機のソフトウェアの記述に用いられている。IBMのPL/I[1]に比して以下の制限がPLCにはある。

(i) PLCの扱うデータ型は，非負整数型のみである。

(ii) 入出力機能　ファイル機能がない。

(iii) 値を返す手続き(Cの関数に当たる)がない。

(iv) ストリング操作を除いて，組込み関数がない。

(v) DŌ文が，DŌ WHILE型とDŌ TŌ BY型に限られる。

更に，PL/Iにない機能として，以下のものがある。

(i) Cのunionに類したCELLというデータの集合体を記述する機能がある。

(ii) 交換機特有のデータ型がある。

(iii) 機械語を記述することができる。

一方，ターゲット言語は，Kernighanらの定義したC[2]に，異なる構造体においても，メンバ名の重複を自由に許す，という拡張を加えたものである。

(2) 変換ルール

ここでは，PLC－C変換の主たる変換ルールをあげる。

(a) BASED変数

BASED変数は，メモリ領域をあるデータの枠組に従って参照する場合に用いられる。例えば，図3において，②は，ポインタ変数①の指す領域を②で定義された構造体で参照することを示す。このBASED変数は③のように，その変数に対応するCの構造体に変換する。

```
DCL   P   PTR, ------ ①
DCL   1   A   BASED(P),  ┐
          2  B   BIN,    │②
          2  C   BIN:    ┘
P->A.B = 1 ;
            ⇓
struct  {                      ┐
      struct   dummy0 {        │
           int  B ;            │
           int  C ;            │③
      } A ;                    │
} *P ;                         ┘
P->A.B = 1 ;
```

図3. BASED変数の変換例

```
DCL   1   A1,
          2  A2   BIN(8),  ┐
          2  A3   BIN(8),  ┘ ②
          2  A4,
             3  A5   BIN(4),   ③
             3  A6   CELL,
                4  A7   BIN(8),
                4  A8   BIN(8):
            ⇓
struct  {
    unsigned  int   A3: 8 ;  ┐
    unsigned  int   A2: 8 ;  ┘←
    struct  {
        unsigned  int  dummy1:12 ;  ┐
        unsigned  int  A5: 4 ;      ┘←   ①
        union  {                      ←
            unsigned  int  A7: 8 ;
            unsigned  int  A8: 8 ;
        }  A6 ;
    } A4 ;
} A1 ;
```

図4. 構造体の変換

(b)構造体

PLCの構造体は以下のルールに従って，Cの構造体に変換する。

(i)CELLはunionに変換する(図4①参照)

(ii)BIN(i)，BIT(i)(1≦i≦15)は，1ワード(16ビット)内で，その出現順序を逆にする(図4②参照)。(PLCとCとではワード内の割付け順序が逆になるため。)

(iii)構造体の次の要素がワード境界で，その前までがワードの最後までに割付られていない場合，ダミーのメンバを発生する(図4③参照)。

(c)構造体の参照

PLCでは，構造体の変数参照は，曖昧さのない限り途中の名前を省略できるが，Cでは省略できない。従って，図5にあるように，途中のメンバを補って出力する。

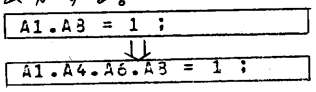

図5. 構造体参照の変換

(d)手続呼出し

PLCは手続の呼出しが番地呼出しであるのに対し，Cでは値呼出しである。これは，Cの関数の実パラメータに&演算子を，仮パラメータに*演算子を付加することで変換可能である。

(3)実現

今回は，比較的短期間(1ヶ月)で完成を要請されたため，PLCの全ての機能を自動変換するものではなく，入力プログラム中に出現する頻度の低い機能(例えば前述の手続呼出し)は，人手で変換を行うことを，作成方針とした。

