4V-11 OS／omicronにおける複数の浮動小数点方式のサポート

森岳志，中川正樹，中森真理雄，高橋延匡
(東京農工大学・工学部)

1. はじめに

アプリケーション・プログラムの記述において、浮動小数点演算機能は必要不可欠な道具として用いられているが、オーバフローやアンダフロー問題など使いにくい面も多く、数値解析の研究者達にとって不備も多い。

このような背景の下で、近年、オーバフローやアンダフロー問題の解消を目的とした松井・伊理方式[1],浜田方式[2]や、規格化を目的としたIEEE方式[3]などの新しい浮動小数点方式が提案された。我々は種々の浮動小数点方式を比較することができるように、ひとつのコンパイラでそれら複数の方式をサポートすることにした。さらに、コンパイルの手間を軽減するために、実行時にそれらの方式を選択できるようにした。本稿は、これらの複数の新浮動小数点方式をＯＳ／omicron用言語処理系の上で実現したことについての報告である。

2. ＯＳ／omicronとｃａｔの概要

我々は、自分達で手の入れられる“自前の”計算機システムの研究・開発を行なっている[4]。そのオペレーティング・システムをＯＳ／omicronと呼び、日本語情報処理や人工知能への応用を前提としたアーキテクチャを構成している。ＣＰＵには16Mバイトのアドレス空間を持つモトローラ社のMC68000を採用している。ＯＳ／omicronでは生産性や移植性を重視し、システム記述言語に言語Ｃを採用している。そのための言語処理系として、自前のＣコンパイラｃａｔ第２版が1986年春から開発システム上で稼働している[5]。

ＯＳ／omicronではリロケータブル、リエントラントなコードを標準としている。これはＯＳ／omicronの設計目標であるマルチタスクやプログラムのROM化を実現するためである[4]。リロケータブルなコードを実現するために、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一切用いないようにした。そのため、幾つかのアドレス・レジスタをベース・レジスタとして使用している。ｃａｔは、以上のプログラム実行環境を想定したコードを生成する。

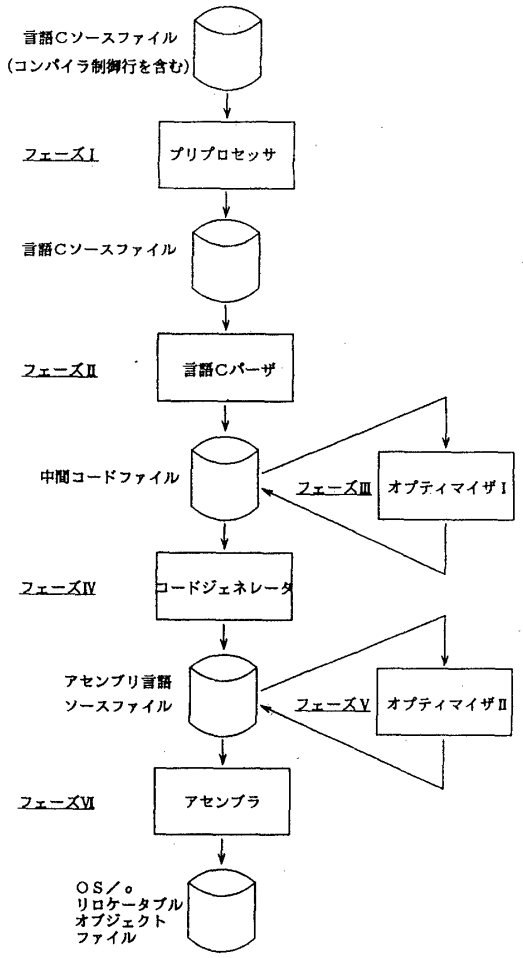

図１．ｃａｔの構成

3. ｃａｔ第2版とその拡張

ｃａｔ第2版の構成を図１に示す。ｃａｔはコンパイル機能の他、中間コードを用いた文書化ツールやデバッグツールに用いられる[6]。各フェーズの行数の合計は3万行に及んでいる。

ｃａｔ第2版ではシステムプログラムの記述に徹したために浮動小数点演算機能のサポートは行なっていなかった。しかし、ＯＳ／omicronでは、言語Ｃをアプリケーションプログラムの記述言語としても用いる予定であり、浮動小数点演算機能のサポートは必要不可欠な課題となった。

浮動小数点方式としては、伝統的な方式に対する不備を解消するために、近年、種々の方式が提案されているので、我々はこれらの新方式をｃａｔにとり入れることにした。しかし、これらの新浮動小数点方式は新しいため、未知の部分が多い。そこで、それらの諸方式を比較することができるように、次に示す４種類の浮動小数点方式をｃａｔでサポートすることにした。

(1)浜田方式[2]
(2)IEEE方式[3]
(3)松井・伊理方式[1]
(4)MIL-STD-1750A方式(図2)

現時点では、(1)-(4)をソフトウェアによってサポートしている(将来はハードウェアでサポートしたいと考えている)。

Implementation of Multiple Floating Point Representations on OS/omicron

Takeshi MORI, Masaki NAKAGAWA, Mario NAKAMORI and Nobumasa TAKAHASHI
Tokyou University of Agriculture and Technology

