COBOL マシンとその設計思想†

——アーキテクチャについて——

山本昌弘†† 中崎良成††
横田 実†† 箱崎勝也††

本論文は開発した COBOL マシンのアーキテクチャについて，その特徴と設計思想に重点をおいて論じている．COBOL マシンは COBOL で書かれたソースプログラムに簡単な変換を行って作られた中間言語プログラムを直接実行する高級言語マシンで，汎用計算機に接続されて付加プロセッサとして動作する．そして，COBOL の演算処理，データ操作，表操作，実行順序制御などの機能は COBOL マシンで実行し，入出力処理や通信制御などの機能はホストプロセッサである汎用計算機で実行される．

COBOL マシンのアーキテクチャは COBOL の言語仕様にできる限り近づけるように設定されており，(1)多種類の内部データ形式，(2)複雑な属性のデータを効率良く表現するデータディスクリプタ，(3)高度なデータアクセス機構，(4)ソースステートメントと直接対応する高機能命令，(5)ホストプロセッサとの高度なインタフェース機能，などを備えている．

その結果，COBOL の大部分のステートメントは COBOL マシンの1命令へ展開でき，翻訳処理が高速化され，オブジェクトメモリ量も大幅に減少することが明らかになった．また，COBOL マシンを高性能なハードウェアで実現することにより，実行速度も向上する．

1. はじめに

著者らは COBOL で書かれたプログラムを中間言語[1]へ変換した後実行する COBOL マシン[2]を開発したが，本論文では COBOL マシンのアーキテクチャについて論じている．

本 COBOL マシンは汎用計算機に接続されて付加プロセッサとして動作し，COBOL の演算，条件，データ操作，実行順序制御などの中核ステートメントに対する処理を高速に実行する．一方，入出力や通信制御などのステートメントに対する処理はホストプロセッサである汎用計算機で実行される．また，ソースプログラムから中間言語への翻訳処理はホストプロセッサ上の翻訳ソフトウェア[3]によって行われる．**図1**は COBOL マシンを用いたシステム構成の例を示す．

中間言語方式の高級言語マシンの開発において重要な点は中間言語をどのレベルに設定するかである．本 COBOL マシンの中間言語に当るマシンアーキテクチャは COBOL 言語仕様にできる限り近づけることを念頭に，特に次の点を考慮して設計を行った．

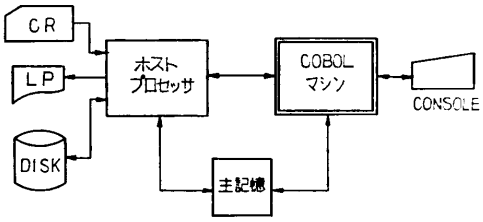


図 1 COBOL マシンを用いたシステム構成例
Fig. 1 System configuration example including COBOL machine.

