ソフトウェア工学　48－1
（１９８６．６．２５）

ＯＳ／omicron用言語Ｃコンパイラｃａｔの開発
－VAX/UNIXクロスシステムからの移行－

屋代　寛、森　岳志、並木美太郎、中川正樹、高橋延匡
東京農工大学　工学部　数理情報工学科、+）現在、㈱日立製作所　基礎研究所

本稿は、ＯＳ／ｏ用言語Ｃコンパイラｃａｔの開発について述べている。ｃａｔの開発は、1984年春に始まり、開発当初はVAX/UNIX上のクロスコンパイラとして開発を行なった。クロスコンパイラの開発は、1985年３月に完了した。次に、このクロスコンパイラをMC68000システム上にセルフコンパイラとして稼働させるための作業が進められた。この作業は、第２版ｃａｔの開発と並行して進められ、1986年１月には第２版ｃａｔがセルフコンパイラとして稼働している。

本稿は、コンパイラの開発方式、その開発において行なったツールの作成、それを通して得られたソフトウェア開発の教訓についての報告である。

"Development of a Language C Compiler : CAT for OS/omicron - Transition from the VAX/UNIX Cross System to the MC68000 Target System -" (In Japanese)

HIROSHI YASHIRO, TAKESHI MORI, MITAROU NAMIKI, MASAKI NAKAGAWA, NOBUMASA TAKAHASHI
Department of Infomation Science, University of Tokyo Agriculture and Technology
(the third author currently at Advanced Research Labratory, Hitachi Ltd.)

This paper describes practice and experience associated with the development of a Language C compiler CAT for OS/omicron, which started in Spring 1984. OS/omicron is an operating system for a super personal system of MC68000 base. First, the cross version of CAT was developed and made available on VAX/UNIX in March, 1985. Next, the project embarked on the development of its resident version on OS/omicron by transporting the object code of CAT tranlated by the cross CAT. The resident CAT began to be used in January 1986. CAT has been developed completely by ourselves without using such tools as YACC and PCC on UNIX.

１．はじめに

計算機システムを取り巻く環境は常に変化している。特にハードウェア環境は、この十年間で目覚ましい発展を遂げている。マイクロプロセッサや半導体メモリ、周辺LSIのコストパフォーマンスは半導体技術の進歩によって著しく向上してきた。また、光ディスクや光ファイバなどの次世代技術も着々と実用化されてきている。同時に、計算機システムを利用するアプリケーションも多様化してきている。計算機システムの歴史は汎用大型の道を歩んできたが、ハードウェア環境の革新、そしてアプリケーションの多様化によって、計算機システムの小型専用化の方向性を明確にしてきている。過去のしがらみにとらわれることなく成長を続けてきたワードプロセッサやパーソナルコンピュータなどの分野では、この傾向が顕著に現れている。今こそ、時代のニーズに応じた新しい計算機システムのアーキテクチャを研究する時期を迎えている。この転換が遅れれば遅れるだけ計算機システムの社会への浸透は妨げられるであろう。

我々は、以上の認識をもとに、コストパフォーマンスの良いハードウェア資源を豊富に利用し、アプリケーション（特に、日本語情報処理、パターン認識、人工知能）を意識した計算機システムの研究・開発を行なってきた。

現在、我々はこの研究用計算機システムのためのオペレーティング・システムＯＳ／ｏ（omicron）を開発中である。プロセッサには16ビットマイクロプロセッサMC68000を使用し、16Mバイトのアドレス空間をフルに利用した実記憶方式を採用している。実記憶方式を採用した理由は、オンライン・リアルタイム処理を行なうアプリケーションに有利であることによる。システム記述言語には言語Ｃを採用しており、1985年春には、“自作”のクロスコンパイラを東京大学大型計算機センター（以下、東大センターと記す）のVAX/UNIX上で稼動させている。以後、VAX/UNIXクロスシステムおよびターゲットシステム上に各種変換ツールを作成し、この言語Ｃコンパイラ（ｃａｔ：C compiler developed at Tokyo Univ. of Agriculture and Technology）をターゲットシステムに移植した。本稿は、ｃａｔ開発における Software Practice & Experiment の報告である。

２．開発の歴史

2.1 背景

東京農工大学数理情報工学科（以下、当学科と記す）では、パターン認識・人工知能のための並列処理の研究を行なうため、1980年にプロジェクトPIEを発足させた。そこでは問題解決向きのマルチマイクロプロセッサシステムの開発を目指している。このマルチマイクロプロセッサシステムをsystem／π、そのＯＳをＯＳ／πと呼んでいる。ＯＳ／ｏは、その前段階としてのシングルプロセッサシステム用ＯＳであり、ＯＳ／πを開発するためのＯＳとして位置付けられる。したがって、ＯＳ／ｏは開発環境の整備とシングルプロセッサ・マルチタスクによる評価データの収集を行なうことを一つの目的としている。しかし、ＯＳ／πは専用処理バックエンドプロセッサのＯＳであるため、システム資源全体の統括、ファイルシステム、人間とのインタフェースなどはＯＳ／ｏが基本となる。したがって、ＯＳ／ｏは並列処理をＯＳ／πに委託するとしても、その他の面では大学の研究室レベルの研究をサポートできる研究用スーパーパーソナル計算機のＯＳとして機能する必要がある。つまり、前章で述べた研究を容易に行なうことができる環境を提供しなければならない。そこで、アプリケーション指向のＯＳとして、以下の項目を設計方針とした。

