言語Cコンパイラのマルチバイト化の実現方式†

並木美太郎†† 早川栄一†† 下村秀樹††
田中泰夫††* 中川正樹†† 高橋延匡††

日本においてはコンパイラの日本語化を，1バイト2バイト混在コード体系，各国語を表す文字型の導入などによって行ってきた．しかし，ソフトウェアシステムの各階層で文字コードの一貫性がないため，システムに種々の制約を課することになっている．そこで，文字型をマルチバイトとするコンパイラを実現し，このコンパイラによってOSや応用プログラムを開発することで，日本語プログラミング環境を構築することとした．しかし，システム開発初期に存在するのは1バイトの環境であり，コンパイラのフル2バイト化が問題となった．筆者らはコンパイラをクロスコンパイラの手法でフル2バイト化する方式を提案した．この方式では，文字型や文字リテラルの変更を自己記述されたコンパイラのソースプログラムに対して行う．しかし，実際のコンパイラはマルチフェーズ構成となっており，どのフェーズにどのような変更を行うかが実現上の問題となる．本論文では，マルチバイトコードをソースプログラムとする言語Cコンパイラの実現方式について述べる．パーザのみの変更によるフル2バイトのクロスコンパイラの実現方式を体系的に整理した．さらに，その方式を改良し，コンパイラのソースプログラムの変更を減らしながらマルチバイトコード化を行う方式を提案する．この方式を用いて，ISO 10646 フル4バイトの言語Cコンパイラを実現し，方式の有効性を確認した．

1. はじめに

英語をはじめとする欧米圏の言語は，ISO 646 に代表されるように文字コードを1バイトで表現する．対して，1978年に標準化された JIS C 6226 (現 JIS X 0208) は，2バイトで漢字を表現する．近年では，多国語処理の必要性から，漢字圏に共通の2バイトコードである Unicode[1]，さらには全世界の文字コードを4バイトで表現する ISO 10646[2),3)]などが話題となっている．

従来の計算機は1バイトコードを基本としてプログラミングシステムを構成しているため，マルチバイトコードへの対応が問題となる．互換性の問題から，日本においては1バイト2バイト混在コード体系によって計算機の日本語化を行った[4)]．しかし，応用プログラム，プログラミング言語，OS の間で文字コードの一貫性がないため，プログラミングの手間の増加，システムに種々の制約を課することになっている．

筆者らは，日本語情報処理を行うための基盤として，フル2バイトコードを内部コードとする OS/omicron を開発した[5),6)]．OS/omicron のシステム記述言語として言語Cを使用している．この言語Cコンパイラ CAT (C Compiler developed at Tokyo University of Agriculture and Technology) の文字型を2バイトとし，このコンパイラで OS をコンパイルすることにより，フル2バイトの OS を実現した．さらに，このフル2バイトコンパイラを日本語プログラミング環境にも用いている[7)]．しかし，OS/omicron および言語Cコンパイラ CAT の開発環境は，UNIX・CP/M-68K などの1バイトのプログラミング環境であったため，CAT も当初は1バイトコードを文字集合として開発された．OS/omicron 開発の初期の段階で，1バイトコードの環境下におけるコンパイラのフル2バイト化が問題となった．

本論文では，マルチバイトコードを文字型とする言語Cコンパイラの実現方式について述べる．マルチバイト化を考慮した処理系に対して，パーザのみの変更によるフル2バイトのクロスコンパイラの実現方式を提案する．さらに，その方式を改良し，コンパイラのソースプログラムの変更を減らしマルチバイトコード化を行う方式を提案する．この方式を用いて，ISO 10646 フル4バイトの言語Cコンパイラを実現し，方式の有効性を確認した．

2. マルチバイト化の基本方針

言語処理系において文字コードの関与する要素として，プログラムテキスト中のコメント・文字（列）定数・文字型・識別子，ファイルや端末入出力などの外部コードがある．コンパイラをマルチバイト対応にす

† Methods of Implementing a Multi-byte C Compiler by MITAROU NAMIKI, EIICHI HAYAKAWA, HIDEKI SHIMOMURA, YASUO TANAKA, MASAKI NAKAGAWA and NOBUMASA TAKAHASHI (Department of Computer Science, Faculty of Technology, Tokyo University of Agriculture and Technology).
†† 東京農工大学工学部電子情報工学科
* 現在 (株)富士 Xerox
Fuji Xerox, Ltd.