1）豊富なデータ形式の設定

2）複雑な属性や構造を備えたデータに関する統一的な表現

3）高度なデータアクセス機構

4）言語仕様で規定されるステートメントと対応する高機能命令の設定

また，本 COBOL マシンは付加プロセッサ形式の高級言語マシンのために，ホストプロセッサとの効率良いインタフェース機能を備えている．

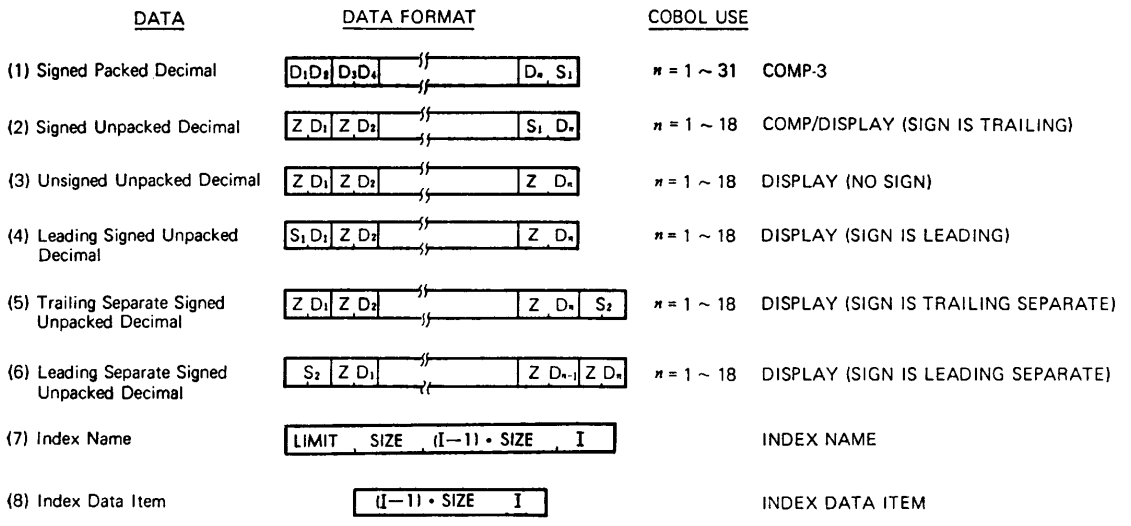
なお，本 COBOL マシンは ANSI 74 COBOL 言語仕様[4]に準拠するプログラムを実行できる．

2. 内部データ形式

COBOL プログラムのデータ部で使用できるすべて

† COBOL Machine and its Design Concept —Machine Architecture— by MASAHIRO YAMAMOTO, RYOSEI NAKAZAKI, MINORU YOKOTA and KATSUYA HAKOZAKI (C & C Systems Research Laboratories, Nippon Electric Co., Ltd.).

†† 日本電気(株) C & C システム研究所



D: Digit (0 ~ 9), Z: Zone (1111_2),
S_1: 4 bit sign code (1100_2; positive, 1101_2; negative)
S_2: 8 bit sign code (01001100_2; positive, 01000000_2; negative)
LIMIT, SIZE, I: Table size, element size, occurrence number

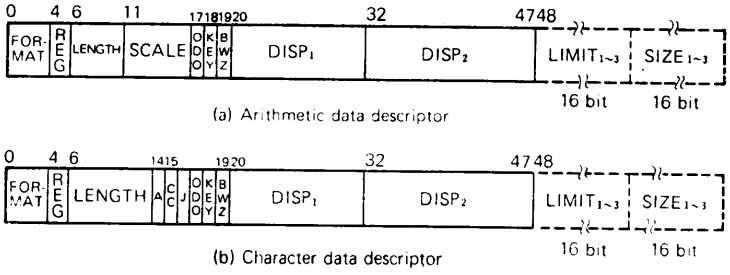
図 2 十進データおよび指標データの形式
Fig. 2 Decimal data and index data formats.

のデータタイプに対応する内部データを設けた．COBOL の代表的なデータである十進データについてみると，6種類のデータ形式を使用できる．しかし，汎用計算機では通常2種類（パック型十進とゾーン型十進）しか備えていないために，対応する内部データを持たない十進データについては，内部処理によって前記2種類のいずれかに対応させることによって実現されている．また，演算に際して，データタイプの変換が必要になる．このために，COBOL マシンは COBOL で使用できるすべての十進データに対応する6種類の内部データを備えている．その結果，COBOL マシンでは対応づけのための内部処理やデータタイプの変換処理を行わなくてもよい．すなわち，これらの処理はハードウェアによって実現されることになる．

この他，二進数，英字，英数字，編集のために用いられる編集項目や編集制御データ，表の要素を規定するために用いられる指標データや指標データ項目などを備えている．**図2**は十進データと指標データの形式を示す．

3. データディスクリプタ

COBOL では，表の指定，十進データの小数点，四捨五入，丸め指定および編集指定などが可能である．このような複雑な属性を備えたデータを統一的に表現するために，**図3**に示すように2種類のデータディスクリプタを設けた．データディスクリプタは6バイト



FORMAT: Data Format
REG: Register Specification
LENGTH: Data Length
SCALE: Integer Part Length
ODO: Occurrence-Depending-On Specification
KEY: Ascending or Descending Key
BWZ: Blank When Zero
$DISP_1$, $DISP_2$: Displacement in a Segment
LIMIT: Table Size
SIZE: Element Size
A: All Clause Specification
CC: Code Conversion
J: Justify

図 3 データディスクリプタの形式
Fig. 3 Data descriptor formats.