（1）マルチタスクの機能を実現する

ＯＳ／πにおける完全な並列処理への移行を考慮し、またＯＳ／ｏにおけるプログラミング効率の向上のためにマルチタスク機能を実現する。プログラミング環境としてのマルチタスクの有用性は言うまでもない。また、人工知能の問題としては、問題を解決する場合に、木構造の探索問題に帰結できるものが多い。この問題解決をマルチタスクとして記述し、ＯＳ／ｏで擬似的な並列処理として実現し、プロセスの生成・実行状況をモニタする。

（2）システム記述言語として言語Ｃを用い、これを基底言語とする

ソフトウェア開発を行なう際、生産性や保守性、移植性を向上させるためにも、アセンブリ言語ではなく高水準言語で記述するのが望ましい。特に、大学の研究室では数年で担当者が卒業してしまうため、ソフトウェアの保守性は死活問題である。ＯＳ／ｏでは記述能力の高さなどからシステム記述言語に言語Ｃを採用した。そして、アプリケーション言語の処理系は言語Ｃで記述し、複数の言語で記述されたプログラムの相互利用をはかる。

（3）オブジェクト・モジュールはＯＳ／ｏ、ＯＳ／πで共通とし、リロケータビリティとリエントラントラビリティを保証する

ＯＳ／πではベースレジスタの動的変更が生じるため、実行中にオブジェクトモジュールをリロケーションすることを想定しなければならない。このため、動的なリロケータビリティが要求される。また、並列処理においては、同一の手続き部を複数実行することが考えられる。複数実行する手続きをメモリ空間上で一つにまとめるためには、プログラムをリエントラントなものにすればよい。一方、単一プロセッサのＯＳ／ｏにおいても実記憶方式でマルチタスクを実現するためには、これらの保証があることが望ましい。タスク管理、メモリ管理のために柔軟性と可能性が得られる。

（4）プログラムをROM化して利用する

頻繁に使用されるコンパイラなどの言語処理系をはじめとするシステム・プログラムやＯＳの核をROM化することにより、ディスクからのロードを省き、他のプログラムの暴走等から保護することができる。

（5）日本語情報処理の研究のために２バイトの漢字コードを標準とする

既存のＯＳで用いている文字コードは、通常１バイトコードを採用している。これは、既存のＯＳが欧米からの輸入品であり、文化の基盤となる文字も輸入してしまっていることに他ならない。しかし、日本語情報処理を行なう場合、１バイトコードだけでは不十分であることは明らかである。そこで、モードを切り換えたり、エスケープ文字を使用して１バイトコードと２バイトコードを混在させるなどして漢字を表現しているが、そのためにアプリケーションプログラムの作成に厄介な問題が生じている。小さなプログラムを作ってみたら、エスケープ文字に対する状態遷移の処理が

半分以上を占めていた、などということは日常茶飯事である。ＯＳ／ｏでは文字コードに２バイトコードを標準とし、アプリケーションプログラムの作成をこの種の非本質的な問題から解放する。

ＯＳ／ｏでは、以上の設計方針を定めたわけであるが、すぐさま実現に取りかかれたわけではない。まず、“自作”のハードウェア上でソフトウェアの開発を行なえる環境を整備することから始めた。これは、システムの開発を白紙の状態から行ない、その中で得られる知見をプロジェクトの足掛かりとするためである。多少、遠回りな方法ではあるが、既存の計算機システムのＯＳをはじめとする各部分が“ブラックボックス”化している現状を顧みたとき、自分達のための計算機システムを作るためには、全て自分たちの手で作らなければならないという認識に立った結果である。しかし、これはクロスシステムや既存の計算機システムをツールとして使用することを否定するものではない。ただ、最後にＯＳ／ｏあるいは、あそのユーティリティとして動作する部分にブラックボックスは作らないということである。

その第一歩として、我々はMC68000の発表直後から当学科所有の中型計算機ACOS-600上でMC68000シミュレータやMC68000クロスベーシックアセンブラ、MC68000用ＣＡＩを開発し、自作のシングルプロセッサシステムを開発した。

2.2 設計

ｃａｔの設計は、1982年春に始まった。まず、ｃａｔに要求されたことは、システム記述言語としてハードウェアの性能や特徴を十分に記述できる能力を持たせることであった。同時に、言語ＣはＯＳ／ｏの基底言語としてアプリケーションへの利用も想定される。そのため、ｃａｔの設定するプログラム実行環境は、事実上、ＯＳ／ｏのプログラム実行環境を規定する。以上の点から、設計の段階で問題となったのは、