る一つの方法として，従来のソフトウェアの互換性を重視し，従来の文字型に加えて各国語のための文字型を追加する方法がある[8]．実際，この方法でいくつかのプログラミング言語が日本語化された[9)~11)]．しかし，この方法では OS を含むシステム全体の文字型の一貫性を確保するのが困難である．そこで，本研究でのマルチバイト化とは，システムで使用するすべての文字コードを同一の符号化文字集合とする．これによって，識別子や外部コードに至るまで文字コードの一貫性を確保できる．さらに，正規化日本語文字列[12)]のアプローチだと半角/全角など出力上の属性をコードに含ませることになるため，同一表現の文字を同一の文字コードとした．

システムで使用するすべての文字コードを同一の符号化文字集合とするために，言語処理系の文字型をマルチバイトコードとする．マルチバイトの文字型を有するコンパイラでコンパイルすることにより，プログラムは自動的にマルチバイトの文字を処理できる．同様に OS もマルチバイトのコンパイラでコンパイルすることにより，マルチバイト対応になる．本研究のマルチバイトコンパイラの開発動機は，マルチバイトコードを内部コードとする OS 上で，マルチバイトコードを文字型とするコンパイラを提供することにより，マルチバイトコードのプログラミング環境を実現することである．

マルチバイトコードを文字型とするコンパイラの開発方式としては，ブートストラップ用として小規模のマルチバイトのコンパイラを実現する方法がある．しかし，従来の1バイト環境におけるデバッガなどのプログラミング環境を有効利用できないなどの問題がある．システム記述言語のコンパイラの多くは自己記述される．そこで，言語処理系のマルチバイト化に関して，次の方法を利用することとした．

（1） まず1バイトのプログラミング環境において，1バイトのコンパイラを自己記述する
（2） このコンパイラを変更しマルチバイトのクロスコンパイラを作成する
（3） クロスコンパイラを用いてマルチバイトのセルフコンパイラをコンパイルする

上記方法を用いたマルチバイト化方式を提案し，フル2バイトコンパイラを実現した[14)~16)]．概略を図1に示す．クロスコンパイラによるコンパイラ開発[13)]の手法はコンパイラ開発に広く利用されている．本論文では，クロスコンパイラによるコンパイラ開発を，コンパイラの扱う文字集合に適用する手法を考察する．

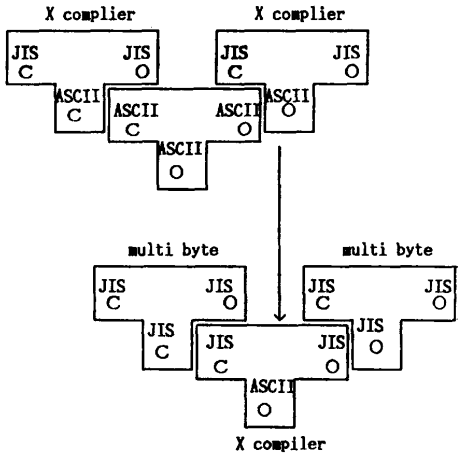


図1 クロスコンパイラによるマルチバイト化の概略
Fig. 1 An outline of implementing a multi-byte code compiler with cross compilers.

本方式では，次の手順でマルチバイトコードのコンパイラを開発する．

（1） クロスコンパイラ作成のために1バイトのコンパイラに次の変更を行い，1バイトの環境下でコンパイルする
・文字型の仕様を1バイトからマルチバイトに変更する
・ソースプログラム中の文字(列)定数を1バイトからマルチバイトに変更する

特に，トークン解析・予約語の文字コードが重要である．

（2） セルフコンパイラ作成のために1バイトのコンパイラに次の変更を行う
・文字型の仕様を1バイトからマルチバイトに変更する
・ソースプログラム中の文字(列)定数を1バイトからマルチバイトに変更する
（3） ソースプログラムのファイルの文字コード変換を行いながら，クロスコンパイラによりセルフコンパイラをコンパイルする

同様の手法により1バイトコードのコンパイラをフル2バイト化した例として，日本語 CLU がある[17)]．日本語 CLU では，MIT で作成されたパーザとライブラリを日本語化し，コード生成とアセンブラを新規に作成している．CLU の日本語化では，クロスコンパイラの入力を1バイトコード，出力を2バイトコードとしている．やはり，いずれかのフェーズでコード変換が必要だが，詳細は記述されていない．CLU の

ライブラリ中には，アセンブリ言語で記述されたランタイムルーチンが多く，フル2バイトのリテラルを処理する前処理系を作成したとの報告がある[18]．変更すべきアセンブリ言語のプログラムが多いときは，クロスコンパイラのアセンブラの前で（または中で）コード変換を行うのが有利であろう．

また，Pコードシステムによる Pascal の日本語化として，コンパイラとPコードのインタプリタを2段階に分けて日本語化したとの報告がある[19]．この実現では，コンパイラの文字定数とPコードインタプリタ中の文字列データだけを2バイト化し，次にコンパイラとインタプリタ中のすべての入出力を2バイト化している．

従来のソフトウェアは1バイトの環境下で作成されたため，マルチバイト化を考慮していないことが多い．このため，変更に多大な手間を要することがある．また，変更により保守，拡張性を損ねることのないようにしたい．

このようなことから，本研究では次の実現方針を定めた．

（1） ソフトウェア工学的観点を考慮すること，特にソースコードの変更を少なくすること

マルチバイト化にともなう変更が多いとき，変更による信頼性の低下，文書の変更，テスト項目などが増加する．そこで，ソースコードの少ない変更でマルチバイト化することを目的とする．

