マイコンＵＮＩＸの開発と開発環境としてのＵＮＩＸ

米田　潔　藤林　信也　寺本　雅則
日本電気（株）ソフトウェア生産技術研究所

１．はじめに

１６ビットマイクロプロセッサμＰＤ８０８６をメインＣＰＵとして用いたシステムにＵＮＩＸ／ＳＹＳＴＥＭ３版を移植開発した〔１〕〔２〕〔３〕。ＵＮＩＸは、その移植性とソフトウェア開発環境としての有効性が高く評価されている。我々は、この移植をマイクロプロセッサ組込みシステムのクロスソフトウェア開発としてとらえ、ＵＮＩＸのクロスソフトウェア開発環境としての有効性について考察した。そこで、この移植作業の概要とＵＮＩＸをクロス開発に用いた使用経験とを併わせて報告する。

２．８０８６システムの構成

図１に移植対象とした８０８６システムの構成を示す。その主な特徴と、クロスソフトウェア開発に与える影響を以下に示す。

２．１　プロテクション機構

マルチユーザシステムとして高い信頼性を確保するためには、ユーザプログラムをＯＳの管理下の記憶域で走行させ、一定の手続き（システムコール）以外の方法で、他の記憶域や入出力、及びＣＰＵ等の資源を侵犯しないように管理する必要がある。８０８６単体ではこのような管理機能を実現できないので、ＣＰＵ部にハードウェアを追加し、特権、非特権の概念を設けてこれに対処した。

移植されるＯＳが、この機構を用いて非特権をセットすると、ＣＰＵからアクセスできる資源が制限される。このような状態で、通常の８０８６用のデバッガを動作させるとデバッガがプロテクションに妨げられ正しく動作しないことが想定される。

２．２　メモリマッピング機構

ＵＮＩＸでは、ユーザプログラムのスワップやデータ領域の拡張のため、動的にメモリの再割付が行なわれる。これを積極的に助けて動作効率を上げるために、メモリがどのアドレス上にとられてもＣＰＵから見た論理構造が同じに見えるように、ＣＰＵ部にアドレス変換機構を設けた。

このことによって、論理空間という概念が生じた。そして、アドレス変換機構を動作させると、ＣＰＵ或いはデバッガが認識し制御するメモリの同じアドレスのメモリ領域がメモリマッピングの状態によって変化する。さらにある瞬間において、論理空間上にマッピングされていないメモリ領域は参照できなくなる。

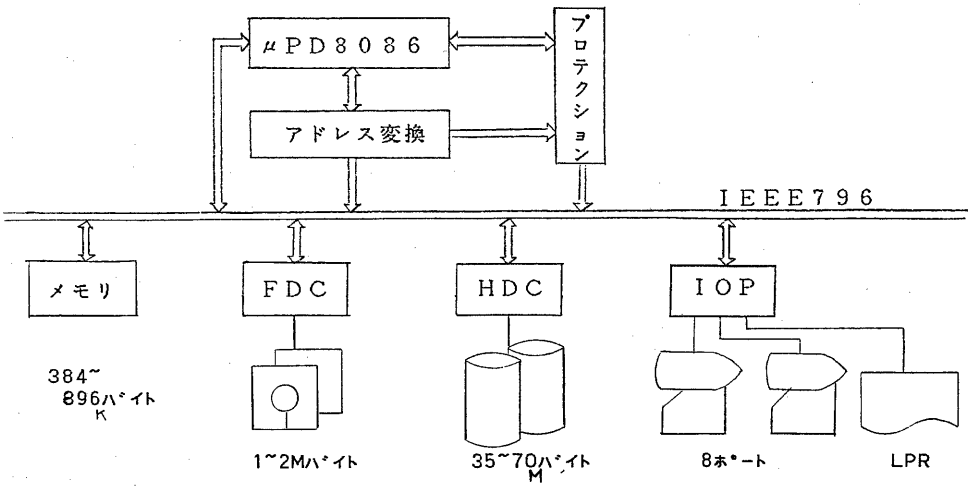

図１　８０８６システムの構成

2．3　デバイスコントローラ

ハードディスク、フロッピーディスク、端末、プリンタの制御にそれぞれ8ビットマイクロプロセッサを充ててシステムの機能分散を計り、メインＣＰＵの負担を軽減した。

これらのコントローラとメモリはＩＥＥＥ７９６バス上に実装されており、これらの間のデータ移送は、マッピング機構やプロテクション機構を通さずに物理空間で行なわれる。従って、開発段階でＯＳ中のこれらのコントローラのドライバルーチンに問題があったときの影響が大きい。

