汎用直接実行型計算機 UDEC のアーキテクチャ†

板野肯三†† 佐藤 豊†††

高水準の対話型プログラミング環境を実現する基礎として，手続き型言語のソースプログラムを直接実行する汎用直接実行型計算機 UDEC を設計し試作した．UDEC のハードウェアは，特定の言語や仕様に依存せず，制御アルゴリズムを入れ替えることで複数の言語に対応する．また，全体を機能別ユニットに分解してパイプライン結合し，各ユニット内の制御テーブルでハードウェアを直接制御することにより高速化を図った．直接実行のための制御アルゴリズムは，言語の構文と意味を組み合わせた実行文法の形で記述して，これをハードウェアで実行できる形式に変換して実現した．UDEC の性能を評価するために実験的に PASCAL のサブセットを実現し，シミュレータおよび実際のハードウェア上で実行して，動作を解析した．

1. まえがき

高水準言語のソースプログラムをハードウェアで直接実行する直接実行型計算機は，言語とアーキテクチャの間のセマンティックギャップを完全に解消するとともに，プログラムの視覚的イメージを実行時に保持することによって，理想的な対話的プログラミング環境の基礎を実現する．しかし，その実現上のコスト，実行速度，柔軟性などの点でコンパイラを用いた方式に比べて不利であり，非現実的であるとさえ考えられてきた[16]．ところが，最近のハードウェアの生産技術の進歩によって，コストの点での問題は既に解消されつつある．一方，比較的完成された分野である，コンパイラを基本とした言語処理系の設計技術を応用して，直接実行アリゴリズムをハードウェア向きに展開する試みが，著者らによってなされてきた[1)~3)]．

これらの結果，直接実行型計算機を現実的なコスト性能比で実現することは，十分可能であると判断した．そこで，直接実行型計算機の実現性の検証と性能の評価を目的として，複数の言語の実行に柔軟に対応できる汎用の直接実行型計算機 UDEC (Universal Direct-Execution Computer)[5)~9)] を試作した．そして，この上で PASCAL のサブセットの直接実行型計算機を実現して性能を解析した．本論文では，この試作の経験をもとに，そのアーキテクチャと性能評価の結果について報告する．以下，直接実行の意義と UDEC の設計の方針，UDEC の全体構成，各モジュールの構成を述べ，シミュレーションに基づいて性能評価と動作解析を行う．

† Architecture of a Universal Direct-Execution Computer UDEC by KOZO ITANO (Institute of Information Sciences and Electronics, University of Tsukuba) and YUTAKA SATO (Doctoral Program in Engineering, University of Tsukuba).

本研究は，文部省科学研究費補助金試験研究(1) 58850063 によって補助された．

†† 筑波大学電子情報工学系
††† 筑波大学工学研究科

2. 直接実行方式とその実現の方針

UDEC は，ブロック構造を持つ実用的な手続き型の言語を対象とし，一次元的なソースプログラムを直接，トークンごとに実行し，トークンごとの対話機能を提供する．その実現にあたっては，複数の言語に容易に対応できること，単純なハードウェアで実現できること，性能的には通常のコンパイルされた機械語に近い実行速度を目標とする．

2.1 UDEC における直接実行方式

直接実行方式では，従来のコンパイル方式では失われてしまうプログラムの文字列の視覚的イメージや，言語の構文および意味を実行時を利用することができ，これによって高度な対話的実行を実現する．

(1) 生のソースプログラムの直接実行

開発段階にあるプログラムに対しては，人間，編集，実行，保存など，非常に多くの要素が関与して，相互に連絡する．プログラムの表現を文字列としてのプログラムに統一することで，これらの間のインタフェースの互換性を保ち，人間にプログラムを文字列として表示したり，保存するためのコストを小さくすることができるので，UDEC はこの生のソースプログラム表現を機械語として直接実行する方式をとることにした．

(2) トークンを単位とする対話的な実行

UDEC は，プログラムを構成する名前や記号などの字句レベルのトークンごとに実行を行う．そして，1トークンの実行ごとに，内部状態の表示や変更などの対話を行う機能を実現する．このような，非常にきめこまかな対話機能によって，プログラムの実行に伴

