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

中川正樹，篠田佳博，藤森英明，高橋延匡
東京農工大学工学部数理情報工学科

1．はじめに

計算機システムのシステム記述言語として高水準言語を用いる試みはMULTICS[1]に始まり，以来，大規模ソフトウェア・システムを構築するうえでの生産性，保守性の利点から，数多くのシステム開発で採用されてきている[2,3]。この動きは，システム記述言語，そして，それによって記述されるソフトウェア・システムの構成論での新しい展望と発展の契機を与えてきた。今日では，幾つかのシステム記述言語が確立し，それらによるシステム・ソフトウェア開発の方法論も浸透しつつある。

ところが，システム記述言語が計算機システムの性格を諸に反映する以上，システムのハードウェア環境のみならず，そのシステムの開発及び使用目的に合致したシステム記述言語及びその処理系が必要となる。特に，そのコード生成部，及びプログラム実行環境は，対象システムに合わせて独自設計しなければならない。なぜならば，処理系の容易な移植は，システムの開発目的の移植につながりかねないからである。このことは，研究目的の計算機システムでは研究の可能性を制限しないためにもなおさら重要であろう。

我々は，オンライン手書き日本語文字認識等のパタン認識や人工知能の問題を対象としたマルチ・マイクロプロセッサ・システム，日本語文書出力のためのインテリジェント・プリントサーバ，そして，これらのシステムを試作するtoolあるいはvehicleとしての単一プロセッサによるTSSシステムの研究開発を行ってきたが，それら3つのシステム総てにおいてMC68000をプロセッサに選び，それらに統一的で，かつ最も自由度の高いプログラム実行環境を設定した。そして，この一本化された実行環境に合わせたシステム記述言語処理系の開発を進めてきた。

本稿では，設計及びコーディングを完了した第一版について述べるが，紙面の都合上，我々のシステムに固有の問題，及び，その解決法に焦点を絞って，アーキテクチャ及びオペレーティング・システムとの関係を中心に述べる。それらは，我々の研究目的と上記3システムのアーキテクチャによって課せられた以下の要求を，言語処理系として如何にサポートするかということである。

(1) 実記憶の環境での，マルチ・ユーザ，マルチ・タスクの実現、特に，並列タスクの一形式として導入したタスク・フォースの実現
(2) 実記憶系におけるプログラムの共有，及び，ROMライブラリのサポート
(3) リエンタラブルでリロケータブルなコードの生成
(4) システム記述言語で書かれたプログラムは総てROMライブラリ化できること
(5) タスク間交信

2．システム記述言語の必要性とその開発目的

2.1 研究背景

本稿で述べるシステム記述言語処理系は，以下で述べる開発中の計算機システムのために設計されたものであり，そしてまた，それら計算機システムの開発目的は，我々の研究背景を抜きにしては語れない。そこで簡単に，それらのシステ

ムを試作開発するに至った背景と動機について述べる。

我々は，日本文入力としてのオンライン手書き文字認識の研究[6]を，小型計算機H-10Ⅱ/A及び大型計算機ACOS-600上で行ってきたが，この研究環境が研究の可能性をかなり制限してきたと言わざるを得ない。H-10Ⅱ/Aはメモリ空間が64kBytesと小さく，認識システムを試作しては評価・改良を繰り返していくパタン認識の研究には極めて不向きである。ACOS-600はH-10Ⅱ/Aよりは，はるかに大規模なシステムではあるが，i)常にオーバーロードであり，シミュレーションはできても，オンライン・リアルタイム処理には使えない，ii)システムが複雑すぎて全く手が出せない，という問題がある。要するに，メモリ空間が大きくて，かつ構成がよく分っているシステムが一番好ましい。

我々はまた，上記研究と並行して，パタン認識や人工知能処理のためのマルチ・マイクロプロセッサ・アーキテクチャの研究を他研究グループと共同で進めてきた[11]。そして，MC68000をプロセッサに選び，開発ツールとして，ACOS上に各種クロス・ソフトウェア，MC68000単一プロセッサ上にデバッガ及びアセンブラを作成した。しかし，これらだけではマルチ・マイクロプロセッサ・システムを開発することは不可能であり，本格的ツールとして，高水準言語をサポートする，同一プロセッサによるシングルプロセッサ・システムが不可欠である。