これらの3項目は、いずれもデバッグの環境や方法の選定に対して配慮すべき点である。

3．移植作業の概要

3．1　移植環境の確立

ＵＮＩＸの移植環境には図2に示す機器構成を用いた。

クロスソフトウェア開発用には、ＤＥＣ社のＶＡＸ１１／７８０上のＵＮＩＸ／４．１ｂｓｄ版を利用した。これと８０８６用のインサーキット・エミュレータ(ＩＣＥ)がオンラインで接続されている。また、ＶＡＸ１１／７８０と８０８６システムの共通の媒体としてはフロッピーディスクと端末用通信回線を用意した。これらは開発用システムで作成された８０８６用のカーネル、コマンドプログラムなどを移送するために用いることができる。

一方、ＰＤＰ１１／４４には移植の対象であるＵＮＩＸ／ＳＹＳＴＥＭ３版を走行させ、詳細な動作の確認に用いた。

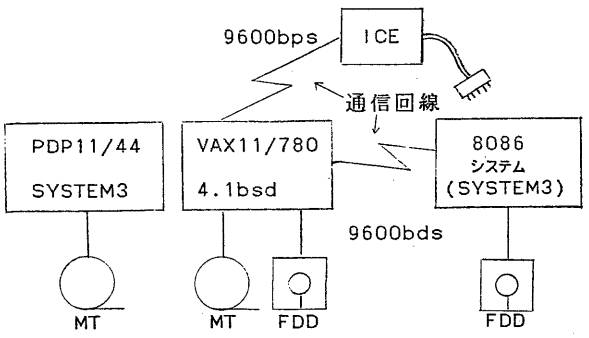

図2　機器構成

3．2　クロス開発ツールの開発

ＵＮＩＸはそのほとんどがＣ言語で記述されている。したがって、移植に先だって、開発対象システムで動作するオブジェクトコードを生成する、Ｃコンパイラ〔4〕、アセンブラ〔5〕、リンカを開発した。これらを用いて生成した８０８６システム用のオブジェクトコードは、論理空間で動作するものである。

またこれらの開発ツールは、８０８６ＵＮＩＸのセルフ開発用コマンドとして、それ自体が移植される。

3．3　ファイルシステムの構築

ＵＮＩＸは、プログラムの実行時に、ディスクなどの2次記憶媒体からプログラムをメモリ中にロードしてきてから実行する。このようにＵＮＩＸは2次記憶支向型のＯＳであり、この種のＯＳを開発するには、先ずＯＳの走行のベースとなる2次記憶中に、必要なプログラムやデータをファイルとして準備しておく必要がある。また、ＵＮＩＸではこれらのファイルは2次記憶中で、ファイルシステムと呼ばれるｔｒｅｅ構造化された論理構造の中に格納されていなければならない。

3．4　データ移送方法の確立

開発用システムで作成された８０８６システム用のカーネル、コマンドプログラム等を、８０８６システムに移送する手段として、我々は、

- ＩＣＥを介した通信回線
- フロッピーディスク
- 端末用通信回線

を用意した。

これらを用いて実際にデータの移送に供するには、これらの媒体を経由するときのデータのフォーマット、或いはプロトコル形式、及び、送り側、受け側のソフトウェアを準備しなければならない。

3．5　検証手段の確立

移植したプログラムがはじめから正常に動作することは現在の技術ではなかなか期待できない。マイコンシステムの開発において、開発したプログラムが正常に動作しない要因としては