また，筆者らは言語処理系と並行してマルチバイトを内部コードとし並列処理の機能を持った OS である OS/omicron[14]を開発していた．OS/omicron の開発も1バイトの環境を利用したが，先に述べたようにマルチバイトのコンパイラを利用する．単に OS 開発だけならば，OS のソースコードに対して，文字型の定義を変更しツールで文字列を変換すればよい．しかし，OS/omicron の実行環境は並列処理用の機械語を有するように設計され，CAT もその機械語を生成するように設計されている．OS/omicron の開発に CAT の実現が必要なことから，少ない手間で1バイトのコンパイラのマルチバイト化を行う．

（2） マルチバイトコードを文字型とするコンパイラの設計指針を得ること

従来の1バイトコードの言語処理系をマルチバイト化するだけでなく，コンパイラ設計の指針を得ることを目標とする．

（3） セルフコンパイラとしてマルチバイトコードを外部コードとする実行環境で稼動すること

マルチバイトコードを文字型とするコンパイラは，マルチバイトコードの OS 上で稼働することを目的とする．従来の1バイトコードの環境下で利用するならば，例えばプリプロセッサにより文字型や文字定数を変換し，1バイトのコンパイラの入力とする方式によってもマルチバイト化できる．しかし，プリプロセッサと1バイトのコンパイラを組み合わせる方式では，マルチバイトの環境下で利用できないという問題がある．

（4） 独自開発とすること

短期にマルチバイトのコンパイラを入手するには，従来からある1バイトのコンパイラをマルチバイト化する方法が考えられる．しかし，従来のコンパイラの多くは1バイトコードを念頭に設計されている．多くの計算機システムにおいて応用プログラムの日本語化が完全に行われていないことなどからも，変更に多大な手間を要すると考えた．また，設計指針を得るため，マルチバイトコードの OS 上で稼働することなどからコンパイラを独自開発する．

本論文では，第3章1節においてマルチバイト化を行うコンパイラを設計する際の注意点について述べる．この注意点は，従来の1バイト環境下で設計されたコンパイラをマルチバイト化する際の注意点としても利用できる．

次に二つのマルチバイト化の方式について述べる．一つは変更は多くなるが，パーザの変更だけでマルチバイトのコンパイラを得られる方式である．もう一つは，変更が少なくなるよう改良した方式である．以後の章でこれら二つの方式について述べる．

3. OS/omicron 用フル2バイトコンパイラ CAT の実現

3.1 設計上の注意点

マルチバイト化を行う際，まずクロスコンパイラで文字型の変更を行う必要がある．言語Cコンパイラ CAT では，コンパイラ中のソースプログラムの文字型を次のようにマクロ定義した．言語Cによるシステムプログラミングにおいては，バイトの大きさを持つ値の型定義を頻繁に利用するが，バイト型の変数定義と文字型の定義を混同しないためである．

```
#define BYTE char   (1バイト環境下では
#define CHAR char   ともに 8 bit)
```

クロスコンパイラおよびセルコンパイラ作成時に

は，マクロ定義を次のように変更し，文字(列)定数をフル2バイトコードに変換する．

```
#define BYTE short (CAT の short 型は 8 bit
#define CHAR int          int       16 bit)
```

コンパイラの開発を ASCII コードのシステム上で行ったが，マルチバイト化を考慮してコーディング時に次の点を留意した．

(1) 文字型と，文字型の大きさを示す型を分離した

前述の CHAR・BYTE である．コンパイラ中で，char 型の変数はすべてこのマクロ定義を使って宣言するようにし，定義を一つのファイルにまとめて変更が容易になるようにした．

(2) 型の大きさの定義を表にする

パーザでは，型の仕様を定義するが，これを表にした．

(3) 可能な限り文字定数・文字列定数をプログラム中に埋め込まない

プログラムの各所に文字(列)定数が散乱し，変更が困難になることを防ぐためである．ただし，これらはプログラムの可読性を低下させることになった．後述する方式では，これらを自動化する方法を採っている．

(a) トークン切出し

if ('a'<=c && c<='z') などのトークン解析に関する文字定数をマクロ定義にした．

(b) メッセージ

すべてファイルから読むようにした．

(4) コード生成において，文字(列)定数を16進数で出力する

フル2バイト化の際，アセンブラの変更を不要とするためである．

(5) 各フェーズ間のインタフェースで名前のレコードを固定長としない

(6) 入出力はバイトストリームと文字ストリームを分離する

getc と get_byte など．

(7) 文字種別を表(配列)で行わない

識別子を構成する文字を

```
char c;
if (ident_table[c])
```

としない．これは，マルチバイト化時に表の増大を招くためである．

3.2 実　　現

本節では，フル2バイト化の手順の詳細を述べる．文献6)，16)では，Tバーによってフル2バイト化の手順を記述した．しかし，コンパイラ中に出現する文字コード種別をTバー上に記述すると煩雑となる．そこで，表記を定義し，コンパイラの関与する文字型を明確にする．

