15. 拡張可能な言語のファームウェアによる処理について

京都大学 工学部　柴山　潔・新実　治男
富田　真治・萩原　宏

1. はじめに

計算機を使用して問題を解く場合，その計算機システムの末端に位置するユーザは，プログラミング言語を介して計算機と接する。そして，使用している計算機アーキテクチャに関する知識をもたない，又もつ必要のないユーザが，各自の抱える問題の処理手順を容易にプログラムできる様に，汎用の高級言語（高水準言語）が用意されている。この点で，高級プログラミング言語はユーザの立場に立って設計され，計算機アーキテクチャとは独立に存在するものである。

だから，「計算機システムをユーザ個別の問題に適応させる」ということは，ユーザと計算機との間に存在するギャップを何らかの方法で埋めることであり，普通これはコンパイラなどのソフトウェアを使用する方法で解決する。しかしこの場合，ユーザが抱える問題を解決する効率は，問題そのものとは直接に関係のないコンパイラなどの能力に大きく依存する。そこで，対象とする応用に特有なデータ構造を論理的に簡便な操作で取り扱うことのできる「問題向き高級言語」を利用することで，ユーザは各自の問題の本質部分の解決に専念できる様にする。そしてこの場合，合わせてその問題向き高級言語を効率良く実行できるアーキテクチャをもった専用計算機を実現することが有効な手段である。LISPマシン，PASCALマシンなどの試みがこのアプローチに属する。高レベルの問題向き言語と汎用計算機ハードウェアとのギャップが大きいために生じる処理効率の低下を，専用化されたハードウェアの採用で防ごうというものである。

しかし実際には，言語を固定することで生じる柔軟性の欠如が大きく，計算機システムとして適応する問題は限られてしまう。なぜなら，問題そのものも多様性をもつことが普通であるため，専用計算機といえども何らかの形で柔軟な構造をもち，問題に応じた拡張性を備えることが必要であるからである。要求されているものは，計算機の適度な柔軟性であり，これは問題適応化のレベルをソフトウェアからハードウェアへどの程度近づけ得るかによって定まる。そして，問題適応型可変構造計算機アーキテクチャの実現手段として，最近はファームウェア技術が脚光を浴びている。

特に，書き換え可能な制御記憶を用いたダイナミックマイクロプログラミングは，目的に応じてファームウェアを切り換えることができるので，ハードウェアの高速性を失わずに従来のソフトウェアがカバーしていた幅広い問題に対して柔軟に適応可能な計算機を構成する技術として注目されている。我々が開発した[1]計算機：QA-1は，このダイナミックマイクロプログラミングを特長の一つとする多目的計算機であり，高級言語処理においてそのファームウェアを活用したシステム開発を行なってきた。そして，計算機アーキテクチャを熟知したユーザは別として，そうでない末端のユーザが計算機の問題適応能力を充分に使いこなせ

ないという点に，システムの問題適応化における改善の余地があることがわかった。

小さな問題解決法を積み上げて，より大きな問題解決に挑む方法は，ソフトウェア・エンジニアリングの主要な手段として認められている。この段階的な言語の問題適応化とともに，計算機も又問題適応化できることが望ましい。すなわち，汎用とされる高級言語をファームウェアで処理する形の高級言語計算機においても，そのプログラミング言語が拡張性をもたないならば，ファームウェアのもつダイナミック性が有効に機能しているとは言い難い。逆に，拡張可能な言語でも，それをソフトウェアのみで処理しているのでは，計算機ハードウェアに近いレベルでの問題適応化ではないために実行効率が悪く，システムが問題に適応したとは言い難い。そこで我々は，レジスタ・レベルの並列処理とダイナミック・マイクロプログラミングを特長とするQA-1を中心として，問題に近いレベルで適応化を図ることのできる拡張可能な高級言語と，計算機に近いレベルで適応化を図ることのできるファームウェア技術とを融合して，ユーザ個別の問題にダイナミックに適応化することを目標としたシステム：FELPS (Firmware Extended Language Processing System) を考えた。(図1.1参照)