方式(1)-(4)の中では、浜田方式を標準とすることにした。その理由は、事実上、オーバフローやアンダフローが発生しない,データ長に独立である,などの数多くの利点を、浜田方式が備えているからである(浜田方式用浮動小数点演算ライブラリの研究・開発は、開発用ＯＳ(CP/M-68K)上で、ｃａｔの開発と並行に行なわれた[7])。

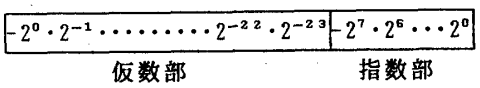

図２．MIL-STD-1750Aフォーマット(32ビット数の場合)

浮動小数点演算機能の導入による言語処理系の変更は、すべてコードジェネレータレベルで行なう方針を取った。一般に、パーザの保守はそのプログラムの性質上、他のプログラムに比べて困難であるため、どのような浮動小数点方式を実現するかにかかわらず、パーザには変更を及ぼさないようにした。

4. 複数の浮動小数点方式の実現法

一般に、浮動小数点演算機能をとり入れる方法には、関数呼出しによる方法とエミュレーションによる方法の二つの方法が考えられる。ｃａｔ第2版を拡張し、上述の二通りの方法を用いて複数の浮動小数点方式のサポートを試みた。

方法1 関数呼出しによる方法

コードジェネレータは、浮動小数点演算に対し通常の関数呼び出しのコードを生成する。浮動小数点演算を行なう関数はライブラリとして用意される。そして、リンケージエディタにより、ユーザ・プログラムとライブラリとを静的にリンクすることで浮動小数点演算機能を実現する。演算の種類(四則演算の区別など)の選択や適用される浮動小数点方式の選択は、コードジェネレータの生成する関数名によって行なわれる。

方法2 エミュレーションによる方法

コードジェネレータは、浮動小数点演算に対しMC68000に備えられたエミュレーション命令を生成する。本エミュレーション命令は内部例外を発生する。演算の種類の選択や適用される浮動小数点方式の選択は、コードジェネレータの生成するエミュレーション･ハンドラへの引数によって行なう。

方法1または方法2のいずれでも、適用される浮動小数点方式の選択は、コンパイル時の(正確にはコードジェネレータの)オプション機能によって行なわれる。両者の方法に対する実行速度の比較は、関数呼出しの手間と例外処理の手間との比較になる。第２章で述べたように、ＯＳ／omicronでは絶対アドレスを一切用いないために、関数呼出し用に備えられているjsrやrtsなどの命令は用いることができず、多少、関数呼出しの手間が大きくなっている。簡単なテストを行なった結果、方法2が速度の点で有利となることが確かめられている。

ユーザがどの浮動小数点方式を使うかを決定するのは、伝統的な方法ではコンパイル時と考えられてきた。したがって、もし、いくつかの浮動小数点方式の比較を行ないたいのであれば、同一のプログラムに対し、浮動小数点方式の数だけコンパイルの作業が必要になる。この手間を軽減するために、方法2の拡張として、適用される浮動小数点方式を実行時に選択する方法を考え、実現した。より具体的には、

(ア)上述の方法2と同様に、コードジェネレータはエミュレーション命令を生成する(ただし方式の選択は除く)。
(イ)各タスクごとに割当った浮動小数点方式選択用のフラグによって、適用される浮動小数点方式の選択を行なう。
(ウ)プログラム中に表われる定数はプログラムのロード時または実行時に、適用される浮動小数点方式の内部表現に変換する。

この方法によれば、比較する浮動小数点方式の数によらず、一回のコンパイル作業によって、さらには一つの実行形式によって複数の浮動小数点方式が適用可能となる。また、プログラムのロード時にただ一度だけ定数を変換する方法を用いた場合、実行速度を落とさず、一つの実行形式で複数の浮動小数点方式の適用が可能となる。

5. おわりに

新浮動小数点方式に導入されている非数の概念をどのように扱うか、さらには新浮動小数点方式の詳細な評価が今後の課題である。

参考文献

[1] 松井,伊理: “あふれのない浮動小数点表示方式”, 本学会論文誌, Vol.21, No.4(1980),pp.306-313.
[2] 浜田: “二重指数分割に基づくデータ長独立実数値表現法II”, 本学会論文誌, Vol.24, No.2(1983),pp.149-156.
[3] IEEE Computer Society Microprocessor Standards Committee Task P754: “A Proposed Standard for Binary Floating-Point Arithmetic”, Draft 8.0 of IEEE, Computer, Vol.14, No.3(1981), pp.51-62.
[4] 高橋: “ＯＳ／omicronの設計思想”, 本学会第29回全国大会(1984.9), pp.339-340.
[5] 屋代,他: “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-”, 本学会ソフトウェア工学研究会資料48-1(1986.6).
[6] 並木,他: “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”, 本学会第32回全国大会(1986.3), pp.475-476.
[7] 森,他: “ＯＳ／oにおける浮動小数点方式に関する研究”, 科学研究費補助金試験研究(1) 非正規数表現法の数値計算に及ぼす影響についての研究” 報告書(1986.3), pp.78-99.