表記の定義：

・コンパイラ中に表れる文字コード

Cfile：ソースファイルの文字コード

Cin, Cout：入出力データの文字コード

何に対する文字コードかはフェーズの機能に依存する．例えば，パーザにおいては，入力はソースファイル，出力は中間コード中の識別子の文字コードである

Cset：識別子に許される文字集合

Ctype：文字型の符号化文字集合

Clit：文字(列)定数の文字コード

Cobj：オブジェクトモジュール中の識別子

f：フェーズの機能．パーザ，コード生成，アセンブラなど

Cenv：プログラムの実行される環境の文字コード．端末入出力，ファイル名など

・各ファイルを次のように記述する

ソースファイル：

s[f；〈Cin, Cout, Cset, Ctype, Clit〉；Cfile]

パーザ，コード生成，アセンブラという機能 f の各文字コードが〈Cin, Cout, Cset, Ctype, Clit〉であり，ソースファイルの文字コードが Cfile

オブジェクトコード：

o[f；〈Cin, Cout, Cset, Ctype, Clit〉；Cobj]

上記のソースファイルをコンパイルしたオブジェクトファイル．オブジェクトファイル中の識別子の文字コードは Cobj

プログラム：

f(s；〈Cin, Cout, Cset, Ctype, Clit〉；Cenv)

パーザ，コード生成，アセンブラまたはコンパイラソースファイルを s とし，プログラムの動く環境の文字コードは Cenv となっている．返す値は，コンパイラならばオブジェクトファイルとなる．

〈…〉の中がすべて同じ文字コード系なら一つの文字で表す．

例えば，ASCII コードを内部コードとする計算機における言語Cコンパイラの自己生成は，次のように表現される．

C'(p；A；A)←C(s[C'；A；A]；A；A)

また，提示した方式の概略は，次のように表現される

(1) s[C; A; A] に変更を加えて s[C; J; A] とする

(2) MetaC(p; J; A)←C(s[C; J; A]; A; A)

(3) s[C; J; A] をコード変換して，s[C; J; J] とする

(4) C(p; J; J)←MetaC(s[C; J; J]; J; A)

なお，CATのフェーズ構成は次のように表現される．

```
C(p; X; Y)=
    parser(p; X; Y)→codegen(p; X; Y)
                   →asm(p; X; Y)
        "→" は結果を受けわたすことを示す
o[p; X; Y]←C(p; X; Y)
p(f; X; Y)←LinkEdit(o[p; X; Y]
  +o[Lib; X; Y]; Y; Y)
```

o[lib; X; Y] はコンパイラの各フェーズにリンクされるライブラリルーチンである．なお，OS/omicronは静的リンクなので，リンカ LinkEdit によって実行形式モジュールを作成する．

次に，実現の手順を示す．

(1) MetaCAT 1 の生成

1バイトのコンパイラのパーザを変更することによって，クロスコンパイラを作成する．

(a) ライブラリに対してマクロ定義CHAR・BYTEを変更する

クロスコンパイラの実行される環境は1バイトであることから，端末入出力・ファイル名に対しては，コード変換を行う．この変更によって，s[lib; A; A] は s[lib; ⟨J, A, A, J, J⟩; A] となる．このソースプログラムをコンパイルする．

```
o[lib; ⟨J, A, A, J, J⟩; A]←
  C(s[lib; ⟨J, A, A, J, J⟩; A]; A; A)
```

(b) パーザに次の変更を行う

- マクロ定義 CHAR，BYTE の変更
- char 型の仕様を1バイトから2バイトに変更
- 予約語表，トークン解析中の文字コードを数値(JIS コード)に変更
- 出力の識別子を JIS から ASCII に変換する処理を付加

この変更の結果，パーザのソースプログラムは，s[parser; ⟨J, A, A, J, J⟩; A] となる．これをコンパイルする．

```
o[parser; ⟨J, A, A, J, J⟩; A]←
    C(s[parser; ⟨J, A, A, J, J⟩; A]; A; A)
parser(p; ⟨J, A, A, J, J⟩; A)←
    LinkEdit(o[parser; ⟨J, A, A, J, J⟩; A]+
        o[lib; ⟨J, A, A, J, J⟩; A]; A; A)
```

このクロスパーザを，従来のパーザと交換する．

```
MetaCAT 1(p; ⟨J, A, A, J, J⟩; A]=
    parser(p; ⟨J,A,A,J,J⟩; A)
    →codegen(p; A; A)→asm(p; A; A)
```

この MetaCAT 1は，入力であるソースプログラムの文字コードがフル2バイト，出力の中間コードおよびオブジェクトモジュール中の識別子は1バイトとなっている．すなわち，フル2バイトのクロスコンパイラであり，このクロスコンパイラによって従来のプログラムをコンパイルすれば，char 型はすべてフル2バイトとなる．制限は，識別子の文字集合が小さくなっていることである．つまり，漢字識別子を持ったソースプログラムをコンパイルできないが，従来のプログラムをコンパイルできる．

