（1977. 9. 14）

COBOL用高級言語マシン

中崎良成　山本昌弘　箱崎勝也　梅村護
横田実　（日本電気株式会社　中央研究所）

1. 序

各種の高級言語について，これまで多くの高級言語マシン・アーキテクチャ[1]が提案され，ファームウェア及びハードウェア技術を用いて実現されている。しかしながら，これらの殆んどすべては単に汎用計算機のマイクロプログラム技術を用いて，又はICやMSI等の小規模半導体素子を用いて構成されている。本稿ではCOBOL向き高級言語マシン・アーキテクチャ[2] COMBAT（Cobol Oriented Machine Basic Architecture）とそのハードウェア構成[3]について述べる。COMBATを実現するCOBOLマシンはマイクロプログラム技術とLSIやVLSI素子を用いて作られる高機能かつ高性能なハードウェア素子〈例えば，高速RAM，マイクロプロセッサ，プログラマブル・ロジック・アレイ（PLA）〉とを有効に組み合わせて構成される。

COBOLプログラムは通常のコンパイラに対応するソフトウェア・トランスレータにより，高レベルなCOMBAT命令へ変換される。ソースプログラムレベルでユーザーが指定するCOBOL処理はCOMBATマシンの命令レベルで効率よく行なわれる。COMBATマシンの命令はCOBOLのソースステートメントに近いため，COBOLプログラムからCOMBATマシン命令への変換は通常の計算機のコンパイラよりも簡単になる。

COMBATマシンの命令はマイクロプログラム技術と高度な機能を処理することを考慮して作られた専用ハードウェア素子のサポートにより，効率良く実行される。

本高級言語マシンはスタンド・アロンの言語処理用専用プロセッサとして，又分散処理システムを構成する言語プロセッサとして使用することができる。以下では，COMBATマシン・アーキテクチャ，ハードウェア構成，性能評価結果について述べる。

2. マシン・アーキテクチャ

COMBATはCOBOL指向の高級言語マシン・アーキテクチャである。このアーキテクチャはCOBOLで頻繁に使用される，複雑なデータ構造へのアクセス，高度なデータ操作や表操作を効率良く実現する機能を備えている。そして，マイクロプログラム技術と高度なハードウェア機能モジュールを用いることにより，効果的に実現されることを意図して設定された。

殆んどのCOBOLステートメントは1つのCOMBAT命令へ変換することができる。又，複雑なステートメントではそのステートメントの基本機能を実行するCOMBAT命令を用意し，この COMBAT 命令を複数個合わせたものへ変換される。更に，COBOL のデータ部（DATA DIVISION）でユーザーが定義したすべてのデータに対応する多くの内部データを備えている。複雑なデータ属性を持った複雑なデータ構造を効率良く処理するために，データ・ディスクリプタを導入している。又，データタイプの変換，十進データの小数点位置合せ，表形式のデ