う内部状態の変化が最小の単位で観察できるので，特にプログラムの誤りが小さな部分に局所化された後のデバッグなどに，理想的な追跡機能を実現する基礎となる．

（3） 言語の構文や意味に基づく構造的デバッグ

UDEC では，ソースプログラムの構文を実行時に常に認識しているので，任意の構文（式，文，手続きなど）を単位とする構造的な実行の制御や監視が行える．プログラムの誤りが局所化されていない時点でのデバッグなどにおいてプログラムの動作を大まかに観察するための機能は，このような言語の構文を単位とする実行の制御で実現される．同様に，言語の意味も常に認識しており，これを利用して言語の意味で規定される誤りを自動的に検出する．

2.2 実現の方針

UDEC の設計にあたっては，複数の言語に柔軟に対応できるだけでなく，ハードウェアの特性を生かして高速かつ単純に実現できること，構成方式を実験的に柔軟に再構成できることを考慮した．さらに，特定の言語向きに UDEC を設定することを容易にするために，構文の認識と同時に実行すべき意味を実行文法[7]で記述した．

（1） 複数の言語への柔軟な対応

複数の言語に柔軟に対応できるように，UDEC の各ユニットの制御の多くはテーブル化され，また，それが困難な意味の実行に関するものはマイクロプログラム化した．これにより，UDEC にはハードウェア・レベルで特定の言語に依存した部分は存在しない．

（2） ハードウェア向きの実現方式

UDEC の各種制御テーブルはその出力を直接的に制御信号としてハードウェア機構を制御することによって，ハードウェアの単純化と高速な制御機構を実現している．また独立に実行できる機能をモジュール化して並列に動作させることで高速化を図った．

（3） 再構成が可能な構成方式

UDEC の設計段階では，目的別に最適な複数の構成方式を検討した．この結果，異なる構成をとる場合でも，多くのモジュールは共通に利用できることが判明した．そこで，モジュール間の結合をメッセージ通信の形で実現することで，実験的に複数の構成方式を試行することを容易にした．

（4） 実行文法による設計

実行文法は，トークン単位の直接実行を表現するのに向く LL(1)文法をもとに，これをハードウェアによる直接実行系の実現や性能を直接的に反映するよう制限して記述した構文を骨組みとして，構文の解析と意味の認識や実行を一体化して記述できるよう拡張したものである．具体的には，生成規則の右辺が含む終端記号をたかだか1個，非終端記号をたかだか2個とし，終端記号は右辺の先頭にのみ現れるよう制限した．一方，終端記号として実行時の文脈を判定する記号を許した．そして各生成規則ごとに，その生成規則の展開時に実行すべき意味，字句解析や構文解析の機構の制御，エラー処理を記述する．

3. ハードウェア構成

UDEC ではソースプログラムの直接実行の過程を，字句解析，構文と意味の認識，制御構造とデータ構造の認識および実行に分離し，それぞれの処理を独立したユニットで実現した．プログラムの構造の解釈を高速化するために，制御構造やデータ構造を抽出してテーブルに保持して使用する．また，字句解析時にトークンをコード化して，ユニット間の通信やテーブルの検索を高速に行う機構を単純に実現した．

3.1 構成ユニット

図1 に UDEC のモジュールレベルでの構成を示す．UDEC は，パイプライン型に結合された，フェッチ部 (FU)，字句解析部 (LU)，名前検索部 (SU)，構文意味認識部 (SSR)，制御実行部 (CU) およびデータ処理部 (DU) を中心に構成される．実行するプログラムはプログラムメモリ (PM) に，データの実体はデータメモリ (DM) に格納される．また，字句テーブル (LT) は LU で認識された字句を記録し，三つのディスクリプタテーブル (DDT, PDT, CDT) は，データ，手続き，制御構造のそれぞれを記述するディスクリプタを格納する．

3.2 ユニット間のパイプライン結合

一つのトークンに対する実行は，字句解析，構文解析，意味の解析と実行などのフェーズで行われる．そこで，これらの処理を行う各ユニットを，互いに独立した自律的なユニットとして並列に動作できるようにし，これらをパイプライン型に結合して，高速な処理を実現する．ユニット間のデータの転送と同期は，同期機構を備えたリングバスの高速なメッセージ通信によって実現されている（**図 2**）．