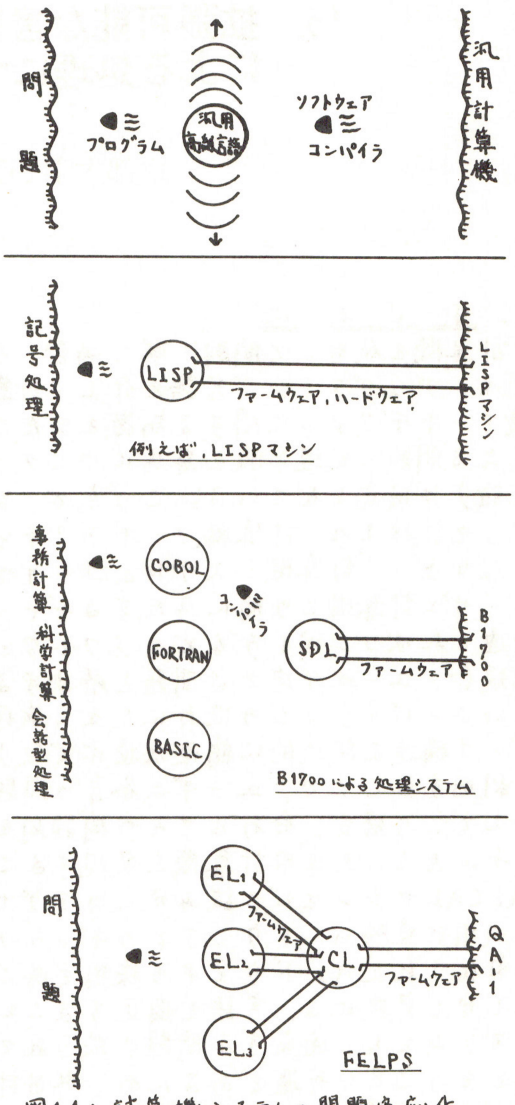

図1.1：計算機システムの問題適応化

2. システムの概要

(1) 高級言語の小さなセット (Core Language：CL) の処理機能とその拡張機能をファームウェアで用意し，マン＝マシンの会話によって，各ユーザ個別の問題に適応する言語 (Extended Language：EL) へのダイナミックな拡張を図る。

(2) CLはアルゴリズム記述用の高級言語であり，さらに拡張機能，会話機能を使うためのコマンドの組を含んでいる。ユーザは，コマンドを利用して，会話的にプログラムの作成を進める。

(3) ユーザがCLを使用して定義した「手続き」は，マイクロプログラムによって解釈実行される。手続きのデバッグはマン＝マシンの会話によってなされ，完成された手続きは，最適化ルーチンを備えたコンパイラによってマイクロプログラム化されて，言語体系にELとして組み込まれる。これをCLの拡張と呼び，拡張されたユーザ個別の言語体系をELと呼ぶ。ELを使うと，プログラム中に

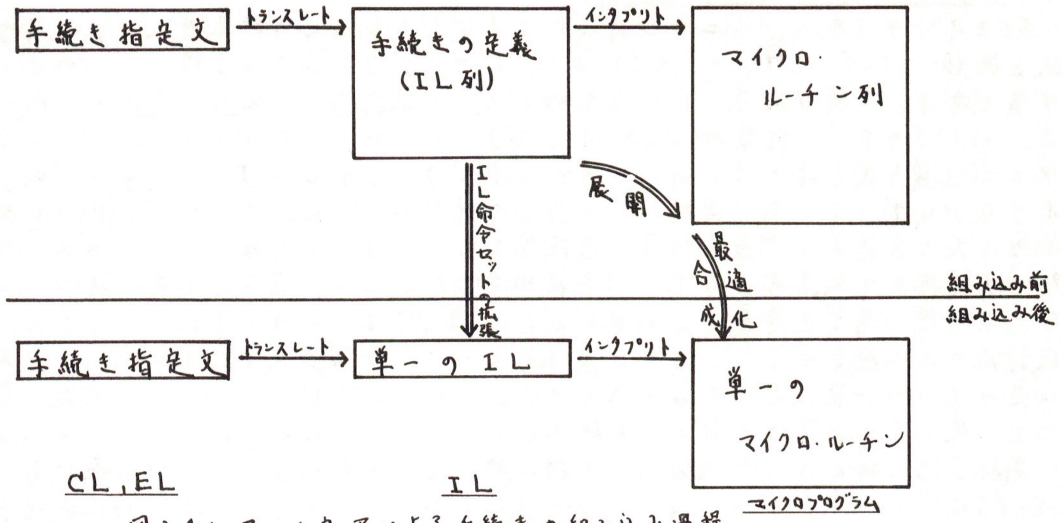

図2.1：ファームウェアによる手続きの組み込み過程

「手続き指定文」を書けばその手続きが直接にファームウェアで実行されるので，手続きの定義をCLに展開してそれを実行する通常のマクロ展開とは異なり，システムをユーザ個別の問題向きに適応させたことになる。(図2.1参照)

(4) FELPSは，図2.2に示す様に，QA-1と複数のミニコンとから構成されている計算機複合体上に実現され，機能分担による有効な分散処理を行なう。マンマシンの会話機能，入出力機能，ファイル管理の実行などの外部I/Oを直接に使用する処理はミニコン側で分担し，QA-1はファームウェア高級言語計算機として，ミニコンによって字句解析などの簡単な処理を経た後の内部記号列(Intermediate Language：IL)を解釈実行する。