(2) 2バイトコンパイラの生成

(a) 1バイト版のソースプログラムに次の変更を行う

- マクロ定義 CHAR，BYTE の変更
- トークンに漢字コードを追加

ただし，これは16進定数で記述する．実際にリテラルで書けるのはフル2バイトになってからである．

- CHAR 型の仕様を1バイトから2バイトに変更

この結果，s[f; J; A] というソースプログラムになる．

(b) コード変換を行いながら，MetaCAT 1 によってコンパイルする

CodeConv(A): ASCII コードを JIS フル2バイトへ変換するプログラム

```
s[f; J; J]←CodeConv(s[f; J; A])
o[f; J; A]←MetaCAT 1(s[f; J; J];
                    ⟨J, A, A, J, J⟩; A)
f(p; J; A)←LinkEdit(o[f; J; A]
                  +o[lib; J; A]; A; A)
```

f: パーザ，コード生成，アセンブラ

s[lib; J; A] を最初にコンパイルし，o[lib; J; A] を作成する．

```
MetaCAT2(p; J; A)=parser(p; J; A)→
    codegen(p; J; A)→asm(p; J; A)
```

同様の手順でLinkEdit(p; J; A)も作っておく．これが文字集合も JIS となったクロスコンパイラであ

る．ただし，コンパイラ自身のオブジェクトコード中の外部名は1バイトコードとなってしまっているため，もう一度すべてのフェーズをコンパイルする．

（3） ターゲットのフル2バイトコンパイラの生成

2-b）のソースプログラム s[f；J；J] を用いる．

```
o[f；J；J]←MetaCAT2(s[f；J；J]；J；A)
f(p；J；J)←LinkEdit(o[f；J；J]
                    +o[lib；J；J]；J；A)
        これも最初にライブラリをコンパイルしておく
CAT2(p；J；J)=parser(p；J；J)→
                codegen(p；J；J)→asm(p；J；J)
```

以上の手順によって，フル2バイトコードを入力とするコンパイラを実現した．

3.3 フル2バイト化のまとめ

CAT の総行数は全体で約 40K 行，上記変更を CP/M-68K で行った[14),15)]．変更内容を**表1**に示す．変更に費やした労力は，4人で3週間であった．

表1 フル2バイト化における変更点
Table 1 The modifications of CAT for updating to full 2 byte code.

	クロス	ターゲット
型仕様の変更	4	→
パーザ		
識別子切出しに漢字を追加	1	→
型に対する定義表の変更	10	→
中間コード中の識別子を2バイトで出力		2
中間コード出力時に1バイトコードへ変換	1	削除
予約語の文字列を2バイトコードへ変換	24	もとへ戻した
コードジェネレータ		
中間コードの入力関数を変更	1	→
文字列定義の疑似命令出力を変更	1	→
アセンブラ		
識別子切出しに漢字を追加	1	→
リンカ	なし	なし
ライブラリ		
入出力のコードを1バイトへ変換	2	なし
バグ		
文字列のバイト長を文字数としていた	1	→
文字列領域を文字数で割り当てようとした	1	→
ファイル読込みのストリーム関数名をバイトストリームとしていた	1	→
ハッシュ関数の計算を符号付きにした	1	→

※ "→" は同じ変更であることを示す

現在，ターゲットコンパイラは OS/omicron 上で稼動し，手書き文字認識・文書推敲支援システム・卓上電子出版のソフトウェア開発に使用されているほか，OS/omicron 自身の開発にも使用されている．

4. マルチバイト化の方式改良と ISO 10646 フル4バイトコンパイラの実現

4.1 マルチバイト化方式の改良

第3章で示した方式の特長は，パーザを変更するだけで（制限付きの）フル2バイトコンパイラを実現できることである．しかし，ターゲットコンパイラを得るまで二つのクロスコンパイラを作成する必要がある．また，パーザの変更においては，中間コード中の識別子を2バイトから1バイトに変換するルーチンを追加しているが，このルーチンはターゲットコンパイラでは不要になる．

さらに変更作業において，メッセージの文字列はファイルとしたが，パーザ中に埋め込まれる予約語表の文字列，

```
{"if", TOKEN_IF}, /*表の初期化式*/
  ↓
static CHAR if_string [ ]
                    = {0x2369, 0x2366, 0} ;
                    /*if の JIS コード*/
  …
{if_string, TOKEN_IF},
```

の変更を，人手で16進数に変換した．

前章の方式では，ソースコードに対する無駄な変更があること，文字(列)定数の変更を人手で行ったことによって手間が増加した．そこで，クロスコンパイラの生成を一回で済ませ，文字(列)定数の変換をツールで行う方法を考案した．改良した方式を次に示す．

表記で，Nは新しい文字コード，0は古い文字コードを表す．

（1） クロスコンパイラ MetaCAT の生成

（a） すべてのフェーズのプログラムに次の変更を加える

- マクロ CHAR，BYTE の変更
- CHAR 型の仕様を変更
- トークンに新しいコードを追加

ただし，これは16進定数で埋め込む．実際にリテラルで表現できるのはマルチバイトになってからである．

これによって，クロスコンパイラのソースプログラ

ム s[f; ⟨N, N, N, N, 0⟩; 0] となる.