PLC－C変換は，語彙解析部，構文解析部(入力プログラムを構文解析し，構文木と名前表を作成する)，変換部(構文木と名前表からCのプログラムを生成する)から成る。語彙解析部はLEX[3]，構文解析部はYACC[4]を用いた。作成規模は3.9Kステップであり，3人月を要した。図6に入力プログラム，図7に出力プログラムの実例を示す。

(4)適用例

4章で述べるプロジェクトにPLC-C変換を適用した。入力されたPLCソースプログラムは25Kステップであり，人手による開発に比して，1/2以上の工数削減の成果が得られた。

```
   IF DMTDCMP.KIND GE 1 AND DMTDCMP.KIND LE 4
      THEN
         DO                          ;   /* <1> DO    */
   CASE DMTDCMP.KIND OF 1:4  ;   /* REQ PROC CATRGC
(1):
/*          < A >                                  */
/*********** CP DET-TRF CTL-TBL          *********/
/***********                  RESIST      *********/
      DMTUSUB.PRN = EMTCPUC.NOD3 ;
      CALL CMTUIP                 ;   /* CP DET-TRF CHK
                                      /* CALL 24.5   (MT
      DMTTCIP.PRN(DMTUSUB.JDX) = EMTCPUC.NOD3 ;
      IF EMTCPUC.ECD4 = U3NUL                        /*
         THEN DMTTCIP.F4(DMTUSUB.JDX) = 1 ;          /*
         ELSE                        ;               /*
/*          < A1 >                                 */
/********** OUT DIVICE NO SET          *********/
      IF EMTPUC.ECD6 = UOLP          /* OUT DIVICE LP
         THEN
            DO                       ;   /* <2> DO  */
            DMTTCIP.SID(DMTUSUB.JDX) = EMTCPUC.NOD6 ;
                                      /* PRN SET
            DMTTCIP.DVNOB(DMTUSUB.JDX) = EMTCPUC.N61 ;
                                      /* RECOMPH NO SET
            END                      ;   /* <2> END */
         ELSE
            DO                       ;   /* <3> DO  */
            DMTTCIP.SID(DMTUSUB.JDX) = TACCDR.INCP ;
                                      /* PRN SET
            DMTTCIP.DVNOB(DMTUSUB.JDX) = TACCDR.INDEV;
                                      /* RECOMPH NO SET
            END                      ;   /* <3> END */
         DMTTCIP.ECD(DMTUSUB.JDX) = EMTCPUC.ECD6 ;
                                      /* ECD CORD SET
/********** END      < A1 >            *********/
```

図6. PLC－C変換入力例

```
if ( DMTDCMP.KIND >= 1 && DMTDCMP.KIND <= 4)
{     /* <1> DO   */
  switch ( DMTCMP.KIND ) {
     /* REQ PROC CATEGORY     */
  case 1 :
    /*          < A >                                */
    /********** CP DET-TRF CTL-TBL          *********/
    /***********                  RESIST      *********/
    DMTUSUB.Z2.>PRN = SCYTCRBP -> EMTCPUC.CPI.NOD3 ;
    CACMTUIP() ;      /* CP DET-TRF CHK        */
    /* CALL 24.5   (MT084)  */
    DMTTCIP.INFOC ( DMTUSUB.JOX ) - 1J.Z2.PRN = SCYTCRBP
    -> EMTCPUC.CPI.NOD3 ;
    if ( SCYTCRBP -> EMTCPUC.CPI.ECD4 == U3NUL)   /* PH2  */
       DMTTCIP.INFOC ( DMTUSUB.JOX ) -1J.Z1.Z11.F4 = 1 ;  /* P
    else  ;     /* PH2  */
       /*       < A1 >                               */
       /********** OUT DIVICE NO SET          *********/
    if ( SCYTCRBP -> EMTCPUC.CPI.ECD6 == UOLP )  /* OUTPUT DIV
    {  /* <2> DO  */
       DMTTCIP.INFOC ( DMTUSUB.JDX ) -1J.Z3.Z31.SID = SCYTCRBP
       -> EMTCPUC.CPI.NOD6 ;     /* PRN SET              */
       DMTTCIP.INFOC ( DMTUSUB.JDX ) -1J.DVNOB = SCYTCRBP
       -> EMTCPUC.CPI.N61 ;      /* RECOMPH NO SET       */
    }  /* <2> END */
    else
    {  /* <3> DO  */
       DMTTCIP.INFOC ( DMTUSUB.JDX ) -1J.Z3.Z31.SID = SCYTCRBP
       -> TACCDR.CDR.INCP ;      /* PRN SET              */
       DMTTCIP.INFOC ( DMTUSUB.JDX ) -1J.DVNOB
       -> TACCDR.CDR.INDEV ;     /* RECOMPH NO SET       */
    }  /* <3> END */
    DMTTCIP.INFOC ( DMTUDUB.JDX ) -1J.Z3.Z31.ECD = SCYTCRBP
    /* ECD CORD SET          */
    /********** END      < A1 >            *********/
```