が基本形で，表の次元数に応じて4バイト単位で増加する．しかし，データディスクリプタを介するアクセスは余分なメモリアクセスを必要とし，性能の低下につながるため，できる限り通常のオペランドを用いるように考慮した．

4. データへのアクセス機構

COBOL が使用される応用分野では複雑な演算処理は比較的少なく，MOVE などの簡単なデータ操作命令が頻繁に用いられる．このために，COBOL マシンでは，データへのアクセスを効率良く行う機構を備えた．

4.1 オペランド形式

オペランドは複数個のオペランドシラブルで表現され，オペランドシラブルとして4種類設けられている．

（1） リテラル型：8ビット以内のデータを直接オペランド部で指定する．短い長さの数値，文字定数，表意定数などはこれを用いて表現される．

（2） ダイレクト型：汎用計算機の通常のオペランド形式に相当するもので，主記憶の比較的構造の簡単なデータをアクセスするのに用いられる．英字，英数字，二進数，整数形式の十進数はこれを用いてアクセスされる．表データも指定可能である．また，桁の合った小数点つき十進数はこの形式を用いてアクセスできる．

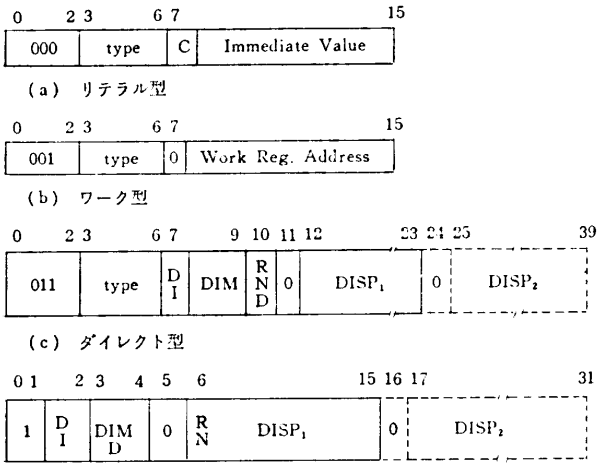


図 4 オペランドシラブル形式
Fig. 4 Operand syllable formats.

（3） インダイレクト型：ダイレクト型で表現できない複雑なデータ構造をデータディスクリプタを介してアクセスする方法である．桁合わせが必要な小数点つき十進数や編集項目データなどはこれを用いてアクセスされる．

（4） ワーク型：COBOL マシンに内蔵された高速レジスタをアクセスするのに用いられる．1つの COBOL ステートメントが複数個の COBOL マシンの命令へ展開される時には命令間で受け渡しされるデータが必要になる．これらのデータは主記憶の代りに高速レジスタに割り当てられ，ワーク型オペランドを用いて管理される．

これらの4つのオペランドシラブルの一例を**図4**に示す．

4.2 表データへのアクセス

COBOL では最大三次元までの表データを指定でき，表の要素を指定するのに指標データや添字データを用いる．表の要素へのアクセスは，たとえば，$A(d_1, d_2, d_3)$（ただし，$d_1 \sim d_3$ は各次元の大きさ）と宣言された三次元の表の1つの要素 $A(i_1, i_2, i_3)$ にアクセスするには，汎用計算機では，次式に示す要素の上限検査とアドレス計算を通常の四則演算と比較命令を実行することにより行われる．

$$\left.\begin{array}{l} 1 \leqq i_1 \leqq d_1,\ 1 \leqq i_2 \leqq d_2,\ 1 \leqq i_3 \leqq d_3 \\ A+(i_3-1)\times a+(i_2-1)\times a\times d_3 \\ \qquad +(i_1-1)\times a\times d_3\times d_2 \end{array}\right\} \quad (1)$$