実験的に全体の構成を柔軟に変更できるよう，ユニット間を固定的な専用線などで結合せず，これらを高速のリングバス[12]を介したメッセージ通信によって結

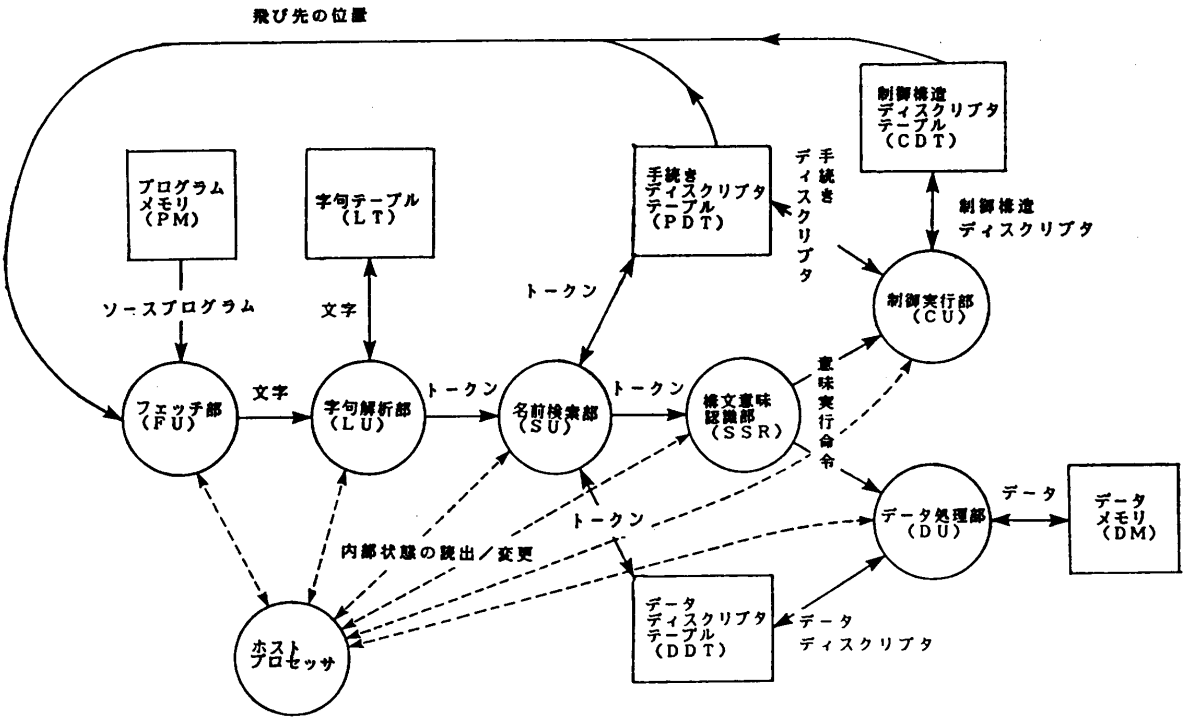

図 1 直接実行型計算機 UDEC の構成
Fig. 1 Organization of the direct-execution computer UDEC.

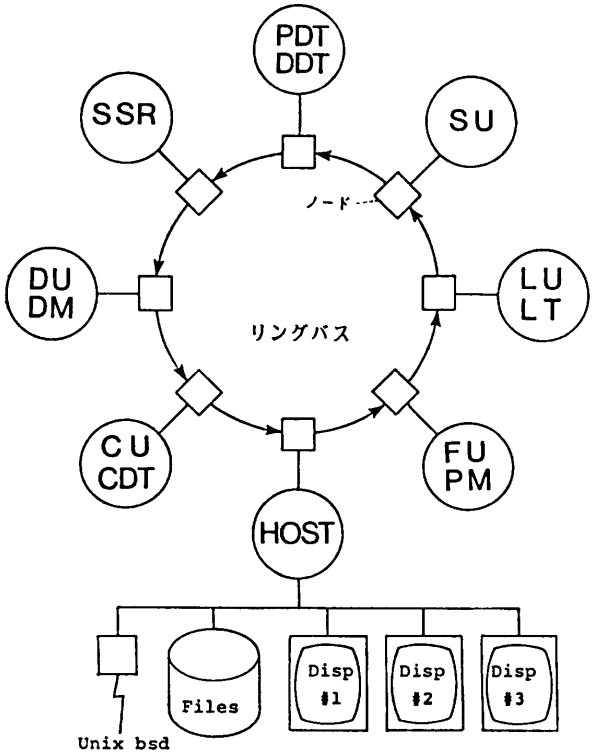

図 2 リングバスによるユニットの結合
Fig. 2 Interconnection of units using a ring bus.