データへアクセスする時用いられる添字処理等が機械語命令レベルで自動的に行なわれる。

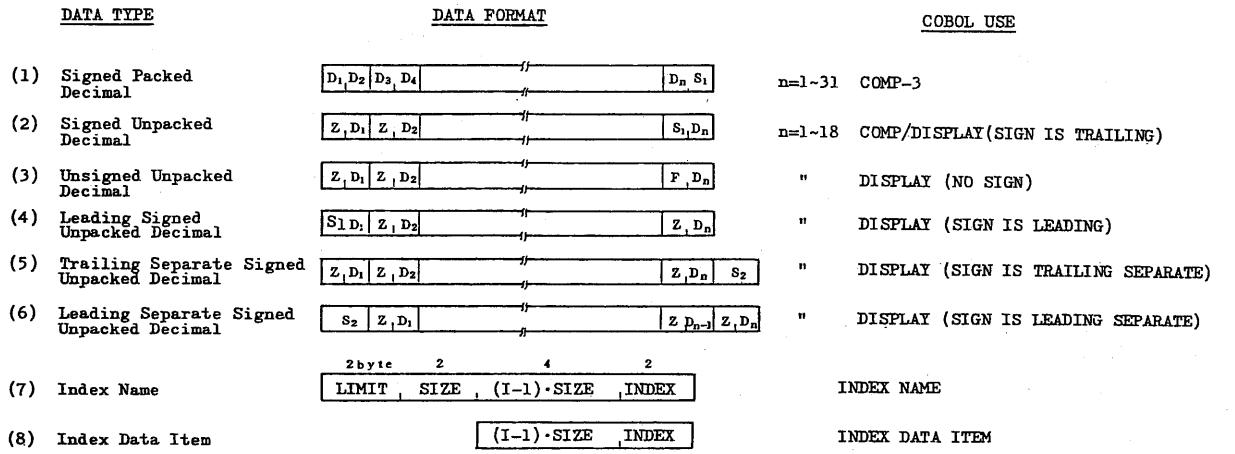
2.1 データ形式とデータ・ディスクリプタ

代表的な汎用計算機では命令レベルで直接処理できる十進データは1種又は2種である（ゾーン十進数とパック十進数）。しかしながら，COBOLユーザーは表示用として，多くの十進数データを使用することができる（例，前置符号付十進数，後置分離符号十進数）。COMBATではANS COBOL言語仕様[4]で利用できるすべての形式の十進数に対応する6種類の内部十進数を備えている。又，表の要素の参照はCOBOLの指標名や指標データ項目に対応する内部データを用いて容易に行なうことができる。図1は6種類の十進数，指標名，指標データ項目の構成を示す。

COBOLプログラムは複雑な構造や属性を備えたデータを扱うことができ，又PICTURE句で指定される編集操作のような高度な処理を要求することができる。このような処理を実現するために，2つのデータ・ディスクリプタを備えており，データ形式の変換，表形式のデータに関するアドレス境界検査，BLANK WHEN ZERO句等のためのゼロ又はスペース文字の挿入，四捨五入処理，編集処理等を指定することができる。図2はデータ・ディスクリプタの構成を示す。

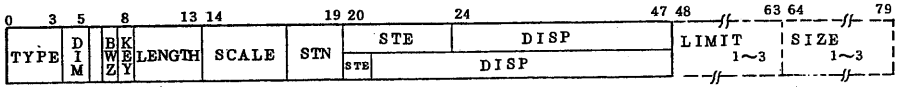
2.2 COMBAT 命令

COMBAT命令は命令コード，バリアントおよび複数個のオペランド・シラブルで構成される。COBOLプログラムを効率良く実行するために，強力なデータ・アクセス機能を備えている。COMBATでは単純なデータへアクセスするための簡単なオペランド・シラブル形式と複雑なデータへアクセスするための高度なオペランド・シラブル形式が用意されており，4種類のオペランド・シラブル形式があ

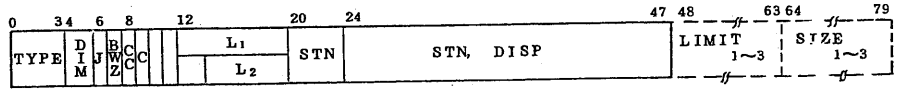


Where D : Digit (0~9), Z: Zone (1111_2),
S1 : 4 bit sign code (1100_2; positive, 1101_2; negative)
S2 : 8 bit sign code (01001100_2; positive, 01000000_2 : negative)
LIMIT: table or list size
SIZE : table element size
INDEX (I) : occurence number

図 1. COMBATの十進数データと指標データ形式



(a) arithmetic data descriptor



(b) character data descriptor

Where ; TYPE : Data Type
DIM : Dimension Number
LENGTH : Data Length
SCALE : Integer Part Length
BWZ : Blank When Zero
STN, STE: Segment Specification
LIMIT: Table Size
SIZE : Element Size
KEY : Ascending or Descending Key
J : Justify
CC : Code Conversion
C : ALL clause specification

図2. データディスクリプタ形式

る。命令中に直接データを備えたリテラル・シラブルはCOBOLソースプログラムの短い長さの数値定数，文字定数，表意定数を規定するために用いられる。又，1つのCOBOLソースプログラムが複数個のCOMBAT命令へ変換される時には，命令間で利用される中間データが必要になる。この中間データを規定するために，ワークレジスタ・シラブルが有り，高速メモリが割り当てられる。主記憶中にある通常の単純な属性のデータはダイレクトオペランド・シラブルによりアクセスされ，これは汎用計算機の通常のデータアクセス機構に対応している。インダイレクトオペランド・シラブルは複雑なデータをデータ・ディスクリプタを介してアクセスする時に用いられ，次の様な場合に使用される。