ただし，a は表の基本要素のサイズ，A は表のベースアドレスを示す．また，表の要素は記憶装置では，$A(1,1,1)$, $A(1,1,2)$, $A(1,1,3)\cdots A(1,2,1)$, $A(1,2,2)\cdots$ の順に配置されているものとする．

これに対して，COBOL マシンでは，各次元に対応する指標または添字データを示すオペランドを並べるだけで，ハードウェアによって（1）式と等価な処理が行われる．**図5**に一例を示す．

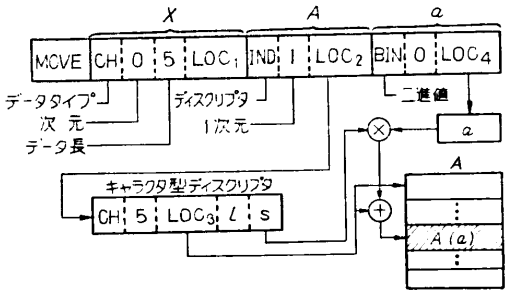


図 5 アドレス展開過程 (MOVE X TO $A(a)$)
Fig. 5 Address development process for MOVE X TO $A(a)$.

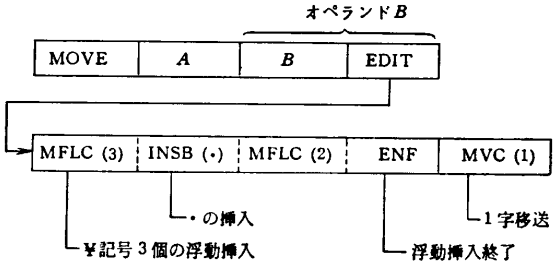


図 6 編集マイクロオペレータの例
(*B*が ¥¥¥¥. ¥¥9の場合)

Fig. 6 Editing microoperator for ¥¥¥¥. ¥¥9.

表 1 ベースレジスタの使用規則

Table 1 Base register usage.

ベース レジスタ	使 用 目 的
0	実行中の COBOL プログラムを呼んだ親プロセスへのポインタ
1	スタックセグメントアドレス
2	データセグメントアドレス
3	ディスクリプタセグメントアドレス
4	COBOL マシンコードセグメントアドレス
5	ホストプロセッサとの通信データ保存用
6	ホストプロセッサとの通信データ保存用
7	リンケージセグメントアドレス

4.3 編集項目の表現

編集処理は COBOL において頻繁に用いられる機能の1つである．COBOL マシンでは，編集項目であることを示すオペランドと編集制御データを示すオペランドの対によって指定される．編集制御データは編集動作を規定する編集マイクロオペレータの列で表現され，**図6**は浮動挿入編集の例を示す．

4.4 仮想アドレス機構

汎用計算機と同等のアドレス機構を提供するために，ACOS システム 300 と同一のセグメンテーション方式によるアドレス機構[5]を採用した．1つのCOBOLプログラムが翻訳されると，複数個のセグメントに分割され，各セグメントは8個の内の固有のベースレジスタを介してアクセスされる．**表1**はベースレジスタの使用規則を示す．

5. マシン命令

COBOL マシンのマシン命令の設定に当り，以下の点に留意した．

1) 1つの COBOL ステートメントは原則として1つの命令へ変換できるようにする．これによって，変換のためのソフトウェアが簡単になる．また，実行時には，元のソースステートメントとの対応がつくために，ソースステートメントの動作を反映した高度な処理（たとえば，複雑な PERFORM や SEARCH 命令では，ループ動作を行うように命令列の実行順序を決定する）を行える．