図7. PLC－C変換出力例

3.5 デバッグサポート

ホストマシン上でファイル化されたプログラムはICEにダウンロードされ，ICEを使ってデバッグされる。以下にデバッグサポートとしての機能について述べる。

(1) ホスト，ICE間のデータ転送

ホストマシンとICEは光ネットワークで結合されており，ホストマシンからICEへのデータのダウンロード，ICEからホストマシンへのアップロードが可能である。

ホスト，ICE間のデータ転送のプロトコルは基本的伝送制御手順で行われている。また，転送データはHEX形式であり，1レコード内にデータレコード長，チェックサムの情報が含まれており，転送時，受信側でチェックされる。このことにより，データ転送の信頼性が確保されている。

(2) デバッグ手順

この環境のもとでのプログラム作成，デバッグの手順は図8のようになる。これによると，従来の，ファイル化によって得られたオブジェクトを一旦MT等媒体にセーブし，それをターゲットにロードする，といった手間は省かれ，ホストからダイレクトにICEにオブジェクトをロードできるので，ファイル化工程から即デバッグ工程に移ることができる。

また，デバッグ環境からファイル化環境に即もどることができるので，従来のように，プログラムの誤りをパッチで修正，検証するという手間が省かれ，即ソースプログラムを修正することにより，プログラムの訂正が行われる。

(3) プログラム開発環境

ホストマシンとICEが光ネットワークで結ばれていることから，作業端末はターゲットの近くにある必要はなく，また，ファイル化作業とデバッグ作業は同一端末でできるので，自席近くに置いた端末を利用してプログラム開発することができる。

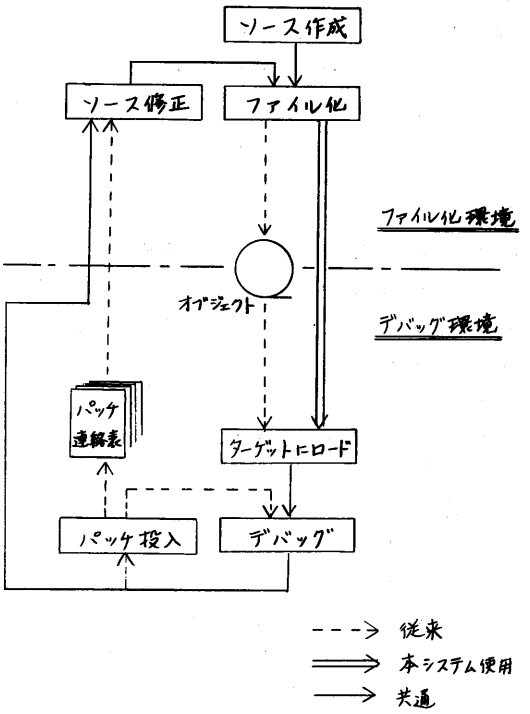

図8. デバッグ手順

(4) シンボリック・デバッグ

ホスト側からシンボル情報をICEにダウンロードすることにより，ICEでのデバッグをシンボリックに行うことが可能である。

これは，ホスト側で，まずオブジェクト・プログラムからシンボルの情報（シンボル名，アドレス，セグメント名等）を抽出し，それをHEX形式に変換し，ICEにダウンロードすることにより実現される。