さらに当学科では，MC68000のための上記ツールを利用して，幾つかのインテリジェント・デバイスが開発されているが，そこでも，生産性，保守性，信頼性の面で高水準言語による開発が切望されている。

以上の背景から，研究のtool，vehicleそしてenvironmentとして，研究室のメンバが同時に使える小型TSS計算機システムを試作するに至った。本節の以下の部分では，シングル・プロセッサ・システム，マルチ・マイクロプロセッサ・システム，及びインテリジェント・デバイスの一例について，そのアーキテクチャを概観し，それらに共通のプログラム実行環境及び言語処理系を設計した方針を述べる。

2.2 対象システム

(1) シングル・プロセッサ・システム：本稿の姉妹編に詳しいので，ここでは省略。

(2) マルチ・マイクロプロセッサ・システム

本システムは，並列処理のスペクトラムの中において，計算機ネットワーク(疎結合)とアレイ・プロセッサ／パイプライン・プロセッサ(均質並列処理)の中間に位置し，両者の特徴を有するとともに，パタン認識や人工知能処理でその必要性が広く認識されている，異質プロセスによる並列処理の実現を目指している。その基本アーキテクチャは，複数のプロセッサを複数のメモリ・モジュールにマルチ・ポートとマトリックス・スイッチで結合する形式をとる(図1)。ここで，システム記述言語処理系に課せられる最大の問題は，プロセッサからメモリ・モジュールのアクセスに介在するRelocation Registerである。この機構により，それぞれのプロセッサは，アドレス空間内に，複数のメモリ・モジュールを重ならないように取り込むことが可能になるのだが，このことは，プログラムの実行中にリロケーションが起り得ることを意味している。

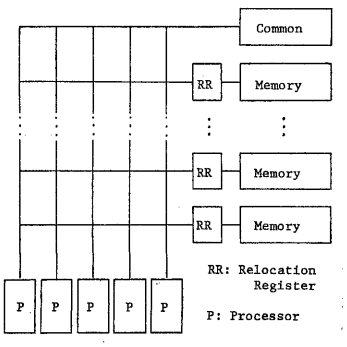

図1. マルチマイクロプロセッサ・システムの概念的アーキテクチャ

(3) インテリジェント・デバイス

当学科において，MC68000を内蔵したインテリジェント・デバイスが幾つか試作開発されているが，我々のグループにおいても，日本語文書出力のためのプリント・サーバを開発中である[2,3]（図2）。このシステムは，レーザ・ビーム・プリンタ，B4サイズをカバーするフレーム・メモリ，そしてMC68000で構成され，日本語文書イメージをホスト計算機とは独立に作成するフォーマッティング・マシンとして機能する。そのソフトウェアは，特権ROM領域のモニタ，デバッガ，そしてユーザ空間で現在開発中の日本語フォーマッタから成る。

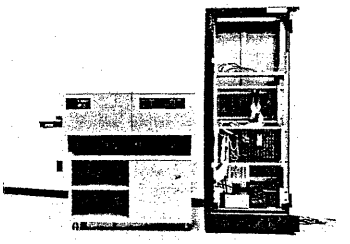

図2．日本語文書出力サーバの外観

2.3 処理系開発の基本方針

オブジェクト・コードの形式はプログラム実行環境によって相当変り得る。たとえば，実行中にリロケーションが起り得ないならば，計算の途中で絶対アドレスを扱っても何ら問題はない。そうすることによって，そのシステムにとっては，より高い効率が得られる。しかし，何種類かのアーキテクチャがある場合，それらに各々独自の実行環境を設計することは，研究目的のシステムにおいては決して効率がよいとは言えない。1つ1つのシステムについて処理系を開発し，かつメインテナンスしていかなければならない。むしろこの反対に，一番自由度の高い実行環境に統一し，あるシステムでは無駄が生じることがあっても，全体としての融通性，生産性，保守性を追求した方が，研究室レベルでの開発には得策であると思われる。我々の場合，上記マルチ・マイクロプロセッサ・アーキテクチャで要求される，プロシージャ，静的データ，動的データ総てのリロケータビリティは日本語文書出力サーバでは勿論のこと，シングル・プロセッサ・システムでも不要に終るかも知れない。しかし，そのことによってシステム・プログラミングの可能性に与えられる自由度は，実行速度を相殺して余りあると考える。それに，実行環境に新しい要求が生じるたびに処理系の基本部分を変更していくのは大変な労力を要する。むしろ、無駄を承知で適応範囲の広いオブジェクト・コードを生成した方が，処理系の寿命を延ばすうえでも賢明であろう。ハードウェアのコスト・ダウンは，こうしたソフトウェア作成における方法論上の贅沢をますます許容する方向に進展している。