(1) 十進データで小数点位置が合っていない
(2) 添字づけによる表データへのアクセス
(3) 編集処理
(4) 四捨五入処理，JUSTIFY指定
(5) 可変長の表データへのアクセス

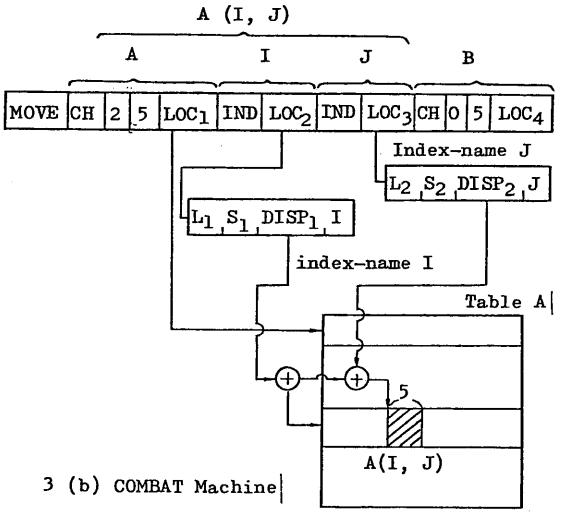
2つ以上の連続する命令間で共通に利用されるオペランドはバリアントのチェーニング指定によってオペランド・シラブルの省略を行なうことが可能である。従って，この機能を用いることにより，この様なオペランドは一度規定するだけで，連続する命令で利用されることになる。

COBOLプログラムで表の要素にアクセスするために頻繁に使用される指標や添字づけ動作は表全体を規定するオペランド・シラブルについて，各次元に対応する指標又は添字用オペランド・シラブルを羅列するだけで，機械語命令レベルで実現される。例えば，二次元の表の要素A(I,J)は通常の計算機では命令列を実行することにより達成される。即ち，例えばIBM 360型の計算機のCOBOLコンパイラは図3.aで示すオブジェクトコード列を生成する。これに対して，COMBAT命令では図3.bで示されるように，データA(I,J)は命令の中で非常に簡単に表現される。そして，データA(I,J)にアクセスする時には，命令で規定された表の開始アドレスと指標データを用いて該当アドレスを計算すると同時に，表の境界検査も行なわれる。

COMBATアーキテクチャでは通常の計算機の機械語命令に似た単純な命令から，COBOLのソースステートメントに対応するような高度なマクロ命令まで，約50種類のCOMBAT命令が設定されている。

```
LH   R1,   D1 (B1)      } Test I ≦ L1
CH   R1,   D2 (B1)      }
BC
LH   R1,   D3 (B1)      } Test J ≦ L2
CH   R1,   D4 (B1)      }
BC
L    R1,   D5 (B1)          DISP1 ⟶ R1
A    R1,   D6 (B1)          DISP + DISP2⟶R1
MVC  LOC4(5, B1), LOC1 (R1)
```

3 (a) System/360 like architecture



3 (b) COMBAT Machine

図3. 指標操作

COMBOL Source Statement

ADD A(I) TO B(b)