これによって，デバッグにおける，メモリのread，write時のアドレス指定，ブレークポイントのアドレス指定等に従来のように物理アドレスで指定する必要はなく，シンボルを使うことができる。また，メモリ内容の表示等にシンボル（ラベル）を含ませる

ことができる。

(5) デバッグ結果のセーブ

ＩＣＥ上でのデバッグの完了したプログラム（ＩＣＥ，又はターゲットのメモリ内容）をホスト側にアップロードし，ファイルにセーブすることができる。

(6) ＩＣＥでのデバッグ機能

ＩＣＥ上にダウンロードされたプログラムは，ＩＣＥの次の機能によりデバッグされる。

- メモリの read，write
- レジスタの read，write
- I/Oポートの read，write
- 逆アセンブル
- メモリプロテクト
- ブレークポイントの設定
- 複合トラップ（readトラップ，writeトラップ，fetchトラップ，アドレス範囲，データ値の範囲，外部シグナル，カウンタの組合せ）
- トレース
- リアルタイムトレース
- エミュレーションモードの選択（ICEのメモリ，ターゲットメモリの使用法）

4. システムの適用例

本システムは，パケット交換のプロジェクトで適用中であり，以下の効果が得られている。

従来のシステムではファイル化をバッケ処理で行っていたが，本システムではTSS中心の対話型のためターンアラウンドタイムが短くなった。また，UNIXの豊富で強力なコマンド群を使用することにより，効率的なプログラム開発を行えるようになった。例えば，lint[5]によりコンパイラでチェックし切れないプログラムエラーを検出したり，sdb[6]の使用により，ターゲットマシン（8086）が完成する前でもUNIX上でソフトデバッグ（論理チェック）を行うことが可能となった。そのほか，make[7]を使うことによってプログラム修正や改良や保守の能率が上がった。

また，ICE使用により，ターゲットマシン上でのデバッグが効率的に行われるようになった。

現在，本システムは適用が始められたばかりであり，使用効果として明確なデータは収集されていないが，現在までのシステム規模，工数等を考慮すると生産性は2倍から2.5倍に上がると期待される。

5. おわりに

本システムはターゲットとしてシングルCPUをデバッグの対象としており，マルチCPUの対応はなされていない。今後，マルチCPUもデバッグできるよう，システムを拡張する必要がある。

また，ファイル化サポートと共に，ソフトウェアの版数管理等，ソフトウェアの保守に関するサポートも大きな課題である。

最後に本システムの開発にあたり御指導いただいた開発支援部小林課長，御協力いただいた複合システム部臼倉課長はじめ，開発支援部古城氏，中村氏，他関係各位に感謝致します。

参考文献

[1] 日本電気："PL/I 文法説明書，FGD 04-3"
[2] Kernighan,B.W. and Ritchie,D.M.:"The C Programming Language",Prentice-Hall,1978
[3] Johnson,S.C.:"Yacc - Yet Another Compiler-Compiler",Bell Lab. CSTR 32,1978
[4] Lesk,M.E.:"Lex - A Lexical Analyzer Generatot",Bell Lab. CSTR 39,1975
[5] Johnson,S.C.:"Lint - A C Program Checker", Bell Lab. CSTR 65,1977
[6] Katseff,H.P.:"Sdb: A Symbolc Debugger", UNIX Programmer's Manual 7th Ed.,Vol.2c, 1979
[7] Feldnam,S.I.:"MAKE - A Program for Maintaining Computer Programs",Bell Lab. CSTR 57,1977