我々はこのような認識に立って，上記システムに共通のシステム記述言語処理系を開発することとし，それらからの要求の和集合をとって第1章で挙げた要求項目を設定した。そしてさらに，システム・ソフトウェアでオンライン処理の実行速度を考慮して，中間コードのインタープリタとはしないこととした。したがって，オブジェクト・コードはMC68000の機械語とするが，コンパイラのデバッグのためにオブジェクト・コードの逆アセンブラを用意した。

3．言語仕様

言語仕様は互換性を考えて言語C参照マニュアル[4]にできる限り準拠したが，仕様の一部については，我々の目的やシステムのハードウェア・アーキテクチャに合わせて変更を加えた。以下にその項目を列挙する。

(1) データ型の追加　MC68000が提供するバイト単位の符号付き/無し整数演算を利用するために，それらバイト型（short と unsigned short）を追加した。また，PDP-11ではサポートされていない32ビットの符号無し整数（unsigned long）もかえた。

(2) 算術変換の変更　言語Cでは式中の文字(char)型, short型はあらかじめ整数(int)型に変換される。また, 式中の単長浮動小数点(float)型の値も前もって倍長浮動小数点(double)型に変換される。しかし, 我々はこの自動型変換は行なわずに, 演算時に被演算数の型の組合せに応じて行なわれる型変換だけとした。理由を以下に示す。

i) MC68000はバイト単位の演算が可能なため。

ii) 浮動小数点演算は, 割込み処理を使ってソフトウェアで行うため, かなりの処理時間を要する。特に, このことは倍長では著しく, 最悪の場合1m sec.を要することがある[2,10]。したがって, 単長で済む場合は, その方が有利であるため。

(3) 定数表記の追加　以上のことから, 整数定数としてshort型とint型, 浮動小数点定数としてfloat型とdouble型の区別が必要になる。そこで, short型整数定数を指定するために, 定数の後にsまたはSを書くか, 前にキャスト(short), (short int)または(int short)を置くこととし, 倍長浮動小数点定数についても, dまたはDを付けるか, キャスト(double)を前置することとした。

(4) gswitch文の導入　switch文の特殊な場合として, 分岐に要する時間がcaseラベルの数に依存しないgswitch文を導入した。通常のswitch文がcaseラベルを順にcc較していくコードに変換されるのに対して, gswitch文はジャンプ表による条件分岐コードに変換される。なお, 他システムへgswitch文を含んだプログラムを移植するには, プログラムの先頭で"#define gswitch switch"とすればよい。

(5) 構造体におけるフィールド・メンバの削除　1ワード内の数ビットに識別子を割り付けるフィールドは, その機能が言語Cのビットワイズの演算子で代用できるので、コンパイラを小さくする目的で削除した。

4. タスクと実行環境

4.1 タスクとタスク・フォース

並列タスクには, タスク間の関係の度合いによって, 2種類のタイプがあると考えられる。1つは, それぞれのタスクがほとんど独立な仕事を独立に行う形態であり, もう一方は, タスク間で情報の交換を頻繁に行ないながら, 全体で一つの問題を解決するプロジェクト・チームの形態をとるものである。前者はそれぞれ独立したロード・モジュールによるマルチ・タスクで, static linkの環境では, それらは共有変数を持たないので, タスク間の交信にはsupervisorが関与することになる。この形態としては, コマンド・インタプリタと, それによって起動されるコンパイラ等が挙げられる。一方, 後者は, 1つのロード・モジュールから動的に生成消滅されるタスク群で, 静的変数やアクセス・ファイルを共有し, かつ外部に対して抽象化することができる。また, タスク生成・消滅に際しては, static linkの特徴から動的変数領域の確保・開放だけで済むという利点がある。この形式を経営科学の用語を流用してタスク・フォースと呼ぶことにする。