合した．バスの転送速度（約 300 Mbps）は各ユニットの処理速度に比べて十分高速であり，さらにリングバス上でのこれらのユニットの位置を最適に割り当てることで，結合のためのオーバヘッドは解消される．

3.3 各ユニットの機能

（1） ディスクリプタテーブル(PDT, DDT, CDT)

処理の高速化のために，実行時に一度解析した名前の属性やプログラムの構造は，高速に連想アクセスできるディスクリプタテーブルに格納して利用する．手続きや変数などに対しては，その各種属性を保持するディスクリプタが生成され，その名前が属するブロックを識別するブロック識別子と，LU で作られる名前の一意コードの組をキーとする連想テーブル（PDT, DDT）に格納される．また，制御構造の実行に伴う不連続な制御の移動を高速化するために，それらの構造を抽出したディスクリプタが生成され，ブロック識別子とプログラム中の位置の組によって連想検索されるテーブル（CDT）に格納される．

各種ディスクリプタの形式を**図 3** に示す．各ディスクリプタは共通に，それが属するブロックのブロック識別子と，対応する名前の一意コードあるいは，対応する制御構造のプログラム中での位置をキーとして持つ．ディスクリプタの内容として，手続きディスクリ

手続きディスクリプタ

ブロック識別子	名前	タイプ	位置	励起レコードサイズ

データディスクリプタ

ブロック識別子	名前	タイプ	オフセット	サイズ

制御構造ディスクリプタ(IF)

ブロック識別子	位置	タイプ	ELSE位置	脱出位置

制御構造ディスクリプタ(WHILE)

ブロック識別子	位置	タイプ	条件式位置	脱出位置

図 3 ディスクリプタの形式
Fig. 3 Discriptor formats.

プタは手続きや関数に対して，その局所変数を格納するための励起レコードの大きさ，その本体のプログラム中での位置を格納する．データディスクリプタは仮引数や変数に対して，データ型，サイズおよび励起レコード中での相対位置を格納する．制御構造ディスクリプタは，例えばIF文に対してはELSEの位置と脱出位置というように，制御構造の骨組みを格納する．

（2） 字句解析部 (LU) と字句テーブル (LT)

LU (Lexical Unit) は，FU (Fetch Unit) から送られる文字の列を入力して字句レベルのトークンに分解し，可変長の文字列からなるトークンをコンパクトな固定長コードに変換し，そのタイプを付加して出力する．このコードは，トークンの文字列を一意に識別する固定長コードであり，タイプは名前，定数，予約語および特殊記号に分類される．このトークンのコード化とタイプ分けのために，トークンの文字列を LT (Lexical Table) に登録し，検索する．

FU はプログラムの読み出しを字句解析と並列に行うために分離されたユニットであり，機能的には LU の一部である．FU は，PM (Program Memory) に格納されているソースプログラムを，現在の実行位置から1文字ずつ読み出す．

（3） 名前検索部 (SU)

SU (Scope Unit) は，LU からのトークンを入力し，そのトークンの型が名前であるとき，PDT/DDT中をブロック構造の現在のレベルから検索してその名前のディスクリプタを求める．その名前が発見されたら，それが属すブロックのレベルとディスクリプタに従って，トークンのタイプをさらに詳細化するとともに，トークンのコードを対応するディスクリプタのテーブル中でのインデックスで置き換えて出力する．

（4） 構文意味認識部 (SSR)

SSR (Syntax & Semantics Recognizer) は，実行文法の形で記述された直接実行アルゴリズムを解釈しながら，直接実行系全体の動作を制御する．SSR は，SU から送られてくるトークンと，意味実行部から送られてくる実行結果を同一のレベルで受け取り，構文と意味を認識して，必要な意味の実行の指示を意味実行命令として意味実行部に送る．

SSR は，SU から送られたトークンのタイプに基づいて構文の認識だけでなく，字句レベルの意味，例えば，名前の宣言部での二重定義，実行部での未定義名の参照や手続き名への代入などの誤りを検出する．さらに，意味実行部から実際に実行されているプログラムの意味，例えば IF 文や WHILE 文の条件式の値を受け取って認識し，制御の流れが変更される場合には，構文解析の過程を切り換えるとともに，字句解析部の先行処理を取り消す．

（5） 制御実行部 (CU)

CU (Control Unit) は，手続きや各種の制御構造を記述するディスクリプタを生成するとともに，それらの実行において実行位置が不連続に移動する時に，新たな実行位置を生成して FU に知らせる．