（１）言語仕様
（２）オブジェクト・コードの仕様
（３）プログラム実行環境

である。まず、（１）に対しては、高水準言語レベルでの互換性を考慮して言語Ｃ参照マニュアル［１］に準拠することにした。ただし、register変数はauto変数扱いになることや、ビットフィールドを削除したことなどの相違点がある［２］。

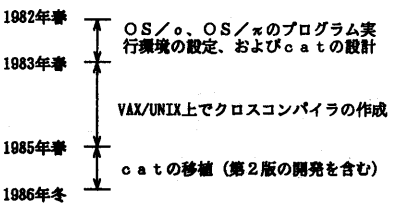

図１．ｃａｔ開発の歴史

次に、（２）と（３）に対しては、オブジェクト・コードをリロケータブルかつリエントラントなものとすることにした。具体的には、手続き・データのアクセスをすべてベース・レジスタからの相対の変位で行ない、分岐命令はプログラムカウンタ相対を用いている。ベース・レジスタには、MC68000の豊富なアドレス・レジスタを利用している。これにより、動的なリロケーションも可能となっている。MC68000にはサブルーチン呼出しのための高機能な命令（jsr, link, unlink等）があるが、これらの命令は使用していない。これらの命令は絶対アドレスを使用するため、動的なリロケーションを行なうことが困難になるからである。このようなプログラム実行環境にしたことにより、生成されるオブジェクト・コードの効率は必ずしも良いものとはなっていない。動的なリロケーションを行なわないならば、絶対アドレスを用いてプログラム実行環境を設定しても良いし、そのほうがオブジェクト・コードの効率は良いであろう。しかし、我々には前述のマルチプロセッサシステムや他の組込みシステムへもシステム記述言語として言語Ｃを使用したいという要求がある。研究目的のシステムにおいて、対象とする計算機システムのアーキテクチャが変わるたびに、コンパイラを変更することはあまり現実的ではない。そこで、我々は一番自由度の高いプログラム実行環境に統一し、あるシステムでは無駄が生ずることがあっても、全体としての生産性、保守性、移植性を追求したほうが研究室レベルでの開発には得策であると考えた。すなわち、考えられ得る計算機システムのアーキテクチャに対して柔軟な対応をとることができ、ライフ・タイムの長いプログラム実行環境を設定することが、オブジェクト・コードの効率を相殺して余りあると確信したのである。

2.3 クロスコンパイラの開発

ｃａｔの開発は、1983年春から始まった。ソフトウェア開発を行なう環境としては、開発当初、クロスソフトウェアとしてMC68000シミュレータとMC68000クロスベーシックアセンブラが存在しており、これらのクロスソフトウェアを使用して、“自作”のハードウェア上にデバッガ等のレジデントソフトウェアを搭載した。しかし、このハードウェアの信頼性がまだ低かったため、ソフトウェア開発用にメーカーからボード単位のMC68000システム（日立製作所SBCシステム）を購入した。同時に、CP/M-68K第１版が送られてきた。このとき、ｃａｔ第１版のコーディングが終了しつつあるときであった。しかし、CP/M-68K第１版のユーティリティとして用意されていた言語Ｃコンパイラは、まだ信頼性の低いものであり、当研究室だけでも数十の虫を観測した。このため、CP/M-68K上での大規模ソフトウェア開発を断念した。

そこで、我々は東大センターのVAX/UNIXを以下の理由から開発システムとして選択した。

（１）ソフトウェア・ツールが充実している
（２）信頼性の高い言語Ｃコンパイラが存在している
（３）共同利用施設として非常に低価格で利用できる

開発手順としては、Earleyが定式化した方法[８]に従った。
①まず、ｃａｔ自身を言語Ｃで記述する。
②開発システム上の言語Ｃコンパイラを用いて①で記述されたｃａｔのソースリストをコンパイルしてｃａｔクロスコンパイラを作成する。
③上記②で作成されたクロスコンパイラを用いて①のｃａｔソースリストをコンパイルする。

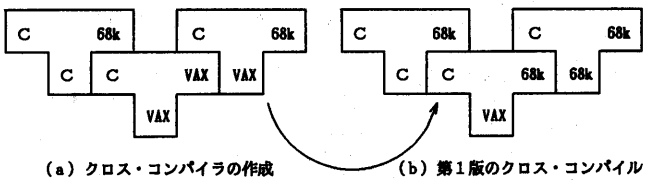

図２．ｃａｔの開発手順

これをまとめたものが、図２である。
以上の方法によって、ターゲットシステム上で稼動するセルフコンパイラが得られる。

1985年春にクロスコンパイラの開発は完了した。この開発には当研究室の修士２名が担当し、プログラム・ステップ数は約２万行にも達した。また、クロスコンパイラによって生成されたセルフコンパイラのプログラム・サイズは、手続き部が290KB、データ部が140KBとなった［２］。