2) データ操作ステートメントなどは，基本操作を何個でも組み合わせた形を表現できるために，非常に長く複雑である．したがって，これらを1つの命令に対応づけることによって得られる利点は少なく，むしろハードウェアを複雑にするだけである．このため，それらのステートメントの基本動作に対応する処理を行う命令を組み合わせることによって実現する．

5.1 命 令 形 式

COBOL マシンの命令は演算の種類を規定する命令コードと複数個のオペランドおよび必要に応じて付加され，命令コードを補うバリアントから成っている．命令コードにはソースステートメントとの対応を示す情報が付加されている．また，**第4章**で述べたように，オペランドは複数個のオペランドシラブルで構成される．すなわち，オペランドがリテラル，ワーク，表形式でないデータでは1つのオペランドシラブルで規定される．一方，表データの場合は次元の個数に等しい指標または添字データを示すオペランドシラブルが付加される．

5.2 基 本 命 令

四則演算，移送などの大部分の COBOL ステートメントと一対一に対応する命令を備えている．**図7**は ADD ステートメントの例を示す．オペランドのデータタイプは命令コードではなく，オペランドシラブルで指定される．このほか，減算，乗算，除算や移送などの命令を備えている．

5.3 多数オペランド命令

COBOL には複数個のオペランドを指定できるステートメントがある．**図8**は複数オペランドを持つ MOVE ステートメントとそれに対応する COBOL マシンの命令を示す．しかし，**図9**に示されるような両辺が複数オペランドのものについては2命令へ展開さ

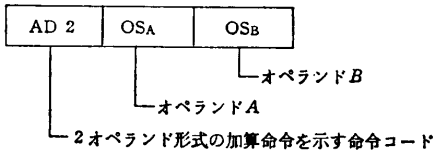


図 7 ADD ステートメントの例
(ADD *A* TO *B*)

Fig. 7 ADD statement example.

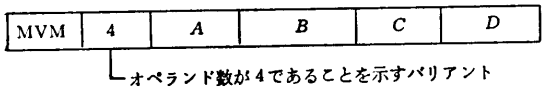


図 8 MOVE ステートメントの例
(MOVE *A* TO *B C D*)

Fig. 8 MOVE statement example.

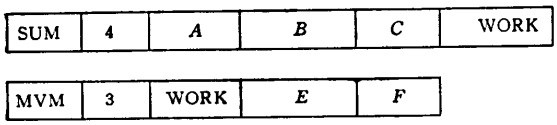


図 9 複雑な ADD ステートメントの例
(ADD *A B C* TO *E F*)

Fig. 9 Complex ADD statement example.

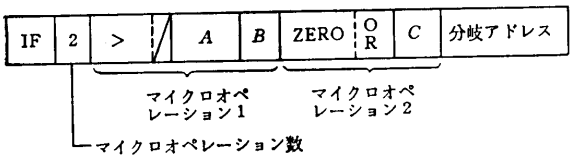


図 10 複合条件を持つ IF ステートメントの例
(IF *A* IS > *B* OR *C* IS ZERO THEN ～)

Fig. 10 IF statement example including a compound condition.

れる.

5.4 条件命令

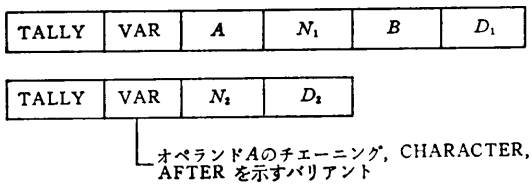
COBOL の IF ステートメントでは単純条件から複合条件まで表現できる. これらと一対一に対応する命令を備えるために, 編集マイクロオペレータと同様に, 1つの単純条件のオペレーションを規定する条件マイクロオペレーションを設けた. **図 10** は COBOL の複合条件の例を示し, 複合条件は単純条件を示すマイクロオペレーションを組み合わせることによって実現される. 図 10 の例ではマイクロオペレーション 2 によって, ゼロ判定の結果とこれまでの結果 (今の例では $A>B$ の判定) との論理オアが作られる. そして, この結果を用いて, 複合条件全体の判定がなされる.