- プログラムが仕様を満足していない
- マイコンシステムのハードウェアの異常
- コンパイラ、アセンブラの異常
- 検証手段の異常
- その他

が多く考えられる。従って検証手段はこれらの要因が切分けられるものでなければならない。また、確かにそれが正常であるというためには、そのための検証手段が必要である。

以上が整って、はじめて、ＵＮＩＸのカーネルやコマンド、等の移植が可能になる。

4．開発環境としてのＵＮＩＸ

4．1　開発環境としての条件

プログラム開発環境、或いは移植開発環境として備えるべき条件として次のものが考えられる。

・　作業環境の設定ができ、作業手順の単純化が計れること。

すなわち、作業目的は明確であるから、作業者がその目的に向って専念でき、直接に目的に結びつかない作業や学習にわずらわされないことが望ましい。たとえば、豊富なツール群があって、どのような作業からの要求にも機能的には対応できたとしても、作業例から見ると要求とツールが１対１には対応しておらず使いづらい場合がある。ファイル内容をフロッピーディスクに出力、又はフロッピーディスクから入力したいときに、ＵＮＩＸではｔａｒコマンドが利用できる。ｔａｒコマンドはもともと汎用のコマンドであり、このコマンドをこのような目的で効率よく用いるには多くの動作オプションをつけて、次のような指定になる。

出力：　tar -cvfb /dev/rxxx 20 ファイル名
入力：　tar -xvf /dev/rxxx

ここで／ｄｅｖ／ｒｘｘｘ　はフロッピーディスク装置を示すファイル名であるが、これが各ＵＮＩＸシステムで異なる名前である。しかし、作業者の目的はファイルのフロッピーディスクへの入出力であるから、次のような目的と対応するコマンドがあればよい。

出力：　tarout ファイル名
入力：　tarin

ＵＮＩＸでは、これがシェルコマンドテキストを用いることで可能である。シェルコマンドテキストでは、複数のコマンドを順次実行したり、条件分岐したり、文字列を合成したりできるので、この機能を利用して、既存のコマンドをかなり目的に合った形に合成し作りかえることができる。

開発用ツールが体系を成していること

エディタや言語処理系（コンパイラ、アセンブラ）は必須ツールであるが、それらとデバッガ、版管理ツール、ライブラリ化ツール、構成管理ツールなどが存在し、それらが有機的に結合されていることが、より高度な開発環境を提供する。

ＵＮＩＸには、ライブラリ化ツールとしてａｒがある。これは本来、複数のファイルを１本のファイルにまとめてコンパクト化するツールであり、対象とするファイルの形式に制限はない。このａｒを応用して、アセンブラ（ａｓ）から出力されたオブジェクトファイルをライブラリ化することができる。

リンカ（ｌｄ）はアセンブラから出力された複数のオブジェクトファイルのリンクをとるツールであるが、ａｒのコンパクト化のルールを知っていて、ライブラリ化されたファイルの中から目的のオブジェクトファイルを抽出しリンクすることもできる。

また構成管理ツールとしてＵＮＩＸにはｍａｋｅがある。一般に、目的のファイルを構成する手順は図３に示すようなものである。この手順をｍａｋｅの入力ファイル（Ｍａｋｅｆｉｌｅ）に記述しておくと、ｍａｋｅはＭａｋｅｆｉｌｅ中の入出力関係にあるファイル間の更新時間関係を調べることにより、必要な手順のみを抽出して順次実行する。このｍａｋｅもａｒのルールを知っているから、図３の例をとるとＡ．ｃが変更されていると、Ａ．ｏ、ｌｉｂｘｘ．ａ、ｘｘが順次更新される。

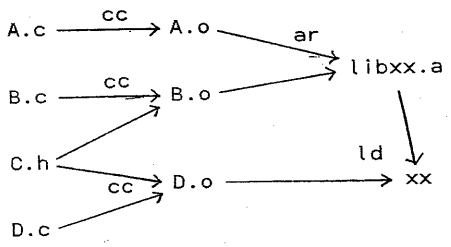

図３　構成手順

ＵＮＩＸ／４．１ｂｓｄ版ではＣ言語のレベルでデバッグができるデバッガ（ｓｄｂ）がある。ｓｄｂを用いるにはＣコンパイラ（ｃｃ）に指示して、デバッグ用の情報を出力させなければならない。しかしこれはｃｃとｓｄｂの間の約束であって、ａｓ、ｌｄ、ａｒ、及びｍａｋｅなどには影響を与えない。