I : Index Name

b : Subscript Datum

Corresponding COMBAT instruction

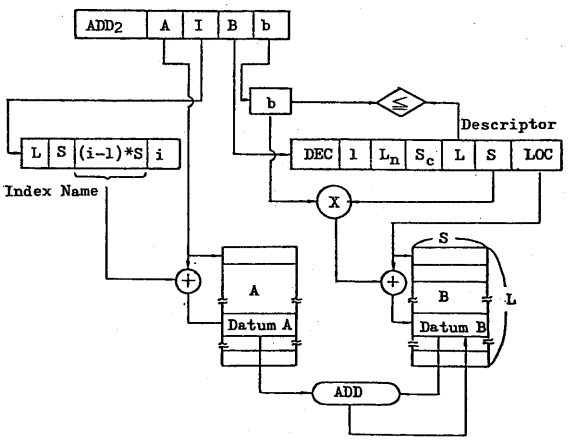


図4. COMBAT・ADD命令の例

COBOLの算術ステートメントや移送ステートメントは2，3，および多数オペランド形式のCOMBAT命令に対応する。図4はCOBOLのADDステートメントに対応するCOMBAT命令を示す。COBOLのSEARCHやPERFORMステートメントは非常に複雑である。このために，これらを1つのCOMBAT命令に変換することはハードウェアの設計を複雑にし，このステートメントの動作を考慮すると必ずしも1つの命令として実現することによる効果が期待できない。従って，COMBATアーキテクチャではこれらのステートメントを実行するために必要な基本動作を行なう命令を備え，これらの命令を組み合わせることにより実現している。例えば，SEARCHステートメントは2つの機能に分割できる。即ち，その1は表の要素アドレスの更新と表の境界検査を行なう機能，その2は現在の要素が求めるデータか否かをしらべる機能である。そして，SEARCHステートメントはその1の機能を行なうCOMBAT SEARCH命令とその2の機能を行なう通常の条件命令へ変換される。

同様に，PERFORMステートメントもCOMBAT PERFORM命令と条件命令へ変換される。COMBAT PERFORM命令はループ制御変数の更新と実行対象手続き部へ制御を渡す動作を行なう。条件命令はループの終了条件を評価する。

複雑なデータ操作ステートメント，INSPECT，STRING，UNSTRINGはSEARCHステートメントと同様に，一連のそのステートメントに対応する基本COMBAT命令へ変換される。図5はINSPECT命令の例を示し，2つのCOMBAT TALLY命令から構成される。1つのTALLY命令は1つの走査オペランドに関するINSPECT動作を実行する。

3. ハードウェア構成

性能／価格比を向上させるために，高い機械語レベルに適したマシン構成が必要となる。

COMBAT マシンは図6に示すように命令取出し(IF)プロセッサ，オペランド取出し(OF)プロセッサ，および実行(EX)プロセッサの3プロセッサで構成される。COMBAT命令はこれら3プロセッサによりパイプライン方式で実行される。このときの処理例を図7に示す。EXプロセッサが最初の命令を実行しているときにOFプロセッサは次に続く命令を実行するために必要なオペランド取出し処理を行っている。この時，IFプロセッサは3番目の命令を取り出す処理を行っている。

通常の汎用計算機の機械語に比べ，より複雑で高度なCOMBATアーキテクチャを実現するために各プロセッサではそれぞれ専用のハードウェアモジュールを利用している。例えば強力で低価格なバイポーラマイクロプロセッサあるいはPLAをはじめ，特殊なハードウェアモジュールとして制御情報生成器，データ変換器と小数点位置合せ器な

COBOL Source Statement

INSPECT S TALLYING N_1 FOR ALL B_1 BEFORE D_1

N_2 FOR CHAR AFTER D_2.