３．ｃａｔの移植

1985年春、ｃａｔはクロスコンパイラとして稼動した。しかし、担当者が卒業してしまったこともあり、そのセルフコンパイラを稼動させる作業（移植作業）は難行した。クロスコンパイラがあれば、理論的には、図２．(b) の方法を用いて、簡単にセルフコンパイラを稼動させることができる［７］。しかし、実際には、以下の問題がある。
（１）移植（転送）経路の確保
（２）ターゲットシステム上でのデバッグ環境
本章では、上記の問題に対して、我々が用いた方法と作成したツールについて述べる。

3.1 移植経路の確保

ｃａｔをクロスコンパイラとして稼働させた時点で、CP/M-68Kのコンパイラもバージョンアップが行なわれて信頼性の高いものとなっていた。そこで、ｃａｔのソースリストをターゲットシステムに転送して、CP/M-68KのＣコンパイラで上記の開発方法を繰り返すことも考えられた。しかし、UNIXコンパイラの識別子が16文字まで有効であるのに対して、CP/M-68Kコンパイラの識別子は有効長が8文字(外部変数は、7文字)までである。名前の短縮化は不可能ではないが、プログラムの可読性を低下させ、処理系の違いで問題が生ずる可能性があるので、この案はしりぞけられた。そのため転送するデータはVAX/UNIX上に生成されたｃａｔのオブジェクトコードとした。

移植を行なう上で最初に考えなければならないことは、開発システム上のセルフコンパイラをターゲットシステムに転送する方法である。転送の方法には、
（１）磁気テープなどの記憶媒体を用いて行なう方法
（２）通信回線を用いて、直接、開発システムとターゲットシステムを結ぶ方法
がある。上記（２）の方法を用いた場合、転送量の多さから通信回線を使用するためのコストがかかる。また、当研究室のターゲットシステムでは、（２）の方法を実現するためのハードウェアおよびソフトウェアがなかったため、我々は（１）の方法を採用することにした。

東大センターのVAX上と他システムとの間で記憶媒体を用いてデータ交換を行なうためには、次の方法がある。
（ｉ）VAX/UNIX上のファイルを磁気テープに直接出力する。あるいは、VAX/UNIX上のファイルへ直接入力する(ただし、ファイル形式は変換する)。これには、UNIXのユーティリティddコマンド［４］を使用する。
（ii）VAXと通信回線で接続されている大型計算機M-280Hへデータを転送し、M-280Hからフロッピーディスクにデータを出力する。VAX～M-280H間の転送には、UNIX上のユーティリティcvosコマンド［５］を使用し、フロッピーディスクへの出力には、M-280Hのオープン入出力装置［６］を使用する。当然、その逆（入力）も可能である。なお、M-280Hの磁気テープ装置を利用することも可能であるが、これは（ｉ）の方法に劣るので考察に値しない。

次に、当研究室のターゲットシステムでは、次の２つの方法でデータ交換ができる。
（iii）ターゲットシステム上のフロッピーディスク装置を用いる。これには、当研究室のツールibmを使用する。このツールを使用することにより、IBM形式のフロッピーディスクを媒体としてデータ交換ができる。EBCDIK→ASCIIコード変換、IBM→CP/M-68Kファイル変換を行なう機能がある。

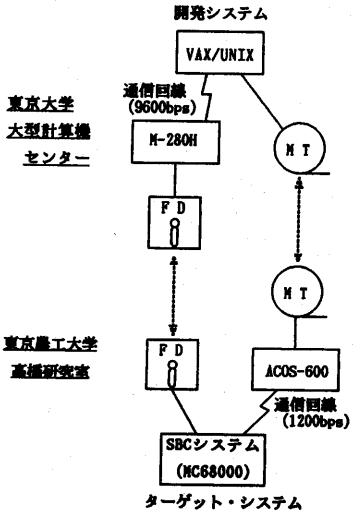

図３．移植経路

（iv）ターゲットシステムと通信回線で接続されている当学科所有の中型計算機ACOS-600の磁気テープ装置を用いる。ターゲットシステム～ACOS-600間の転送には、当研究室自作のツールconnectを使用する。

東大センターのVAX/UNIXシステムと研究室のターゲットシステム間の移植経路の問題は、上述の（ｉ）と（iii）、（ii）と（iv）を組み合わせた２つの方法で解決された（図３参照）。確実な移植経路を確保するため、我々は、この２つの方法を可能とした。

しかし、図３からも判断できるように、２つの方法どちらにも通信回線を用いなければならない。一般的に、通信回線を用いた転送は、キャラクタ型式のテキスト・ファイルに限定される。制御符号等の問題からバイナリ形式のファイルを転送するのは、危険だからである。そこで我々は、セルフコンパイラのオブジェクト・プログラムを転送するために、バイナリ型式ファイルから16進キャラクタ型式のファイルに変換するツールdumpcを作成した。ターゲットシステムでは、このツールで変換されたファイルを再びバイナリ型式のファイルに逆変換する作業が必要である。そのため、これを逆変換するツールredumpもターゲットシステム上に作成した。なお、開発システムのVAXとターゲットシステムのMC68000では、図４に示すようにロングワード、ワードの番地割付けが異なっている（バイト・スワップが起きる）。このため、これらのツールではバイト・スワップに対処する機能を付加している。