CU は，現在実行中の手続きとその実行位置を指示するためのレジスタと，手続きの呼び出しと戻り，および入れ子状の制御構造の実行を管理するスタックを内部に持つ．そして，これらを制御するために，手続きや制御文の構造を記述するディスクリプタを生成して PDT や CDT に登録し，利用する．これらのディスクリプタは，プログラムを最初に実行する際の走査時に作成され，対応するテーブルに格納される．一方，格納されたディスクリプタは，SU で検索されたインデックスを利用してテーブルから直接取り出される．

（6） データ処理部 (DU)

DU (Data Unit) は，励起レコード中のデータへのアクセスを行い，それによって取り出されたデータに対する式の評価を行うユニットである．そのために，入れ子状に構成されたブロック構造に対するアドレシ

ング機構としてディスプレイレジスタとその管理機構[10]，および式の評価を行うためのスタックと演算機構を持つ．

プログラムが使用するデータの実体はすべて，DM (Data Memory)上にとられる励起レコードのスタック中に保持され，このスタックを管理するための情報をDU 内に持つ．これらの情報は，手続きの呼び出しや戻り時に更新維持される．データの実体は，その静的ブロックレベルやデータ型などの属性を記述するディスクリプタを含む，DDT を介してアクセスされる．こうしてとり出されたデータは，演算スタック上に置かれて評価されると同時に，その型に関する整合性が検査される．手続きや関数の実引数と仮引数の数や型の整合の検査も DU が行う．

3.4 ホストとのインタフェース

ホストプロセッサは，リングバスを介して，バス上に分散された各ユニットとの通信およびメモリ要素へのアクセスを個別に行う．

（1） メモリ要素の設定と読み出し

受動的な要素であるメモリは，UDEC 中の各処理ユニットとホストプロセッサから対等の関係でアクセスできる．これによってホストプロセッサは，プログラムやデータだけでなく，ディスクリプタテーブルの設定や読み出しも直接的に行うことができる．

（2） 各ユニットの設定と制御

能動的な要素であるユニットは，ユニット間での通信と同一のレベルで，ホストプロセッサからの割込みをそれぞれが独立に受けつけることができる．これら各ユニットの内部状態を実行開始時に設定したり，デバッグ時に読み出し/変更するために，ホストプロセッサは各ユニットと直接に交信を行う．

（3） 対話インタフェース

各ユニットは，言語のいろいろなレベルの処理を受け持っているので，ホストプロセッサの交信の相手を選択することで，容易に対話的実行の単位を切り換えることができる．例えば交信相手として LU を選べば文字またはトークン単位で，SSR を選べば生成規則単位で交信が行える．実際の対話のためのコマンドの解釈や画面への表示などの機能は，言語への依存性が低く，高速性も要求されないので，これらの機能はホストプロセッサ上に実現される[13),14)]．

4. プロトタイプの実現

字句解析部および構文意味認識部はテーブル駆動型の専用ハードウェアで実現し，その他の意味実行を行うモジュールは，その実行アルゴリズムが言語によって大幅に異なるので，効率的に文字列処理などが行えるように設計して実現したマイクロプログラム可能なプロセッサ MEGA3[11)] 上に実現した．

4.1 対象言語

評価のための実現の対象に選んだ言語は，PASCAL のサブセット[4)] にブロック構造を追加した単純な言語 (TINY-PASCAL[3)]) であるが，評価用に必要であると思われる最低限の機能は備えている．制御構造としては，IF 文，WHILE 文，手続きおよび関数を含んでいる．一方，データ構造は単純変数と一次元の配列を含み，データ型は整数型のみである．これらに加え，スコープの制御方式を評価するために，入れ子状のブロック構造を許した．

4.2 各モジュールの実現

（1） 字句解析部と字句テーブル

字句解析部 (LU) は，文字クラスの生成部，トークンの境界認識部，予約語検査部の各サブユニットをパイプライン型に結合して構成した[9)]．文字クラス生成部は入力された文字をクラスに分類するテーブルとして構成され，トークンの境界認識部はテーブル化された言語の字句構文で制御される有限オートマトンとして実現した．予約語検査部は逐次ハッシュ法[8)] を用いた字句テーブルの登録，検索機構である．TINY-PASCAL の字句構文を認識するためのオートマトンは 9 状態を持ち，入力文字は 13 種類に分類されるので，このための制御テーブルは 117 エントリで構成される．