4.2 タスクとしてのROMライブラリ

ライブラリとしては, (1)ソース・プログラム・ライブラリ, (2)リロケータブル・オブジェクト・モジュール・ライブラリ, そして(3)実行時形式ライブラリ(ROMライブラリを含む)が考えられる。このうち, (1)はプログラム中にin-line展開することによって, (2)はリンク・エディットすることによって利用することができる。static linkで問題なのは(3)の場合である。しかし, 今後のソフトウェアのROM化傾向を考えると, これが最も重要な方式であると捉えなければならない。

我々は実行時形式ライブラリについて，以下のような要件を課した。

(1) 言語Cで記述されたプログラムは，ピュアなものに限らず実行時形式ライブラリにできること。たとえば，コンパイラ，エディタ等もその対象である。

(2) supervisor領域にROM化またはロードして，ユーザ間で共有できること。

実行時形式ライブラリの場合，もはやユーザプログラムとリンクすることはできないので，手続き領域，データ領域それぞれライブラリルーチンとユーザルーチンで別個に持たなければならない。したがって，この形式のライブラリはユーザのタスクによって起動された別タスクと考えるのが自然である。

4.3 リエンタラビリティ，リロケータビリティと実行環境

以上の認識を踏まえて，実記憶系における，並列タスク，プログラム・データの共有，ROMライブラリ，任意プログラムのROMライブラリ化，及びタスク間交信をサポートする，リエンタラビリティとリロケータビリティの方式とプログラム実行環境を以下のように設計した。

まず，リエンタラビリティを実現するために，手続きとデータとを分離した。したがって，コンパイラはオブジェクト・コードとして手続きとデータを分けて生成する。データ領域としては，外部変数，外部静的変数，内部静的変数を1つの領域(静的領域)にまとめ，内部動的変数はスタック上に割り当てることにした。

さらに，リロケータビリティを得るために，たとえば分岐命令はPC相対，データの参照はアドレス・レジスタ相対とし，絶対アドレスは全く用いないようにした。相対アドレスを用いるために，幾つかのアドレス・レジスタを基底に使用している。

(1) 手続き領域の先頭を指すベース・レジスタ

関数呼び出しにおける帰り番地や飛び先番地はこの先頭からの変位を用いている。また，関数のポインタ値もこの先頭からの相対アドレスである。

(2) 静的領域の基底を指すベース・レジスタ

外部変数，外部静的変数，内部静的変数の参照はこの基底からの相対で行う。

(3) 現在アクティブな内部動的変数領域の基底を指すベース・レジスタ

現在アクティブな内部動的変数の参照は，このベース・レジスタ相対で行う。

また，静的領域の基底とは別にユーザ空間の基底を設け，データへのポインタはユーザ空間の基底からの相対とした。こうすることによって，静的領域を共有しないタスク間での，ポインタ変数による交信を可能にしている。

この他，スタック・ポインタと，ユーザ空間の基底の補数値を保持するためにもアドレス・レジスタをそれぞれに割り付けている。後者を設けたのは，変数のポインタ値(データ領域の基底＋変位－ユーザ空間の基底)を図3に示すように1命令で算出してしまうためである。このようにすることによって，実行中どこで中断・リロケーションが起きても再開が可能なコードとなった。

以上をまとめて，メモリ領域の構成を図4に，アドレス・レジスタの割当てを表1に示す。なお，データ・レジスタは総て作業用に開放している。ただし，A7は使用していない。アドレス・レジスタA7はハードウェアで提供するシステム・スタック・ポインタで，これを使う高機能命令も幾つか備えているが，i)これらの命令は絶対アドレスを使う，ii) MC68000のアドレス空間を最大限に利用する方法においては，シス

```
LEA  -d(Ax, A6.L), Working Register

-d: 変数のベース・レジスタからの変位
Ax: A2 or A3
A2: 内部動的変数用ベース・レジスタ
A3: 静的領域のベース・レジスタ
A6: ユーザ空間の基底の補数値を保持
```

図3. ポインタ値の算出法

テムスタック領域の位置及び大きさがハードウェアで制限される，という理由で使用していない。

5. 処理系の概要

処理系の内部仕様についてのより詳細な記述は別の機会に譲ることにして，本稿では簡単にその構成を記すに留める。

コンパイルの方式としては極めて一般的な方式を採っている。字句解析，構文解析で木構造を生成し，それから中間オブジェクトコードを生成する。しかし，まだこの時点では，分岐命令の変位や，外部変数の参照等で未定義な部分を含んでいる。パス2ではそれらを補い，最終的なリロケータブルオブジェクトモジュールを完成する。構文解析では再帰下向き解析法を基本とし，式についてだけ演算子優先順位法で解析している。現在，設計及びコーディング(言語Cで約9000行)を終え，東大大型センタのVAX/UNIXシステム上でデバッグ中である。