5.5 データ操作命令

これまで述べたように, ほとんどの COBOL ステートメントは1つの COBOL マシンの命令へ対応づけられる. しかしながら, COBOL には, INSPECT, STRING, UNSTRING などのデータ操作ステートメントのように, 長くて複雑なステートメントが含まれている. これらを1つの命令へ対応づけることは可能であるが, それらを実現するためのハードウェアを複雑にする割には効果が小さい. そこで, COBOL マシンでは, これらのステートメントの機能を分解し, それらの基本処理を行う命令を設けた. そして, 複雑なものは基本処理を組み合わせることによって実現する方式を採用した. **図 11** は INSPECT ステートメントの例を示す. このステートメントは, (1)走査対象文字列 A をしらべ, D_1 で示される文字列が現われる前に, B の示す文字列が出現した回数を求める, (2)同じ文字列 A をしらべ, D_2 で示される文字列が現われた後のすべての文字数を求める, の2つの処理に分解される. したがって, INSPECT の場合には, 指定された条件のもとで, 1つの文字列をしらべ, 指定された文字列の出現回数を求めたり, 別の文字列に置換 (REPLACE 指定の時) する処理が基本処理と考えられる. このために, 図 11 の場合には, COBOL マシンの2つの数え上げ命令 (TALLY) で実現される.

INSPECT *A* TALLYING N_1 FOR ALL *B* BEFORE D_1
N_2 FOR CHARACTER AFTER D_2

(a) COBOL ソースステートメント



(b) 対応する COBOL マシン命令

図 11 INSPECT ステートメントの例

Fig. 11 INSPECT statement example.

このような考え方を, ほかのステートメントについても適用して基本処理を決定した.

STRING ステートメントでは, 複数個の文字列データから指定された1つの条件を満たす部分文字列をそれぞれ取り出し, 指定された領域へ移送して結合する処理が基本処理である. また, UNSTRING では, 1つの文字列について, 指定された条件のもとで複数個の部分文字列に分解し, それぞれ指定された領域へ移送する処理が基本処理である.

これらのデータ操作ステートメントでは, 命令長を短くするために, オペランドの省略を可能にしている. 図 11 の例のように, 走査対象であるオペランド A が2番目の TALLY 命令においても再び用いられる場合には, 第2の TALLY 命令では, バリアント部のチェーニング指定を用いて, オペランド A を省略している.

5.6 実行順序制御命令

COBOL の PERFORM, SEARCH ステートメン

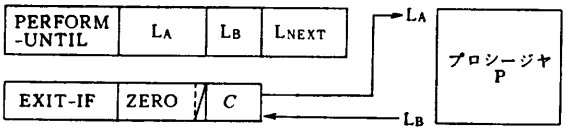


図 12 PERFORM ステートメントの例
(PERFORM *P* UNTIL *C* IS ZERO)
Fig. 12 PERFORM statement example.

トはループ動作などの実行順序を指示する.

PERFORM ステートメントでは, 対象のセクションまたはパラグラフを1回または*n*回実行するような単純な場合は COBOL マシンの PERFORM 命令へ対応づけられる. 一方, 複雑な PERFORM や SEARCH ステートメントはこれらの動作を指定する PERFORM や SEARCH 命令とループ動作終了条件を判定する IF 命令により実現される. **図12**は PERFORM ステートメントの例を示す. PERFORM-UNTIL 命令により, 制御用スタックにアドレス L_A, L_B, L_{NEXT} が作られる. EXIT-IF 命令は通常の IF 動作を行った後, *C*がゼロでなければスタック中のアドレス L_A を用いてプロシージャ*P*を実行するように制御する.

5.7 ホストプロセッサインタフェース命令

入出力処理, プログラム間連絡機能, 通信処理などをホストプロセッサに実行を依頼するためにホストプロセッサ呼び出し命令を備えている. この命令は処理の種類やパラメータなどを指定する.

また, ホストプロセッサには, COBOL マシンを起動するための命令が設けられている.