（2） 構文意味認識部

構文意味認識部 (SSR) の内部は，トークンの入力部，生成規則の選択部，認識結果の実行部で構成され，それぞれ固有のテーブルで制御される．これら制御テーブルには実行文法で記述された対象言語の実行アリゴリズムが，分割して格納される．TINY-PASCAL 用に設計した実行文法では，非終端記号は 88 個，生成規則の総数は 205 個であった．

（3） 意味実行部

意味実行部 (CU, DU) は，マイクロプログラム可能な 16 ビットのマイクロプロセッサ MEGA3[11)] 上に実現した．MEGA3 は，64 ビットの水平型マイクロプログラムで制御され，200 ns の基本クロックで動作する．制御記憶は 4 K語実装され，このうち 2 K語は書き換え可能である．制御アルゴリズムは各ユニット

ごとに，マイクロプログラム化された意味実行命令のデコード部と各命令の実行部で構成される．TINY-PASCAL では，意味実行部の各ユニットの実現に要したマイクロプログラムの規模は，CU が612ステップ，DU が833ステップであった．

（4） ディスクリプタテーブル

連想検索機能を持つ各ディスクリプタテーブルは，ハッシュテーブルとして構成し，ハードウェアによるハッシュ機構[6]で検索する．この機構のハッシュ方式はオープンハッシュ法で，キーは固定長である．ハッシュ関数は，ブロック識別子と名前コードあるいは位置の排他的論理和としている．

5. 性能と評価

ハードウェアとして実現された字句解析部と構文意味認識部，および MEGA 3 上のマイクロプログラムとして実現された意味実行部を，シミュレータと実マシン上で実行して性能を測定し動作を解析した．

5.1 各モジュールの性能

（1） 字句解析部の性能

字句解析部は，1文字を約 0.7 μs で処理する．トークンの平均文字数は，プログラムによって差異があるが，UNIX 上の多くのプログラムを分析した結果，3文字程度であった[9]．これに従うと，1トークン当たりの処理時間は，平均して約 2 μs 程度である．

（2） 構文意味認識部の性能

構文意味認識部は，一つのトークンを認識するのに平均して1.2回の生成規則の展開を行い，1回の展開に3クロックを要する．したがって，250 ns のクロックを用いている現在の実現では，約 0.9 μs 程度で1トークンを認識する[7]．

（3） 意味実行部の性能

MEGA 3 上のマイクロプログラムとして実現された意味実行部のアルゴリズムは，トークン当たりのマイクロプログラムの実行ステップが約60ステップである．MEGA3 は 200 ns のクロックで動作するので，このために約 12 μs 必要である[17]．

5.2 総合的な性能

パイプラインを構成する各ユニットの中で，ネックになるのは明らかに意味実行部である．したがって，UDEC 全体の1トークン当たりの実行時間は，ほぼ意味実行部での実行時間になる．

この UDEC の性能を客観的に評価するために，幾つかのサンプルプログラムを，UDEC 上で実行した場合と，市販の代表的なプロセッサ (VAX11/750, i8086, MC 68010) 上にコンパイルされた機械語で実行した場合について，その実行時間を測定した．表1に，UDEC でのトークン当たりの実行時間を示し，比較のために，コンパイルされたコードでの実行時間もトークン当たりに換算して示した．このように現在の UDEC は，比較されたプロセッサに対して 1/5～1/7 程度の性能を持つ．

しかし，測定データの詳細な解析の結果，意味実行部での実行時間の7割以上は，単純なスタック操作とアドレスの計算のために費やされていることが分かった．これらはハードウェア化によってほぼ1クロックで実行できる．この改良を行うことによって，現在の性能の3倍以上の性能が得られることが分かった[17]．

6. むすび

対話的なプログラミングを，アーキテクチャによって直接的に支援することを目的として，汎用の直接実行型計算機 UDEC を試作した．そして，この UDEC 上で実際に PASCAL のサブセットの直接実行アル

表1 トークン当たりの実行時間の比較
Table 1 Comparison of execution times per token.