ＵＮＩＸでは生成されたオブジェクトファイルには通常シンボル情報が付いている。このままでもプログラムの実行はできるが、ファイルのサイズを削除したり、シンボル情報をリポートしたりするためのコマンド（ｓｔｒｉｐ，ｎｍ）が、このツール系には必要である。

・　ファイルのバックアップが容易なこと

開発時に誤って重要なファイルを消してしまうことが意外に多い。この様な事故に備えてバックアップファイルを用意しておくとよいが、これが、負担に感じない程度の作業であることが望ましい。これについては次に述べる。

４．２　ファイルのｔｒｅｅ構造の利用

４．１で、開発環境としての条件について述べたが、ＵＮＩＸを開発環境として用いる場合、ＵＮＩＸのファイルのｔｒｅｅ構造を利用することにより、

・　作業の単純化、作業手順の明確化
・　目的に合ったコマンドの作り易さ

がさらに向上し、

・　ファイルのバックアップの単純化

も計れることが期待される。

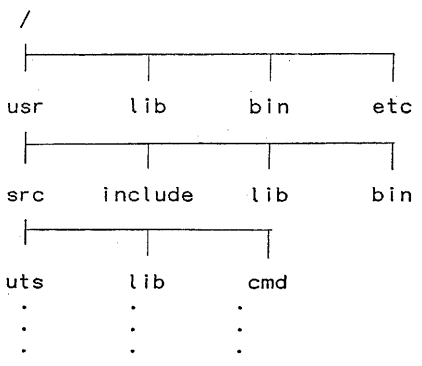

図４　ＵＮＩＸのｔｒｅｅ構造

移植対象ＯＳであるＵＮＩＸ／ＳＹＳＴＥＭ３のカーネル、ライブラリ、及びコマンド関係のファイルの構造は図４のようになっている。

ソースコードは

／ｕｓｒ／ｓｒｃ　以下

ライブラリは

／ｌｉｂ　と　／ｕｓｒ／ｌｉｂ　の下

オブジェクトとコマンドテキストは

／ｂｉｎ，／ｅｔｃ　と／ｕｓｒ／ｂｉｎの下

のファイルにそれぞれ格納されている。

これと同一構造のｔｒｅｅを図５に示すように開発用ＵＮＩＸの移植作業用ノード（ディレクトリ）の下に、３つ展開した。それぞれのｔｒｅｅは、

・　システム３のオリジナル版の格納用（Ｓ３）
・　８０８６クロス開発用（８６／ｒｏｏｔ）
・　８０８６版管理、バックアップ用（８６／ｓｃｃｓ）

に使用する。

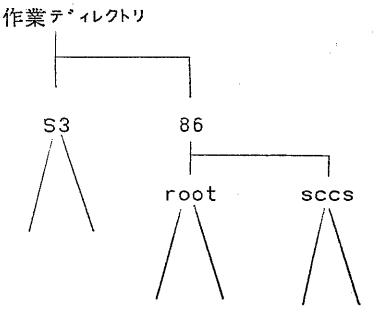

図５　開発システムの作業環境

この３つの同一のｔｒｅｅ構造を採用すると、コマンドのクロス開発の場合には、図６に示した流れになった。移植するコマンドのオリジナルソースコードは　①　に格納されている。これをＣＰ　コマンドを用いて，ｓｃｃｓ（版管理ツール）で管理するためのヘッダ`ｓｃｃｓｉｄ`を付けて　②　にコピーしてくる。次にすぐに　ＡＤＭＩＮ　コマンドによって　③　にｓｃｃｓ管理用の登録を行なう。これによりこのソースコードには１．１という版数が与えられる。以降、この版数は、ＧＥＴＥ、ＤＥＬＴＡ　を繰り返すことで更新される。　③　から　②　へは　ＧＥＴ

E または GET で持って来れる。 GETE は修正用でファイルは書込許可状態になる。GET はコンパイル又は リスト用で書込禁止のファイルを作る。また GET で持って来たファイルのｓｃｃｓｉｄ部は、版数、登録年月日などの可続文字列になる。 ③ のファイルには登録からの全ての版がコンパクトに納められるが、GETE と GET は常に最新版を ② に持って来る。