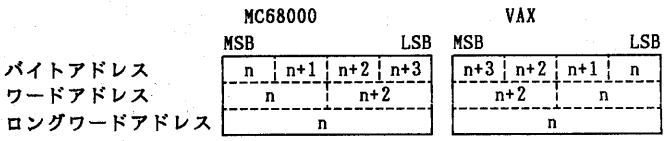

図４．MC68000とVAXのアドレス割り付け

3.2 ターゲットシステムのデバッグ環境

ｃａｔの移植を開始した1985年春には、ＯＳ／ｏ第１版と、その開発に用いたCP/M-68Kがターゲットシステム上で稼動していた。この２つのＯＳでＯＳ／ｏアセンブラとＯＳ／ｏリンケージエディタが稼動していた。本来なら、ｃａｔのデバッグはＯＳ／ｏ第１版で行なうべきである。しかし、ＯＳ／ｏ第１版には、いくつかの問題点（信頼性、ファイル容量の200KB制限等）があったため、CP/M-68Kによってデバッグを行なうことにした。

クロスシステムを用いたソフトウェア開発では、クロスシステムの開発環境だけでなく、ターゲットマシンにおけるデバッグ環境の整備も重要なものとなる。この問題に対処するため、前節（3.1）で述べたツールの他に、以下のツールも作成した。

（１）ツールrmdump　：ｃａｔの出力するオブジェクトモジュールの構成が担当者の誤解によって、ＯＳ／ｏリンケージエディタに入力できない構成になっていた。そのため、正常なオブジェクトモジュールの構成にするツールを作成した。

（２）ツールrmchk　：正常なオブジェクトモジュールのチェックを行なうツールである。通信回線での転送ミスやクロスコンパイラの虫によって、情報が欠落していないかどうかをチェックする。

（３）ツールrmcnt　：プログラムサイズ（手続きやデータの大きさ）を測定する。

（４）ツールfcallstr　：変数や関数がどのモジュールで定義され、その参照関係がどのようになっているかを調べる。

（５）ツールosocpm　：ＯＳ／ｏの実行型式モジュールをCP/M-68Kの実行型式モジュールに変換する。このツールで変換を行なうことによって、セルフ・コンパイラのデバッグがCP/M-68K上で可能になる。

また、セルフコンパイラはＯＳ／ｏの実行環境上で動作するようになっており、CP/M-68Kでこれをサポートするために、以下の実行時ルーチンも作成した。

（６）ライブラリsetup　：CP/M-68K上でＯＳ／ｏのプログラム実行環境をサポートするため、各基底レジスタの初期設定を行なうセットアップルーチンである。

（７）ライブラリio　：CP/M-68Kでファイル入出力を行なうためのライブラリルーチンである。

今まで述べてきたツールの総プログラムステップ数は、言語Ｃのソースコードで約２万行に及び、ソフトウェア開発における環境整備の重要性を物語っている。

3.3 移植の方法

前節（3.1、3.2）で述べたツールによって、実際の移植を行なうことが可能となる。移植の手順は、以下のとおりである（図５参照）。

（１）転送する対象がバイナリ型式のファイルならば、ツールdumpcを用いてキャラクタ型式のファイルに変換する。この際、転送量を後でチェックできるように、キャラクタ数や行数を確認しておく。

（２）東大センターのVAX/UNIXから直接磁気テープに転送するか、または、M-280Hを経由してフロッピーディスクに転送する。このどちらを選択するかは、センターの混雑度によって決定した。

（３）当研究室のターゲットシステムへ、ツールibmまたはツールconnectを使用して転送する。

（４）上記（１）でキャラクタ型式に変換したファイルは、ツールredumpで逆変換する。これにより、セルフコンパイラのＯＳ／ｏリロケータブルオブジェクトモジュールがターゲットシステム上に転送されたことになる。

（５）ツールrmdumpを用いて、誤った型式のリロケータブルオブジェクトモジュールを正常な型式に変換する。

（６）CP/M-68Kで稼動するＯＳ／ｏリンケージエディタを使用し、セルフコンパイラを実行時型式モジュールにする。

（7）上記（6）の実行時型式モジュールは、ＯＳ／ｏ型式であるため、ツールosocpmを使用して、CP/M-68K型式に変換する。

以上の操作が終了することにより、はじめてターゲットシステムでのデバッグを行なうことが可能となる。もし、虫が発見された場合には、東大センターのVAX/UNIX上でクロスコンパイラを訂正した後、セルフコンパイラを再び作成して、（1）の手順からやり直すことになる。