6. オブジェクトプログラムの構成

翻訳ソフトウェアによって生成されるオブジェクトプログラムは以下の5種のセグメントで構成され, ホストプロセッサと共用される主記憶を介して COBOL マシンに渡される.

1) リンケージセグメント: プログラムの実行開始アドレス, 各セグメントのベースアドレスなどを格納し, 表1のベースレジスタ7を用いてアクセスされる.

2) データセグメント: プログラムで使用される定数や変数を格納する. 複数個のデータセグメントから成り, ベースレジスタ2 (表1) を用いてアクセスされる.

3) ディスクリプタセグメント: データディスクリプタを保存するセグメントで複数個から成り, ベースレジスタ3を用いてアクセスされる.

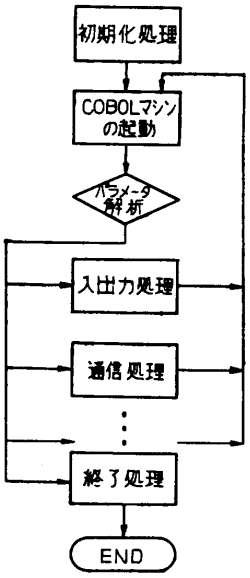


図 13 CSP の構成
Fig. 13 CSP configuration.

4) コードセグメント: COBOL マシンの命令列から成り, COBOL マシンにより実行される. そして, ベースレジスタ4で示される.

5) ホストプロセッサコードセグメント: COBOL マシンの起動, 停止処理, COBOL マシンで実行されない入出力処理などを行うためのホストプロセッサ命令列で構成される. **図13**はその構成を示し, COBOL マシンサポートプログラム (CSP) と呼ぶ.

オブジェクトプログラムの実行はホストプロセッサの制御プログラムにより, 1つのジョブステップとしてスケジュールされ, COBOL マシン上で実行される.

7. むすび

COBOL マシンのアーキテクチャは多種類の内部データ, 高度なアドレッシング機能, 高機能命令を備えている. その結果, COBOL の大部分のステートメントは COBOL マシンの1命令へ展開できることが明らかになった. これによって, 翻訳処理が高速化され, オブジェクトメモリ量も大幅に減少することが期待される[6]. しかし, 実行時には高度な処理が要求されることになるが, これについては, 高性能なハードウェアによって効率良く実現されることを想定している.

なお，ハードウェア構成および性能評価についての報告は別の機会にゆずることにする．

謝辞 本 COBOL マシンは通産省工業技術院大型プロジェクト「パターン情報処理システム」の一環として開発したもので，本プロジェクトの関係各位に深謝します．また，本 COBOL マシンの開発に当り，日頃有益なご指導を頂く当社ソフトウェア生産技術研究所祢津所長代理，およびアーキテクチャ開発に参加下さった当研究部永井主任，梅村主任，熊野氏に深謝します．

参考文献

1) 山本，箱崎，梅村，永井，熊野，中崎：COBOL マシンのアーキテクチャについて，情報処理学会第 17 回全国大会論文集，pp. 307-308 (1976).
2) 中崎，山本，箱崎，梅村，横田：COBOL 用高級言語マシン，情報処理学会マイクロコンピュータ研究会，No. 2-1 (1977).
3) 山本昌弘：COBOL 用高級言語マシントランスレータ，電子通信学会情報・システム部門全国大会論文集，p. 416 (1979).
4) American National Standard Programming Language COBOL, X 3.23 1974, American National Standard Institute, Inc., (1974).
5) 北村，市古：ACOS シリーズ 77 NEAC システム 300/400/500 のアーキテクチャ，日本電気技報，No. 114, pp. 6-12 (1975).
6) 横田，中崎，山本：COBOL 用高級言語マシンのアーキテクチャ評価，情報処理学会第 21 回全国大会論文集，pp. 139-140 (1980).

(昭和 56 年 5 月 25 日受付)
(昭和 56 年 7 月 13 日採録)