(b) 文字(列)定数を変換するツールを作成する

ソースプログラム中の文字(列)定数を16進定数に変換するツールを作成する. 例えば, フル2バイトコンパイラ作成のときは,

```
if('a'<=c && c<='z')
  p="ident";
      ↓
if(0x2361<=c && c<=0x237a)
  p="¥x23¥x69¥x23¥x64¥x23¥
     x65¥x23¥x6e¥x23¥x74¥x00";
```

という変換を行う. ただし, 入出力ともに文字コードは, 古い文字コードである. このツールを LitConv(s) と表す. このツールの実行によって,

```
s[f; ⟨N, N, N, N, N⟩; 0]←
        LitConv(s[f; ⟨N, N, N, N, 0⟩; 0])
```

となる. この左辺は, s[f; N; 0] である.

(c) 文字(列)定数を変換しながらコンパイルする

```
o[f; N; 0]←
   C(LitConv(s[f; ⟨N, N, N, N, 0⟩; 0]); 0; 0)
f(p; N; 0)←
   LinkEdit(o[f; N; 0]+o[lib; N; 0]; 0; 0)
```

この結果, 古い文字コード環境上で実行されるクロスコンパイラ

```
MetaCAT(p; N; 0)=parser(p; N; 0)→
                 codegen(p; N; 0)→
                 asm(p; N; 0)
```

を得る.

(2) ターゲットコンパイラへの移行

古い文字コードから新しい文字コードへのコード変換のツール CodeConv を作成し, ターゲットコンパイラをコンパイルする.

```
s[p; N; N]←CodeConv(s[p; N; 0])
f(p; N; N)←MetaCAT(s[p; N; N]; N; 0)
C(p; N; N)=parser(p; N; N)→
           codegen(p; N; N)→
           asm(p; N; N)
```

4.2 ISO 10646 コンパイラによる方式の評価

前節で述べた方式を, JIS X 0208 フル2バイトコード系から ISO 10646 フル4バイトコード系へ適用し, 有効性を確認した. 言語Cコンパイラは, 第3章で作成したフル2バイトの言語Cコンパイラ CAT を用いた. ソースファイルの規模は, 最新版の CAT (ANSI 準拠) で 70K 行となっている.

表 2 新方式における変更点
Table 2 The modifications of CAT using the updated method.

	クロス・ターゲット
型仕様の変更	4
パーザ	
型に対する定義表の変更	10
中間コード中の識別子を2バイトで出力	2
コードジェネレータ	
中間コードの入力関数を変更	1
文字列定義の疑似命令出力を変更	1
アセンブラ	なし
リンカ	なし
ライブラリ	なし

符号化文字集合は, ISO 10646 基本多言語面の基本日本語面 (群 32, 面 64) とした[2]. 従来の JIS X0208 は高位帯 I-11 へ写像される. 本実現では, 英数字など ASCII 文字集合に含まれる文字もここへ写像し, 同一表現の文字は同一コードとなるようにした. 制御符号は, NUL(0) を3バイト前置し4バイトの表現とした. コード変換のプログラムは, JIS X 0208 からこのコード系に変換する.

フル4バイトへの変更点を表2に示す. 新規に作成したプログラムは, LitConv 文字定数変換ツール (言語Cで 600 行), JIS X 0208 から ISO 10646 へのコード変換のツール(言語Cで50行)である. 変更作業は, OS/omicron 上で行い, ツール作成開始から, ターゲットコンパイラ完成まで, 1人で6日間の労力を要した. またフル4バイトコンパイラを SUN-OS (SUN 3) 上にも移植した.

フル2バイト化時に比べ, 極めて早く変更できた理由は次のとおりである.

(1) 変更箇所が減っていること
(2) クロスコンパイル作成の回数が減っていること
(3) 文字(列)定数の変換が自動化されたこと
以前はコード表を見ながらの変更であった.
(4) マルチバイト化に伴うバグは修正されていたこと
(5) 開発環境が向上したこと
以前はフロッピーディスクベースの CP/M-68K であった.

次に第3章の方式との長所/短所を示す.