このようにしてｓｃｃｓ管理を行ないながら改造を完了した ② のファイルを、② の位置でコンパイルしてオブジェクトファイルを得る。これを INSTALL によって ④ に移し、TAROUT、TARIN コマンドによって ④からフロッピーディスクを介して ⑥ にダウンロードする。

以上のように作業環境の設定と作業の流れを規定すると次のような効果があった。

- CP、ADMIN、GETE、DELTA、GET などのような所定の手続きを簡単にするシェルコマンドテキストが作り易い。

これらのコマンドは、いずれもシェルコマンドであり、目的のファイル名だけを指定する。またこれらのコマンドは全て図６の ２ の位置において、ディレクトリを移動することなく使用できる。

CP と DELTA のシェルコマンドテキストを例として図７に示す。

```
CP:
(echo 'static char sccsid[]="%Z%%M% %R%.%L%;'
 cat `pwd | sed s:86/root:s3:`$1) > $1

DELTA:
delta `pwd | sed s:root:sccs:`/$1 $1
```

説明:
echo は,'...' を出力する。
cat は,ｱｰｷｭﾒﾝﾄで示されたﾌｧｲﾙの内容を出力する。
pwd は,現在のﾃﾞｨﾚｸﾄﾘを出力する。
sed s:AAA:BBB: は,入力中の文字AAAをBBBにかえる。
delta file1 file2 は,file2でsccs管理用ﾌｧｲﾙfile1を更新する。
`pwd | sed s:86/root:s3:`は, 2 のﾃﾞｨﾚｸﾄﾘの位置で使用すると, 1 のﾃﾞｨﾚｸﾄﾘ名に置き替わる。
`pwd | sed s:root:sccs:`は, 2 のﾃﾞｨﾚｸﾄﾘの位置で使用すると, 3 のﾃﾞｨﾚｸﾄﾘ名に置き替わる。
$1 は,ﾌｧｲﾙ名に置き替わる。

図７ CP と DELTA コマンド

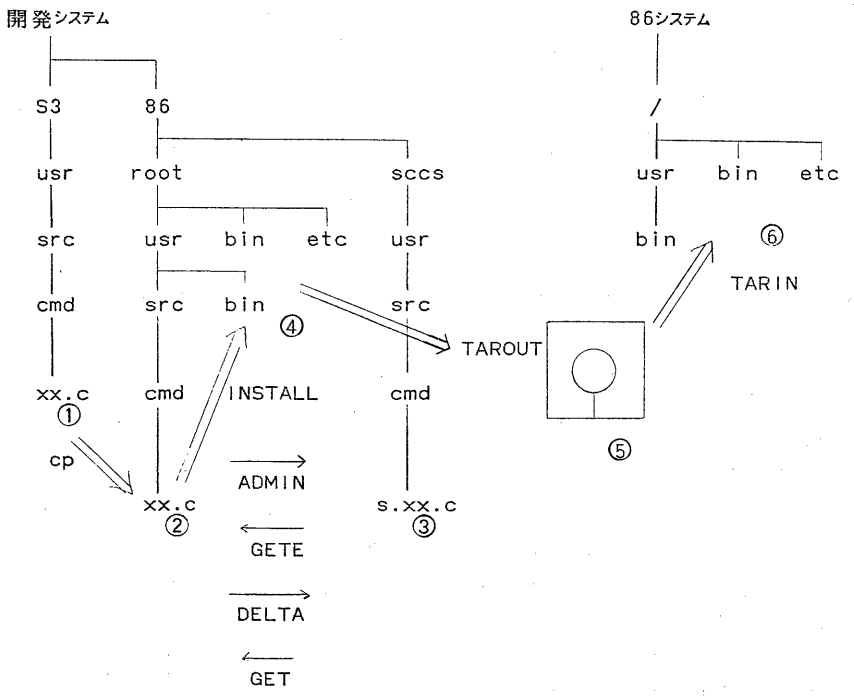

図６ コマンドの移植手順

・　ＳＹＳＴＥＭ３のオリジナルの構成管理（ｍａｋｅ）用の手順を記述したファイル（Ｍａｋｅｆｉｌｅ）がほとんどそのまま使用できる。

カーネルやライブラリは、ｔｒｅｅの階層構造を利用して、ソースコードのモジュール化を行なっている。これに従って構成管理もモジュール単位に分割されており、これらはオリジナルと同一のディレクトリ構造にすることによって無修正で連結して用いることができる。