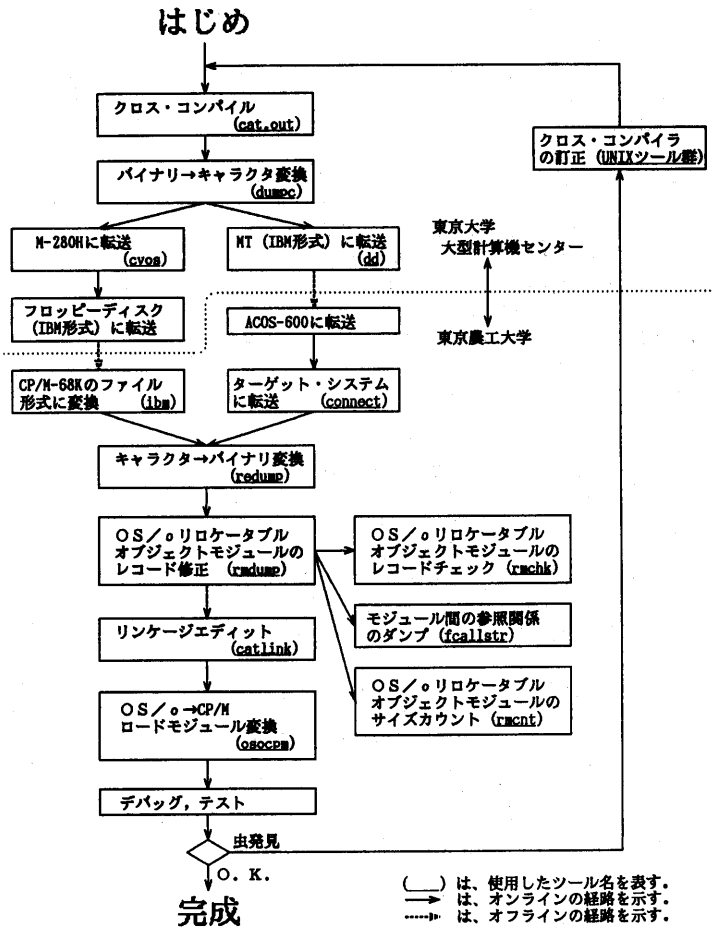

図５．ｃａｔの開発手順

４．再構成

言語Ｃコンパイラｃａｔの移植作業を進める間に、ｃａｔは以下の問題点を抱えていることが判明した。その問題点の詳細は本稿の姉妹編［９］にゆずるが、概略は以下のとおりである。

（１）保守性の問題
（２）拡張性の問題
（３）オブジェクトコードの効率

以上の問題点の他、ｃａｔのマクロプロセッサには、定数置換の機能しかなく、大規模プログラム開発に必要なファイルのインクルード機能や条件付きコンパイルの機能がなかった。

移植作業を進める中でこのような問題が浮き彫りにされた。また、これらの問題によって移植作業さえも難行することが予想された。そこで、我々は移植作業と並行してｃａｔの問題点を解消するために再設計を行なった。再設計の方針を次に示す。（以下では、再設計を行なった後のｃａｔを第２版、それ以前のｃａｔを第１版と呼ぶ。）

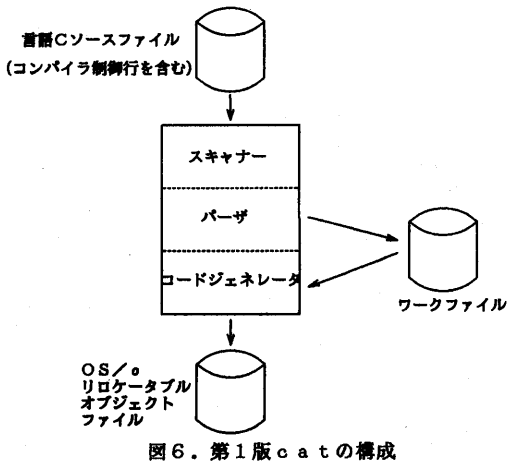

図６．第１版ｃａｔの構成

（1）保守性を考慮してプログラムを各フェーズ毎に分割する
（2）拡張性を考慮して構文解析のフェーズであるパーザは、機種に依存しない応用の効く中間コードを生成する。パーザに対しては、文書化ツールへの応用等幅広い要求がある
（3）パーザには、プログラムの完成度が高い第１版のパーザを改造して使用する。なお、この改造は軽微ですむ