(1) 第3章の方式
長所：
・パーザの変更だけでクロスコンパイラを入手できる
・プロトタイプしながら開発できる
短所：
・無駄な変更を必要とする
パーザにおいて，中間コードの識別子変換の変更を必要とし，しかも後でその部分を廃棄する
・クロスコンパイラを2回作成しなくてはならない
(2) 改良した方式
長所：
・変更を少なくすることができる
・ツールで定数変換を行ったので，変更の手間が減った
・クロスコンパイラ生成は1回
短所：
・変更を一時に全フェーズに対して行う必要がある
全フェーズの変更終了後でないとクロスコンパイラをデバッグできない．

5. 考　　察

単に文字型をマルチバイトとするコンパイラを得たいときは，第4章の LitConv のようなプリプロセッサを用いることが考えられる．しかし，リンカをマルチバイトに対応しないと，外部名に対してマルチバイトコードを扱えない．また，このような方式を採用すると，マルチバイトコードを内部コードとするシステム上でこのコンパイラを利用できなくなるという問題がある．

次に，本論文で提示した二つの方式について議論する．短期間にマルチバイトのクロスコンパイラを作成し，しかも，その後セルフコンパイラを作成する必要のあるときは，第3章の方式が有利となる．この方式に第4章の方式の文字定数を変換するツールで文字定数を処理すれば，さらに容易にマルチバイトのクロスコンパイラを作成できる．

改良した方式は，全フェーズを同時にマルチバイト化する変更が必要なためデバッグなどに不利だが，ソースコードの変更を最小限にとどめることができる．

改良した方式において，言語Cのソースコードの文字（列）定数変換を行うのでなく，コード生成またはアセンブラで（またはその前で）定数変換や識別子のコード変換を行う方法もある．しかし，ツールの作成が容易なことから言語Cの文字(列)定数を変換することにした．また，この方式は，中間言語やアセンブリ言語を生成せず直接機械語を生成するコンパイラにも適用できる．

本方式の実現に用いた言語Cの機能を次に述べる．
(1) マクロまたは typedef による型の再定義
クロスコンパイラ作成では，1バイトの文字型の上で2バイトの文字型を定義しなくてはならない．このために，型の再定義を行う機能が必要である．本論文の実現ではマクロを用いたが，typedef でも可能である．
(2) 文字と数値の等価性
文字型を再定義すると同時に，文字コードも変更しなくてはならない．その場合，変更された文字コードをクロスコンパイラ中で文字定数として記述することはできない．文字型の値と数値を演算できなくてはならない．言語Cは型変換が柔軟なので容易にこの変更を行うことができた．
(3) 文字列と数値配列の等価性
文字定数の変更と同時に，文字列定数をマルチバイトコードの値を持つ数値配列としなくてはならない．言語Cの文字列定数中の8・16進定数を用いてこの変更を行った．

上記機能を有するプログラミング言語であれば，本方式を適用できる．従来の1バイト系で作成されたコンパイラに対しては，第3章で述べた注意点をチェックした後，本方式を適用しマルチバイト化を行える．

最後に，他の自己記述された他言語のコンパイラのマルチバイト化を考える．日本語 CLU では，CLU の抽象データ型機構が便利であったと述べている[18]．しかし，抽象データ機構のランタイムルーチンの変更を必要とする．特に，アセンブリ言語によって記述されているときは，アセンブリ言語に対するプリプロセッサを作成するのが便利であろう．

Pascal においては，型を再定義し，ord のように文字の順序を得る関数を変更すれば(1)と(2)は達成される．しかし，(3)が困難である．文字列定数を数値配列にすればよいが，保守性の悪いコンパイラとなるであろう．中間コードインタプリタと組み合わせたコンパイラでは，文字定数の変換をインタプリタで行う方法が優れている[19]．

システム記述言語ではないが，Fortran 90[9] では，文字集合を規定すると同時に，それ以外の文字コードを扱うための文字型と文字定数を言語構文として規格化している．この種の構文を有する言語では，マルチ

バイトコードの文字列を表現できるので，より容易にコンパイラのマルチバイト化が可能になる．

6. おわりに

本論文では，言語C処理系の関与するすべての文字コードをマルチバイトコードとする，マルチバイトコンパイラの実現方式について述べる．

パーザを変更するだけでクロスのマルチバイトコンパイラを実現し，さらにターゲットコンパイラの実現方式を整理・体系化した．この方式を用いて，言語Cコンパイラ CAT をフル2バイト化した．フル2バイトの CAT を用いて，フル2バイトコードを内部コードとする OS/omicron を開発し，さらに，そのプログラミング環境として CAT を使用している．

CAT のフル2バイト化の方式を改良し，ソースプログラムの変更を減らす方式を提示した．この方式を，フル2バイトから ISO 10646 フル4バイトコードへのコンパイラの変更に適用し，その有効性を確かめることができた．