・　ファイルのバックアップを簡素化できる。

開発時には、作業者は図６の　②　の位置のように８６／ｒｏｏｔのｔｒｅｅの中にいる。このとき、誤ってファイルを消すとすれば、このｔｒｅｅの中ファイルであるが、これらはｓｃｃｓを用いた版管理を行なっているので、８６／ｓｃｃｓのｔｒｅｅの中の対応するファイルの版管理ファイルからすぐに復帰できる。また、このようにバックアップを兼ねた版管理ファイルが８６／ｓｃｃｓのｔｒｅｅに集中しているので、磁気テープなどへの吸い上げも、８６／ｓｃｃｓのｔｒｅｅのみを対象するだけでよい。

．３　クロスツールの作成

４．１で述べたように開発用ツールが体系的であることが高度な開発環境を与える一要因となる。これは、クロス開発でも同様である。そこで、クロスオブジェクト作成ツールを含めたクロス開発系をいかに得るかが、クロス開発環境作りのポイントになる。この点においてもＵＮＩＸを開発用に用いることは有効であった。一般のクロスソフトウェア開発では、オブジェクトファイルやライブラリファイルの制御ブロックには特別な制約を受けない場合が多い。そこで、このような場合には、これらの制御ブロックを開発用のＵＮＩＸと同形式にしておけば、ＵＮＩＸの開発ツール系がそのまま流用でき、即ちにクロス開発系が実現できる。

我々の移植作業では、開発用ＵＮＩＸと、移植対象であるＵＮＩＸで版が異なったためこれらの制御ブロックを同形式にはできなかった。したがって、再構成が必要であった。そこで、我々は逆にＳＹＳＴＥＭ３版の開発系を開発用ＵＮＩＸである４．１ｂｓｄ版に移植した。これは、両者がＵＮＩＸであったので、図８のようにソースファイルを選択してコンパイルするだけで無修正で再構成ができた。

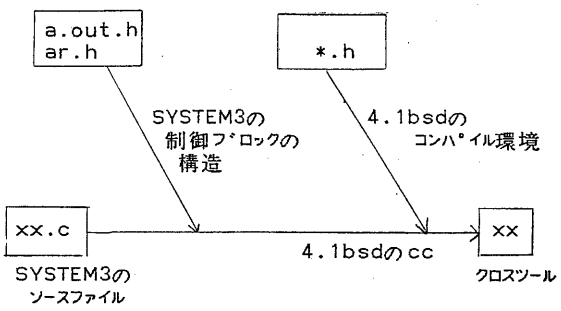

図８　クロスツールの作成

５．データの移送

５．１　ファイルシステムの構築

移植初期段階においては、先ず移植対象システム側でのファイルシステムの構築手段を得なければならない。これには、大きく分けて次の２通りの方法が考えられる。

・　開発システム側でファイルシステムを構築し、移植対象システムにファイルシステムごと移す。

・　移植対象システム中にファイルシステムの構造を作り、そこに個別にファイルを移して格納する。

我々の環境では、図２に示すように、開発システムと移植対象システムとの共通媒体として次の３つが利用できた。

・　ＩＣＥを介した通信回線（９６００ｂｐｓ）

・　フロッピーディスク

・　端末用の通信回線（９６００ｂｐｓ）

しかしこれらの媒体はいずれも大容量のデータ転送には向いていない。したがって、上の２つの構築方法は開発効率の面から考えて決定した。すなわち、初期の、カーネルや最初に起動されるプロセスの検査段階において、単一のファイルが頻繁に更新されることから考えて、後者の方法、つまり１回のデータの移送時間が短かくなる、個別にファイルを移す方法を採った。