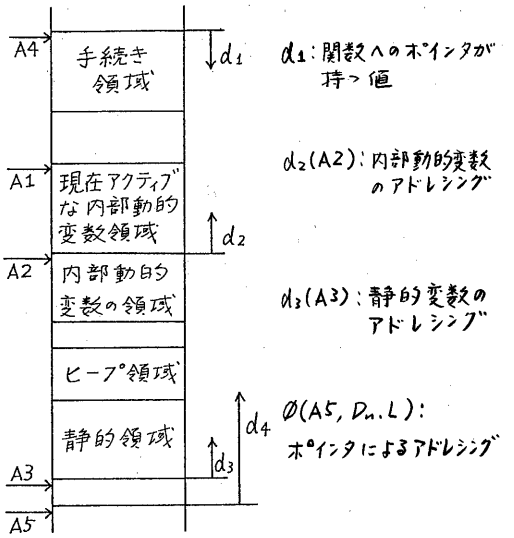

図4. メモリ領域の構成

表1. アドレスレジスタの割当て

アドレスレジスタ	用途
A0	作業用レジスタ
A1	スタック・ポインタ
A2	内部動的変数領域の基底
A3	静的領域の基底
A4	手続き領域の先頭
A5	ユーザ空間の基底
A6	A5の補数値
A7	使用せず

謝辞

本システムの開発のために東大大型計算機センタのVAX/UNIXシステムを使用するにあたり，有難い御指導御支援を賜っている同センタの職員の方々に深謝する。本研究及び本システムの開発には，これまでに試作開発した，ACOS-600上のMC68000シミュレータ，クロスアセンブラ，MC68000 CAIシステム，及びMC68000上のデバッガ，レジデントアセンブラ等が基礎になっている。ここにおいて，上記ツールの作成を卒業研究あるいは修士研究の一環として行った，武部桂史，林努，岩崎俊夫，志村隆則，高田直樹，本間幹男，石丸知之の諸氏，そしてその援助を賜った教職員萩原洋一氏に感謝の意を表する。

参考文献

[1] Corbató, F.J. and V.A. Vyssotsky: Introduction and Overview of the Multics System, Proc. FJCC, (1965)
[2] 藤森美明他: MC68000上で実現した浮動小数点演算方式のC比較 — IEEE方式と浜田方式のC比較 —, 本学会第29回全国大会(1982)
[3] Kernighan, B.W. and P.J. Plauger: Software Tools, Addison-Wesley, (1976)
[4] Kernighan, B.W. and D.M. Ritchie: The C Programming Language, Prentice-Hall, Englewood Cliffs, (1978)
[5] Motorola Inc.: MC68000 16-BIT MICROPROCESSOR User's Manual Second Ed., (1980)
[6] Nakagawa, M. et al: On-line Recognition of Handwritten Japanese Characters in JOLIS-1, Proc. of 6th Intl. Conf. on Pattern Recognition Munich, (1982)
[7] 中田育男: コンパイラ, 産業図書, (1981)
[8] Redell, D.D. et al: Pilot: An Operating System for a Personal Computer, C.ACM Vol. 23, No. 2, (1980)
[9] Ritchie, D.M. and K. Thompson: The UNIX Time-Sharing System, C.ACM Vol. 17, No. 7 (1974)
[10] 篠田佳博他: MC68000上での浮動小数点演算の実現 — IEEE方式と浜田方式の実現 —, 本学会第24回全国大会(1982)
[11] 高橋延匡他: マルチ・マイクロプロセッサシステムとその オペレーティングシステムの基本設計, 本学会第26回全国大会(1983)
[12] Takahashi, N. et al: 浄書: Japanese Output Server with HOspitality, Proc. of ICTP, Tokyo, (1983)
[13] 武部桂史他: MC68000アドレス空間の問題点と構成方式, 本学会マイクロコンピュータ研究会 17-1 (1981)