Corresponding COMBAT instructions

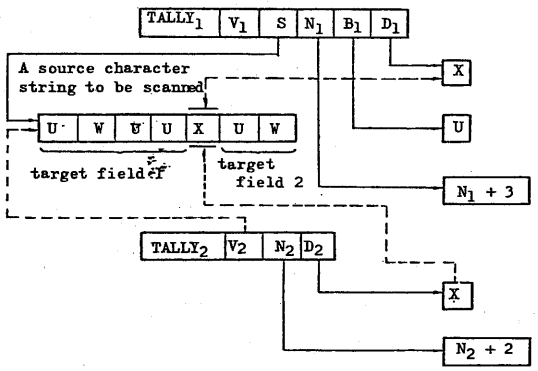


図5. COMBAT・TALLY命令の例

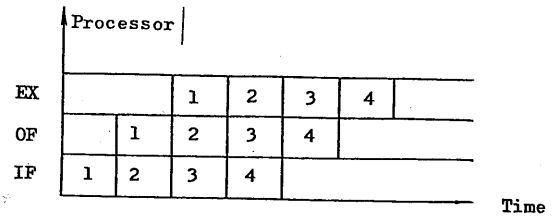


図7. 各プロセッサでの処理例

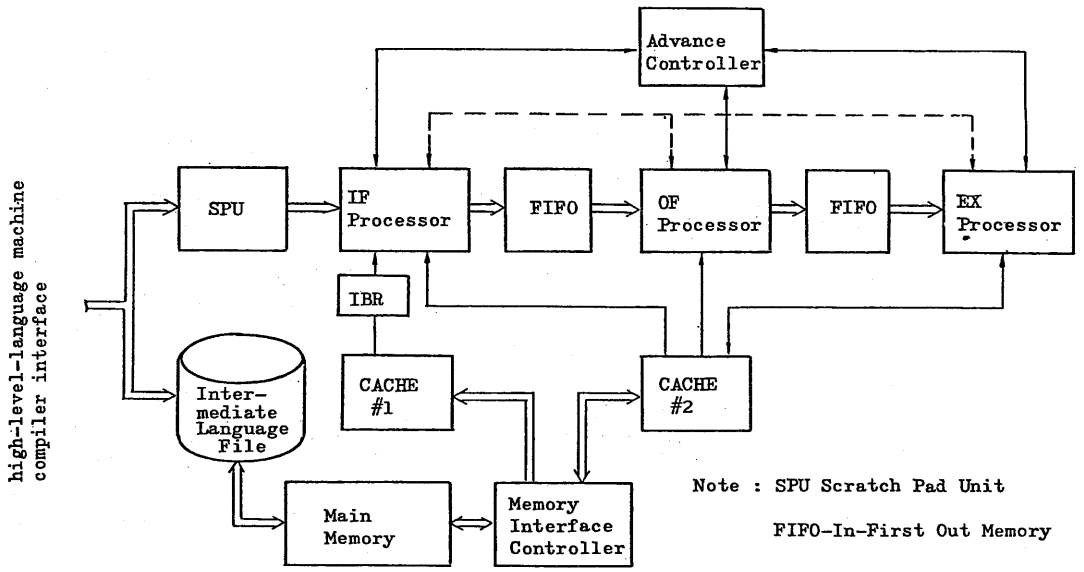


図6. COMBATマシン構成

どがある。

3.1 命令取出し(IF)プロセッサ

IFプロセッサは命令バッファレジスタ(IBR)を介して主記憶から命令コードとバリアントおよびそれらに続く複数個のオペランド・シラブルを取り出す。さらに主記憶からIBRの空き領域に命令列を先取りする。

IFプロセッサからOFプロセッサへの情報転送インタフェースとしてFIFOメモリが使われる。IFプロセッサはオペレーションコード，バリアントおよび命令中のオペランド数情報を命令単位でFIFOメモリに格納する。

IFプロセッサはオペランド・アドレス展開と命令中の各オペランドに対する内部ディスクリプタの生成を行なう。内部ディスクリプタはオペランド・シラブル中の情報とデータディスクリプタ中の情報から作成され，データタイプ，データ長，十進データの小数点位置表示，オペランドのアクセスタイプと論理アドレスから構成される。オペランドのアクセスタイプにはリード型，ストア型とリード・ストア型がある。これらのアクセスタイプはOFプロセッサでオペランドデータを読み出すかあるいは読み出しを行わないで内部ディスクリプタを直接EX部に転送するかを決定するために使われる。表1にオペランドのアクセスタイプの例を示す。

このオペランドのアクセスタイプ情報を利用することにより，OFプロセッサでは命令コードに係りなくオペランドに関して行うべき処理として，データ読出し処理あるいは内部ディスクリプタをEXプロセッサに直接転送する処理を行う。

オペランドが表の要素であり，指標づけ／添字づけされていると，この要素が表の範囲内にあるか否かを判定するための検査を行う(最大3次元)。さらにIFプロセッサでは，指標データあるいはデータディスクリプタ中に保持されている各次元でのサイズ情報を用いて要素の実効アドレス計算を行う。