参考文献

1) Unicode 1.0, The Unicode Consortium (1990).
2) Universal Coded Character Set, ISO/IEC DIS 10646 (1990).
3) 和田英一：ISO DP 10646 第2版および Unicode, NewsSITE, No. 5, pp. 1-6 (1990).
4) 木下 恂：標準プログラミング言語における日本語処理，情報処理，Vol. 26, No. 3, pp. 226-232 (1985).
5) 高橋延匡，武山潤一郎，並木美太郎，中川正樹：MC 68000 用小型 OS：OS/*o* の開発，情報処理学会計算機システムの制御と性能評価研究会，21-6 (1983).
6) 鈴木茂夫，小林伸行，田中泰夫，中川正樹，高橋延匡：OS/omicron における日本語プログラミング環境，情報処理学会論文誌，Vol. 30, No. 1, pp. 2-11 (1989).
7) 並木美太郎，屋代 寛，田中泰夫，篠田佳博，藤森英明，中川正樹，高橋延匡：OS/omicron 用システム記述言語C処理系 Cat のソフトウェア工学的見地からの方式設計，電子情報通信学会論文誌，Vol. J 71-D, No. 4, pp. 652-660 (1988).
8) 中田育男：プログラミング言語における日本語化の現状と今後の方向，情報処理学会プログラミング言語研究会，16-6 (1988).
9) Fortran 90, ANSI X 3J3-198 (1991).
10) 床分真一，今城哲二：COBOL における日本語機能の現状と今後の動向，情報処理学会プログラミング言語研究会，16-9 (1988).
11) 石田晴久：UNIX の日本語化機能と日本語対応Cコンパイラ，bit 別冊「最新 UNIX」，pp. 132-138 (1987).
12) 吉田和幸，牛島和夫：SNOBOL 4 既存処理系への日本語テキスト処理機能の追加，コンピュータソフトウェア，Vol. 2, No. 3, pp. 88-98 (1985).
13) Earley, J. and Sturings, H.: A Formalism for Translator Interactions, *CACM*, Vol. 13, No. 10, pp. 607-617 (1970).
14) 屋代 寛，中川正樹，高橋延匡：OS/*o* 第2版とシステム記述言語C，情報処理学会オペレーティング・システム研究会，32-3 (1986).
15) 田中泰夫，中川正樹，高橋延匡：OS/*o* 用言語Cコンパイラ Cat の日本語化の方式とその実現，第34回情報処理学会全国大会論文集，3 V-5, pp. 833-834 (1987).
16) Souya, T., Hayakawa, E., Honma, M., Fukushima, H., Namiki, M., Takahashi, N. and Nakagawa, M.: Programming in a Mother Tongue: Philosophy, Implementation, Practice and Effect, *Proceedings of the 15th COMPSAC*, pp. 705-712 (1991).
17) 中村昭次，久野 靖，木村 泉：日本語 CLU システムの試作，第30回情報処理学会全国大会論文集，1 R-7, pp. 461-462 (1985).
18) 久野 靖：CLU とその仲間達（4）～CLU で OS を作った話（その1），*bit*, Vol. 21, No. 9, pp. 1205-1212 (1989).
19) 国技義敏，島崎真昭，津田孝夫：PASCAL をベースとした日本語データ及び漢字を用いるプログラミング，第21回情報処理学会全国大会論文集，51-3, pp. 1015-1016 (1980).

（平成4年1月 9 日受付）
（平成4年9月10日採録）

並木美太郎（正会員）

昭和59年東京農工大学工学部数理情報卒業．昭和61年同大学院修士課程修了．同4月(株)日立製作所基礎研究所入社．昭和63年より東京農工大学工学部数理情報助手．平成元年4月より電子情報助手．並列処理，日本語情報処理のソフトウェア／ハードウェアアーキテクチャに興味を持ち，コンパイラ，オペレーティングシステムなどシステムプログラムの研究・開発に従事する．

早川 栄一（正会員）

平成元年東京農工大学工学部数理情報工学科卒業．平成3年同大学大学院博士課程前期修了．同年4月同大学大学院博士後期課程入学，現在に至る．オペレーティングシステムなどのシステムソフトウェアの研究に従事．

下村　秀樹（正会員）

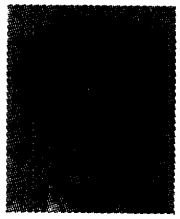
昭和40年生．平成2年東京農工大学大学院修士課程（数理情報工学専攻）修了．同年同大学院博士後期課程進学，現在に至る．日本語情報処理，特に仮名漢字変換，日本語文章作成支援環境の研究に興味を持つ．

田中　泰夫（正会員）

昭和61年東京農工大学工学部数理情報卒業．昭和63年同大大学院修士課程修了．在学中，システムソフトウェアの開発に従事．

中川　正樹（正会員）

昭和52年東京大学理学部物理卒業．昭和54年同大学院修士課程修了．同在学中，英国 Essex 大学留学 (M. Sc. in Computer Studies)．昭和 54 年東京農工大学工学部数理情報助手，平成元年1月数理情報助教授，同4月電子情報助教授．オンライン手書き文字認識，日本語計算機システム，文書処理の研究に従事．理学博士．



高橋　延匡（正会員）

昭和 8 年生．昭和 32 年早稲田大学第一理工学部数学卒業．同年(株)日立製作所中央研究所入社，HITAC 5020 モニタ，TSS の開発に従事．昭和 52 年より東京農工大学工学部数理情報教授．平成元年電子情報教授．理学博士．オペレーティングシステム，日本語情報処理，パターン認識の研究に従事．電子情報通信学会，ソフトウェア科学会，計量国語学会，ACM 各会員．