各フェーズの構成を図７に示す。

ｃａｔ移植環境の整備が完了した時点で、東大センターのVAX/UNIX上でｃａｔ第２版のフェーズⅠ（プリプロセッサ）、フェーズⅡ（パーザ）が稼動していた。フェーズⅠは、CP/M-68K上で開発したプログラムを言語ＣソースレベルでUNIXに移植したものである。フェーズⅡは、前述のとおり、中間コードを出力するように第１版ｃａｔを改造したものである。ｃａｔの移植はこの第２版について行なうこととした。そして、第１版の移植は比較評価などの必要が生じた時点で検討することにした。ｃａｔ第２版では、フェーズⅡ以降は新たに作成するので、その開発を信頼性の向上したCP/M-68Kで行なえば、実際に転送するのは、フェーズⅡだけになる。フェーズⅠは前にも述べたとおりCP/M-68K上に存在しているので転送する必要がない。また、フェーズⅡはVAX/UNIX上でかなりのテストが成されているので、図５のループを繰り返す回数は少なくて済むとが期待できた。また、ｃａｔ第１版よりも転送量も少なくなり、そのことから、デバッグ効率の向上が期待できた。なお、フェーズⅡをソースコードでCP/M-68K上に転送して、クロスコンパイル行なわないのは、3.1節で述べたとおりである。

したがって、実際の移植は次の手順で行なうことにした。
（1）第１版ｃａｔクロスコンパイラを用いて第２版ｃａｔのフェーズⅡをクロスコンパイルする。
（2）上記（1）で生成したオブジェクト・プログラムを3。3節で述べた手順でターゲットシステムに転送する（図８参照）。
（3）第２版ｃａｔのフェーズⅡ（中間コードを生成する版）の実機デバッグを行なう。このフェーズでのデバッグ効率を高めるため、中間コードの内容をチェックするツールdbintmも作成した。（4）フェーズⅢ（オプティマイザⅠ）、フェーズⅣ（コードジェネレータ）、フェーズⅤ（オプティマイザⅡ）は、ターゲットシステムのCP/M-68k上で開発することにした。フェーズⅥのアセンブラは、前にも述べたようにCP/M-68k上で作成してあったが、フェーズ分割のために多少の改訂を行なった。

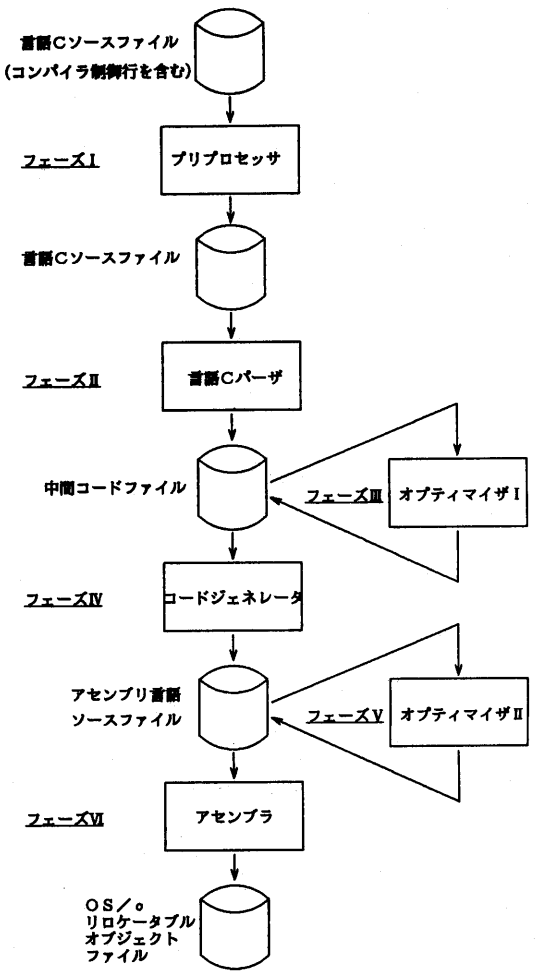

図７．第２版ｃａｔの構成

フェーズⅡの移植が完了した時点で、他のフェーズがターゲットシステム上で稼動していた。そのため、第２版ｃａｔはフェーズⅡの移植が完了した後、すぐに、セルフコンパイラとして稼動した。なお、移植に要した転送量と転送回数、第１版ｃａｔ（クロスコンパイラ）で発生した虫は、次のとおりである。

１回の転送量	約670KB
転送回数	６回
第１版ｃａｔで発生した虫	16ヵ所
内：スキャナ	３ヵ所
パーザ	１ヵ所
コードジェネレータ	12ヵ所

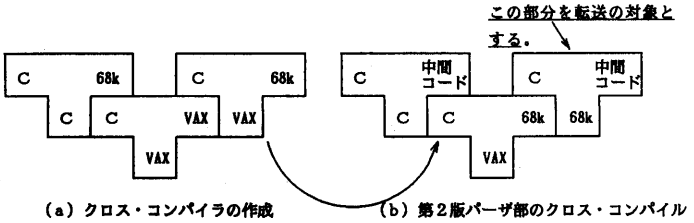

図８．ｃａｔ転送までの手順

第２版ｃａｔは、1986年1月に実機上で稼動した。第２版ｃａｔのプログラム

サイズは、全てのフェーズを合わせると言語Cのソースコードで約3万行に達している。移植・開発には、その環境の整備を含めて約8ヵ月を要し、当研究室の学部4年生2人、修士3人計5人がこの作業を担当した。