５．２データ移送方法

５．１で述べたように、移植初期段階において、我々はファイルシステムを移植対象システム中で

構築する方法を採った。従って、これに応じたソフトウエアが移植対象システム中に必要になった。これは具体的には、ファイルシステム作成プログラム(ｍｋｆｓ)やコピープログラム(ｃｏｐｙ)などである。一方媒体は、作業効率の高いオンラインダウンロードを重視して、ＩＣＥを介した通信回線を利用した。このため、開発システム側に、ＩＣＥのダウンロードの形式とプロトコルに応じたプログラムを作成した。これらを用いて、カーネルや基本コマンドを移送した。

開発の後半、すなわち本格的なコマンドの移植及び検査段階では、移植対象システム上で稼動したＵＮＩＸと、フロッピーデイスクや端末用の通信回線を利用した。フロッピーデイスクは前述したｔａｒコマンドを利用し、端末用通信回線はＵＮＩＸ間通信用コマンドであるｃｕやｕｕｃｐが利用できた。このため、後半の作業段階では特にデータ移送用のツールを準備する必要はなかった。この例のようにＵＮＩＸはもともと他のシステムとの通信コマンドを持っているので、これらの利用できる可能性を検討して見るとよい。またフロッピーデイスクなどはＵＮＩＸからはファイルに見えるので、形式変換の応用プログラムは比較的簡単に作ることができる。

６．検証手段

マイクロプロセッサ応用機器のプログラム開発にＩＣＥを用いることは、既に一般化している。ＩＣＥはおおむね次の機能を持っている。

- プログラムの実行と停止
- メモリ空間、Ｉ／Ｏ空間、ＣＰＵ状態の参照と変更
- ブレイクポイントの設定
- プログラムの実行のトレース
- 逆アセンブル

が、ＩＣＥの最大の特徴は開発機の機能に制限を与えないことである。つまり、ＩＣＥのこれらの機能の実現のために、開発機のメモリ空間や割込回路に影響を与えない。これは、我々の８０８６システムのようにｃｐｕ部分にプロテクション機能やアドレス変換機能を設けたシステムの開発に対し極めて有効であった。アドレス変換を行なうことでｃｐｕは論理空間上で動作することになり、ＩＣＥも論理空間しか見えなくなる。しかし、実行するプログラムも論理空間上で動作するので矛盾は生じない。

マッピングされていないメモリ領域が参照できない問題については、そのような要求が起こるのが、特権モードのときが多かったのでＩＣＥを用いて一時的にマッピングをかえて参照することができた。

また、追加したプロテクション機能によって、逆にプログラムのバグやその他の障害による異常な動作が検出されたので、しばしば開発を助けられた。

またハードウエアの障害による異常の切分けと検証用にはロジックステートアナライザを用いた。

７．むすび

以上で述べて来たように、ＵＮＩＸはクロス開発環境として、環境設定のし易さ、開発ツール系の組み立て易さ、及び通信手段が存在することなどからその有効性が確認された。

一方、検証段階で用いるＩＣＥなどと開発用として用いたＵＮＩＸとの結合度が弱い点が指摘される。我々の環境ではＵＮＩＸとＩＣＥとがオンラインで接続されてはいるがソフトウエアの面から見ると、他のクロス開発ツール系と有機的に結合されてはいない。

シンボルテーブルのＩＣＥへのダウンロードによってアセンブラレベルでのシンボリックデバッグは可能であったが、記述言語でのデバッグやシェルコマンドテキストなどの利用によるＩＣＥの操作性の向上などにより今後新しいアーキテクチャに対しても一層容易で信頼性の高い開発作業が可能となることを期待したい。

参考文献

〔１〕 藤林他："ミニコン/マイコンのUNIX移植"，第27回情処全国大会4G-3(1983.10)

〔２〕 福村他："UNIXのミニコンピュータMSへの移植"，第27回情処全国大会4G-4(1983.10)

〔３〕 米田他："UNIXのマイコンへの移植"，第27回情処全国大会4G-5(1983.10)

〔４〕 佐治他："マイコンUNIX移植のためのCコンパイラの開発"，第27回情処全国大会2E-5(1983.10)

〔５〕 堂田他："マイコンUNIX移植のためのアセンブラの開発"，第27回情処全国大会4E-1(1983.10)