IFプロセッサにおける他の重要な処理は次に実行する命令アドレスの決定である。

無条件分岐命令はIFプロセッサで行われ，次に先取りする命令を決定する。COBOLのSEARCHステートメントはある条件を満足する要素を表の中から見つけ出すために使われる。このためCOBOLのSEARCHステートメントに対応するCOMBAT命令列を構成しているCOMBATのSEARCH命令と，この命令に付随した条件分岐命令は通常，繰返し実行される。このことから，ある要素に関して最後の条件を調べる条件分岐命令の後，次に続く要素がSEARCHステートメント内の各条件を満足するか否かを調べるためにIFプロセッサでは再びCOMBATのSEARCH命令を先取りする。

SEARCHステートメントの機能は1つ以上のプロシージャに制御を移すことであるが，規定の繰返し処理が終了したことを検出すると，IFプロセッサは自動的に正規の命令シーケンスの処理に戻る。

COMBAT・PERFORM命令の処理後，IFプロセッサはこのPERFORM命令によって指定されるプロシージャの先取りを行う。このプロシージャ

表1. オペランドアクセスタイプの例

	OPERAND ACCESS TYPE		
INSTRUCTION	OPERAND A	OPERAND B	OPERAND C
ADD A TO B	READ	READ-STORE	
ADD A B GIVING C	READ	READ	STORE

の処理終了後，IFプロセッサはハードウェアスタックを用いて制御を正規の命令シーケンスに戻す。

IFプロセッサは8箇の4ビットスライスマイクロプロセッサとFIFOメモリと演算論理素子で構成される。FIFOメモリは内部ディスクリプタを8箇(オペランド)まで保持する。

マイクロプログラムをサポートするハードウェアとして設けられている。オペランド数生成部とオペランドのアクセスタイプ生成部にはPLAとROMが用いられている。

3.2 オペランド取出し(OF)プロセッサ

OFプロセッサはIFプロセッサからFIFOメモリを介して送られてきた内部ディスクリプタを参照して主記憶からオペランドデータを読出し，さらにデータタイプの変換を行う。

演算用のデータを読出すときには数字と符号の表現が正しいか否かおよびデータの内容がゼロであるか否かを検査する。このとき十進データ中の符号コードを除き，内部ディスクリプタ中の符号ビットで十進データの正負を表現する。

文字列を読出すときには32バイト以下の文字列が対象となり，読出した文字列の全てが空白あるいは英字だけから成っているか否かが検査される。

OFプロセッサにおけるデータタイプの変換は次のように行われる。

1) 演算／比較命令においては，全ての演算データは32ビットの補数表示二進数あるいは符号なしのパック十進数に変換される。
2) 移送処理で使われる送り側データは受け側データタイプに従って補数表示二進数，符号なしパック十進数，符号なしゾーン十進数あるいは文字列に変換される。
3) INSPECT命令におけるPOINTERデータとSTRING，UNSTRING命令におけるCOUNTデータは常に補数表示二進数に変換される。

COMBAT命令で利用できる多種類のデータタイプに上記データタイプの変換が施されることにより内部データタイプ数は減少する。OFプロセッサでのデータタイプ変換によって，EXプロセッサで扱うデータタイプは少数に限定できる。

OFプロセッサで変換されたデータはEXプロセッサのオペランドレジスタファイルに格納される。

上記の検査結果および新たなデータタイプはIFプロセッサから送られた内部ディスクリプタに付加されてEXプロセッサに送られる。

OFプロセッサの主な構成要素は8箇の4ビットスライスシーケンスコントローラとマイクロプログラムメモリ，32ビットALU，データ変換器，シフタと一時記憶用レジスタファイルである。データ変換タイプの決定は複雑なので，変換用マイクロプログラムルーチンに対するエントリアドレスをPLAを用いて生成する。

3.3 命令実行プロセッサ

FIFOメモリを介してOFプロセッサから送られてくる内部ディスクリプタに従い，EXプロセッサはオペランドレジスタファイルあるいは主記憶から処理データを読出す。

EXプロセッサは命令コードで指示された処理を行い，そこで得られた結果は受け側データに応じて，データタイプ変換，十進数データの小数点位置合せおよび四捨五入処理を施した後，指定された主記憶中の領域に格納する。また編集と