6．考察

ｃａｔ開発で得られたことをまとめると次のようになる。

（1）開発環境の重要性

ｃａｔ開発では、相当の労力を開発環境の整備に費やした。これは、ソフトウェア開発が目的とするプログラムの開発だけではなく、その開発を支援するツールなどの開発環境を整備することも含んでいることを意味している。現在隆盛を極めているUNIXは開発環境の充実したＯＳであり、用意されているツールの豊富さは群を抜いている。UNIXが評価を受けている理由でもあろう。

（2）プロトタイピング技法

第1版ｃａｔは我々にとってコンパイラ開発の第一歩であった。そのために、数々の問題を露呈した。しかし、紆余曲折の末、クロスコンパイラとしての完成にこぎつけた。その結果、第2版開発のためのノウハウや貴重な教訓を残してくれた。もし、第1版の問題が明らかになった時点で第1版を放棄していたならば、我々が得ることのできた知見は半減していたであろう。第1版を稼動状態とすることができたために、それは第2版のプロトタイプとして有用なものとなったのである。ソフトウェアの方式設計においては、“まず動くもの”を作り、その中で問題点を明確にするプロトタイピング技法が注目を浴びている。我々はｃａｔの開発において、図らずもプロトタイピング技法を実線視、その有用性を確認した。

（3）ソフトウェアの保守性・拡張性

大学でのソフトウェア開発は、数年で担当者が入れ替わるために、保守性や拡張性を考慮した設計をすべきである。当然、実行効率とのトレードオフは考えなければならない問題であるが、保守性や拡張性を重視したシステム設計を行なう必要がある。第2版ｃａｔでは、この解決案としてフェーズ分割を行ない、プロセッサに独立な部分（プリプロセッサ、パーザ、オプティマイザⅠ）とプロセッサに依存する部分（コードジェネレータ、オプティマイザⅡ、アセンブラ）に分離した。プロセッサに独立な部分とプロセッサに依存する部分のインタフェースは中間コードを用い、中間コードにはできる限り自由度を持たせた設計とした［9］。

7．おわりに

第2版ｃａｔがセルフコンパイラとして稼働したことにより、ＯＳ／ｏプロジェクトの開発環境は飛躍的に向上した。開発環境の整備から一歩一歩進めている我々の歩みは、決して速いものではなかった。しかし、研究の基幹部分をおろそかにしたのでは、長期的な研究の可能性と自由度は大きく制限される。ｃａｔの開発とそれを用いたＯＳ／ｏ第2版の日本語化などによって、我々の研究が大きく加速されることを願っている。

謝辞

ｃａｔの開発は、多くの人々の努力と援助によって支えられてきた。ｃａｔ開発の基盤を築いた諸先輩方に心から感謝する。特に、ｃａｔ第1版の開発を担当した篠田佳博（現㈱日立製作所）、藤森英明（現日本電気㈱）の両氏には卒業後も何度となく御指導していただいた。

また、不慣れな磁気テープ装置の使用方法をはじめ、VAX/UNIX、M-280H等について細かい点まで御指導していただいた東京大学大型計算機センターの木村、松方、村尾の三氏をはじめとする教職員の方々に深甚なる謝意を示す。

参考文献

[1] B.W.カーニハン、D.M.リッチー著、石田晴久訳：“プログラミング言語Ｃ ── UNIX流プログラム書法と作法 ──”、共立出版

[2] 篠田佳博、藤森英明、中川正樹、高橋延匡：“ＯＳ／ｏのツール・セット（4） ── 言語Ｃコンパイラ ──”、情報処理学会第30回全国大会予稿、pp.365-366、1985.3

[3] 中川正樹、篠田佳博、藤森英明、高橋延匡：“MC68000ユニ＆マルチ・プロセッサ・システム用システム記述言語Ｃ処理系の開発”、情報処理学会計算機システムの制御と評価研究会資料21-7、1983.12

[4] UNIX Programmer's Manual, 7th edition, Virtual VAX-11 Version

[5] “オープン入出力機器利用の手引第1版”、東京大学計算機センター、1983.3

[6] 長谷部紀元、石田晴久、岡本匡人：“VAX/UNIXの端末からM-200Hを使うためのcvosコマンド”、東京大学計算機センターニュース、Vol.13、NO.7、pp.103-105、1981.7

[7] 並木美太郎、田中泰夫、中川正樹、高橋延匡：“言語Ｃコンパイラｃａｔの最適化とソフトウェアツールへの応用”、情報処理学会第32回全国大会予稿、1986.3

[8] Earley, J., Sturgis, H.,“A Formalism for Translator Interactions”, Comm. ACM, Vol.13, NO.10, pp.607-617, 1970.10

[9] 並木美太郎、施清池、森岳志、中川正樹、高橋延匡：“言語Ｃコンパイラｃａｔの方式設計と最適化”情報処理学会ソフトウェア工学研究会資料45-2、1986.6