単位：μs/token

			ackerman 関数	整数配列のソート	整数行列の乗算
UDEC			11.2	10.2	12.6
コンパイルされたコードの実行	VAX 11/750 (UNIX 4.2 BSD)	compiler	1.9	1.4	3.2
		interpreter	42.1	33.9	45.9
	i 8086	compiler (TURBO-PASCAL)	1.9	1.4	2.7
	MC 68010 (UNIX 4.2 BSD)	compiler	1.0	0.7	2.5
		interpreter	28.1	29.5	50.8

ゴリズムを試作して実行して性能を解析した．その結果，このようなハードウェアによる直接実行により，その速度を，従来のコンパイルされたコードの実行速度に近づけることができた．一方，言語の定義や変更に対する柔軟性は，高水準言語が一般的に含んでいる概念を抽出してそのような要素だけをハードウェア化し，言語に依存する部分をテーブル化することで得られた[5),6),15)]．

参考文献

1) Chu, Y.: *High-Level Language Computer Architecture*, Academic Press, New York (1975).
2) Chu, Y. and Itano, K. et al.: Interactive Direct-Execution Programming and Testing, *Proc. of COMPSAC '82* (1982).
3) Itano, K.: PASDEC: A Pascal Interactive Direct-Execution Computer, *Proc. of High-Level Language Computer Architecture Conference*, pp. 152-169 (1982).
4) Aho, A. and Ullman, J.: *Principles of Compiler Design*, Addison-Wesley, Reading (1979).
5) 板野肯三ほか：UDEC：汎用直接実行型計算機，第30回情報処理学会全国大会予稿集，pp. 223-230 (1985).
6) 板野肯三ほか：UDEC：汎用直接実行型計算機の設計と実現，第31回情報処理学会全国大会予稿集，pp. 197-206 (1985).
7) 板野肯三，佐藤 豊，林 謙治：直接実行系の構文意味認識機構の設計と実現，テクニカルノート，HLLA-115，筑波大学電子情報工学系(1985).
8) 板野肯三，佐々木日出美，山形朝義：連想記憶に基づくパイプライン型文字列検索アルゴリズム，情報処理学会論文誌，Vol. 26, No. 6, pp. 1152-1155 (1985).
9) 板野肯三，佐藤 豊，山形朝義：パイプライン型字句解析プロセッサの設計と実現，テクニカルノート，HLLA-114，筑波大学電子情報工学系 (1985).
10) 板野肯三，佐藤 豊：クロススタックキャッシュを用いたブロック構造言語のためのアドレッシング機構，情報処理学会論文誌，Vol. 27, No. 9 (掲載予定).
11) 板野肯三，杉原敏昭：マイクロプログラマブルプロセッサ MEGA3 の設計，テクニカルノート，HLLA-41，筑波大学電子情報工学系 (1984).
12) 板野肯三，日置伸二：階層型リングバス HIRB のアーキテクチャ，情報処理学会論文誌，Vol. 27, No. 1, pp. 72-80 (1986).
13) 佐藤 豊，板野肯三：動的複合実行方式―直接実行系と翻訳実行系を統合した対話型実行方式，ソフトウェア科学会コンピュータソフトウェア，Vol. 2, No. 4, pp. 19-29 (1985).
14) 佐藤 豊，板野肯三：COSMOS：対話型統合的プログラミングシステム，情報処理学会コンピュータシステム・シンポジウム，pp. 115-124 (1985).
15) 佐藤 豊，板野肯三：構造エディタとソースコード・インタプリタの統括的設計と実現法，情報処理学会プログラミング言語研究会，2-1, pp. 1-8 (1985).
16) Myers, G.J.: *Advances in Computer Architecture*, John-Wiley & Sons, New York (1978).
17) 板野肯三，佐藤 豊：マイクロプログラムで実現された直接実行型計算機のシミュレーションと評価，テクニカルノート，HLLA-139，筑波大学電子情報工学系 (1984).

(昭和61年1月9日受付)
(昭和61年5月15日採録)

板野 肯三（正会員）

昭和23年生．昭和46年東京大学理学部物理学科卒業．昭和48年同大学大学院修士課程修了．昭和51年同博士課程単位取得後退学．理学博士．筑波大学計算機センタ技官，同大学電子情報工学系助手を経て，現在，同講師．コンピュータアーキテクチャ，オペレーティングシステム，プログラミング言語に興味を持つ．ソフトウェア科学会，IEEE, ACM 各会員．

佐藤 豊（正会員）

昭和35年生．昭和57年筑波大学第三学群情報学類卒業．昭和59年同大学院修士課程理工学研究科修了．現在同大学院博士課程工学研究科に在学中．プログラミング・システムのユーザ・インタフェースおよび構成法の研究に従事．ソフトウェア科学会会員．