BLANK WHEN ZERO 処理も必要に応じて実行する。

移送処理はMOVE命令の他に，演算命令での結果の移送，データ操作及び表操作で頻繁に行われる。

OPプロセッサで受け側データタイプに応じた送り側データの変換が行われているので，EXプロセッサでの移送処理としては小数点位置合せ，符号挿入および空白あるいはゼロづめ操作を行えばよい。これらの移送処理用に専用のハードウェアを備える。このハードウェアにより小数データの小数点位置に合わせたデータシフト数，符号挿入バイト位置および文字づめ／文字落ちデータ長の決定を行う。

小数の加減算において，符号なし小数データは固定小数点位置の64桁高速ワークレジスタファイルに格納される。このレジスタファイルを使うことにより小数演算が小数点位置を考慮せずに実行される。有効桁数が小さいデータの演算を速く行うためにレジスタファイルの内容がゼロである語の演算は行わず，有効桁を含む語だけの演算を行う。この処理を行うために語の内容を検出する制御回路がレジスタファイル中に備えられている。

COBOLプログラムでは多様な編集操作が求められる。例えば，固定挿入，浮動挿入，ゼロ抑制および置換操作がある。この複雑な編集操作を効率よく行うためにマイクロオペレータ形式が採用されている。このオペレータに従って処理を行うため，編集制御情報生成用にPLAを組込んだハードウェアを採用している。

EXプロセッサで条件分岐命令を実行した結果，IPプロセッサが先取りした命令とは別の命令に分岐することが判明すると，IPプロセッサ，OPプロセッサに分岐アドレスと制御信号を送り，各プロセッサを初期化する。この情報を受けて，IPプロセッサはそれまでに先取りした命令を棄て，分岐アドレスで示された新たな命令の処理を行う。OPプロセッサは初期化された後，新たな命令がIPプロセッサから送られてくるまで待機する。

EXプロセッサでの処理は複雑なので，マイクロプログラム制御をサポートして十分な処理性能を保持するためにデータタイプ変換，小数データの小数点位置合せ，編集制御などに対するハードウェアを備える。その他，4個の4ビットスライスシーケンス制御部，32ビットの二進演算器，8桁の十進加算器と高速ワークレジスタファイルから構成されている。

4. 性能評価

4.1 COMBAT アーキテクチャの評価

マシンアーキテクチャと高級言語との近さを示す指標としてIPF（Instruction Per Function）が使われる。これはソース言語の1ステートメントが平均何個の機械語命令へ変換されるかを示す値である。

単純な（事務計算）処理と複雑な処理の2つの場合について，COMBATアーキテクチャのIPFを求めた。比較のためにCOBOL言語処理を効率よく行なう事を意図して設定されたといわれているNEAC ACOS77と，IBM370のIPFを求めた。

これらのIPFを表2に示す。この結果，COMBATがCOBOL言語にかなり近

表2. IPFの比較

	COMBAT	ACOS	IBM 370
Simple Business Application	1.03	1.87	3.50
Complex Data Processing	1.2	10.9	97.8

いことが示される。このことはCOMBAT用の高級言語トランスレータが簡単になることを意味する。

他にオブジェクトメモリ量は重要な評価項目である。前述の2つの場合のソーステートメントに対してCOMBATアーキテクチャではACOSよりも15%～50%オブジェクトメモリ量が少なくなっている。

COMBATにおいては，COBOLソーステートメントに対する適合性，多様な内部データの保持，ユーザ定義データとの直接的な対応と機械語命令中での強力な指標／添字づけを備えることによりIPFとオブジェクトメモリ量の改善を計ることができた。

4.2 ハードウェアの評価

パイプライン化した多重プロセッサシステムにおいては各プロセッサでの処理量が均衡しているか否かによって性能が大きく影響される。

簡単な事務処理で使われる基本的なCOBOLステートメントを実行するために必要なマイクロプログラムのステップ数を前述の各プロセッサについて求めた。この結果，各プロセッサの負荷がほぼ均衡していることが示された。IF，OF，EXの各プロセッサでの実行ステップ比は約1：1：0.8となる。この比率は指標づけ，十進数データの小数点位置合せ，および編集操作がない簡単な場合の値である。これに対して文字列操作，表操作および複雑なデータ属性や多様な補助操作（ROUNDED, SIZE ERROR, GIVINGなど）を伴う演算命令を実行する場合にはEXプロセッサでの処理は，より複雑になる。これらの場合にはIF，OFプロセッサでの実行時間はEXプロセッサより少なくなる。これを考慮したときには3プロセッサで要する実行時間はさらに均衡すると思われる。

COMBATマシンの性能評価として次の3つの異なる環境において3プロセッサを動作させるときのCOMBATマシン処理速度を求めた。

1) 各COMBAT命令は3つのプロセッサで直列に処理され，ある時間には常に1つのプロセッサだけが動いている場合である。COMBATマシンの実行速度は3プロセッサでの処理に要したステップ数の和となる。この形態では各プロセッサの処理の重なりを許さないので最悪の性能評価となる。

2) 各COMBAT命令は各プロセッサで並列処理され条件分岐命令は分岐不成功で常にブランチを伴わない場合である。各命令に対するCOMBATマシンの実行時間としては3プロセッサ中で最大の実行時間を必要としたプロセッサの実行時間に等しいとして近似する。この場合には3プロセッサが最良の並列処理を行う環境であり，評価結果としても最高の性能を示すことになる。

3) 各COMBAT命令は条件分岐命令において50%の成功率で実行されることを除き2)と同様の環境で実行される場合である。この場合には実際の処理性能に近い評価結果を得ることができる。

表3に簡単な事務処理で使用される基本COBOLステートメントに対するCOMBATマシン実行速度を示す。

表3. COMBATマシン実行速度

	execution speed in μ sec
Case (1)	9.3
Case (2)	4.0
Case (3)	4.6

1 Machine Cycle = 200 n sec

詳細な評価結果としてCOMBATマシンの主な性能改善理由を次に示す。

1) COBOL言語専用のCOMBAT命令
2) 送り側と受け側のデータタイプに応じたデータ形式の直接変換
3) ユーザ定義のデータに対して全データ属性を保持する内部ディスクリプタ
4) 高級言語での処理をサポートする特殊なハードウェア
5) 高速なマイクロプロセッサによるマルチプロセッサ構成

5. 結論

COMBATアーキテクチャ，高級言語マシン構成および現状における性能評価結果を述べた。

COMBATシステムはマシンアーキテクチャとハードウェア設計の両面から最適になるように考慮した高級言語処理システムである。COMBATアーキテクチャはCOBOLプログラムの実行用に高度に専用化されている。COMBATマシンは低価格で高性能なLSI／VLSI素子で実現されている。COMBATマシンにおいては同様のハードウェアレベルの汎用計算機アーキテクチャよりもCOMBATに変換される方がより効率的である。

複雑なCOMBATアーキテクチャは低価格で比較的遅いと考えられるマイクロプロセッサを構成要素とするCOMBATマシンで実現される。COMBATマシンにおける高い処理性能はマルチマイクロプロセッサ構成とマイクロプログラムに対する特殊なサポートハードウェアによって達成されている。

COMBATアーキテクチャは設計済みであり，詳細なハードウェアは現在設計中である。

謝辞

本研究開発は通商産業省　工業技術院の大型プロジェクト「パターン情報処理システムの研究開発」の一環として行われている。

本研究を進めるに当たり日頃適切な御指導をいただく当所コンピュータ・システム研究部，藤野部長，称津課長に深謝すると共にCOMBATアーキテクチャとハードウェアの設計を支援していただいている本プロジェクトの諸氏に深謝します。

文献

1. H. Weber, A microprogrammed implementation of EULER on IBM System 360 Model 30, commu. ACM10, 9, 1967, PP. 549-558
2. 山本他「COBOLマシン・アーキテクチャについて」第17回情報処理学会全国大会，1976　PP 307～308
3. 中崎他「COBOLマシン・ハードウェア構成について」第17回情報処理学会全国大会，1976　PP 309～310
4. American National Standard Programming Language COBOL, X3.23 1974, American National Standard Imstitute, Inc., 1974