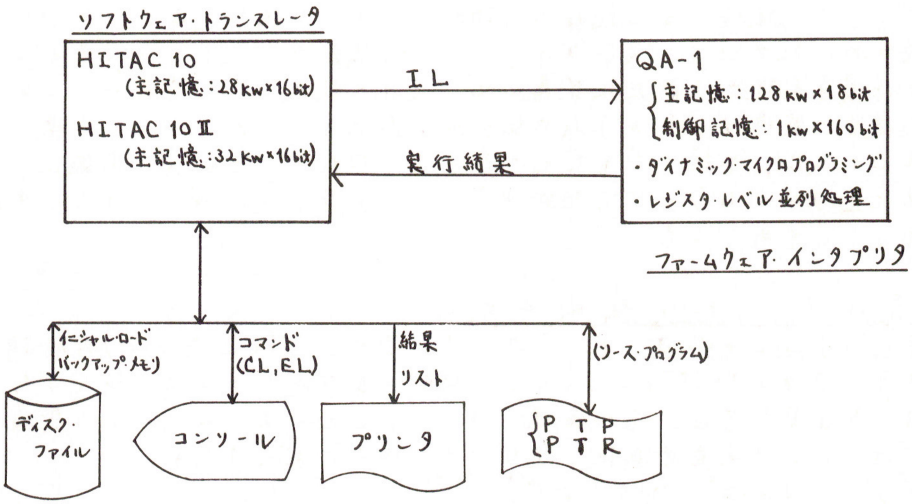

図2.2：FELPSのハードウェア構成

3. 核言語（CL）の選定

FELPSの対象は，ミニコンあるいはメガミニレベルの計算機上で，汎用の高級言語（FORTRAN，PL/I，APL，BASICなど）をプログラミングの手段としているユーザ層である。言語の拡張方式には大別して，(i) shell方式；(ii) core方式の2種がある。FELPSのホスト計算機はQA-1であり，そのハードウェア及びファームウェア上でshell方式言語のフルセットのインタプリタをインプリメントするのは，マイクロプログラミングの困難さからみても効率的でない。この点で，PL/Iなど(i)の方式の言語をまず選定対象から除外した。言語の小さなセットからユーザ個別の言語体系に拡張する方式：(ii)を採用するためには，拡張方法を会話的にして，かつその使い易さを考慮する必要がある。FELPSは，その対象ユーザをシステム設計者のみに限定せずに，できるだけ多くのユーザにシステムを解放し，システムそのものの一般性を高める試みである。このためCLにはコマンド機能を含ませて，使い易い拡張性を備えた言語体系を基本としている会話型システムとした。

高級言語が備えるべき機能として機械独立性，すなわち言語の移植性がある。FELPSは，ハード/ファーム/ソフトウェアのトレードオフを総合的に検討する試みであり，ハードウェアからソフトウェアまでを一体として設計対象としたトータルシステムである。その究極の目的はシステムそのもののパーソナル化であり，CLはファームウェア高級言語計算機となるべきホスト計算機上にできる限り簡単にインプリメントできることが望ましい。そこでCLは，既存の高級言語を参考にして，コンパクトな言語体系として新たに設計した。ソフトウェアエンジニアリングの面からみると，ユーザ個々のプログラム作成過程の簡略化を重要視したことになり，ユーザ間でのプログラム互換性の問題は，共用するプログラムをCLで書くことによって解決している。

言語の拡張レベルは種々あり，その拡張性によってCLの能力も大きく変り得る。FELPSのCLで拡張できる要素としては，データ型，演算子，手続きを考えた。CLはアルゴリズム記述用高級言語であり，それを用いて書くプログラムは，ユーザ個別のデータ構造を操作する手続きである。この場合，ユーザがデータ型をダイナミックに拡張できる能力の有効性が，データ抽象化などのアプローチにおいて確かめられている。又，ダイナミックに拡張されたデータ型をもったデータに対する演算手続きを新たに定義して，それを拡張された演算子としてELに組み込む機能の装備が，システムの拡張性を高める。さらに，データ型と演算子の拡張性をもとに，手続きをもCLからEL，ELからELへの拡張要素として考え，それをファームウェアで処理することにより，システムそのものを問題適応化させることをめざした。

4. 核言語（CL）の拡張手順

システムのイニシャルロード：ユーザ個別のEL（CLも含む）が格納されているシステムファイル（ディスク）から，QA-1及びミニコン上へシステムのイニシャルロードを実行する。その後は，コマンドを用いてシステムとの会話を逐次進めながら，ELの変更や拡張，プログラムの実行をくり返す。

プログラムとコマンド（図4.1参照）：〈プログラム〉はコンソールから入力される〈コマンド〉の系列である。ファームウェアで書かれている標準手続き（簡単な入出力処理，初等関数計算など）か，あるいはユーザが個別に定義した手

続きを順次〈即時実行コマンド〉によって実行させることが，従来のプログラム実行にあたる。〈即時実行コマンド〉には，手続きを実行する〈手続き指定コマンド〉と，演算子を使用した式を実行する〈式コマンド〉とがある。これらのコマンドで必要とする実パラメータやオペランドにはリテラル定数のみが許され，そのコマンドは単独でその機能を実行し終える。〈拡張定義コマンド〉は，データ型，演算子，手続きを新たに定義する際の初期入力用コマンドであり，ユーザは実行前にこのコマンドによって各々を定義しておく必要がある。〈デバッグコマンド〉は，〈拡張定義コマンド〉によってユーザが新たに定義したデータ型，演算子，手続きを修正したり，その実行トレースを行なったり，完成したこれらをユーザ個別のCLに組み込んだりする場合に使用する。又，〈編集コマンド〉

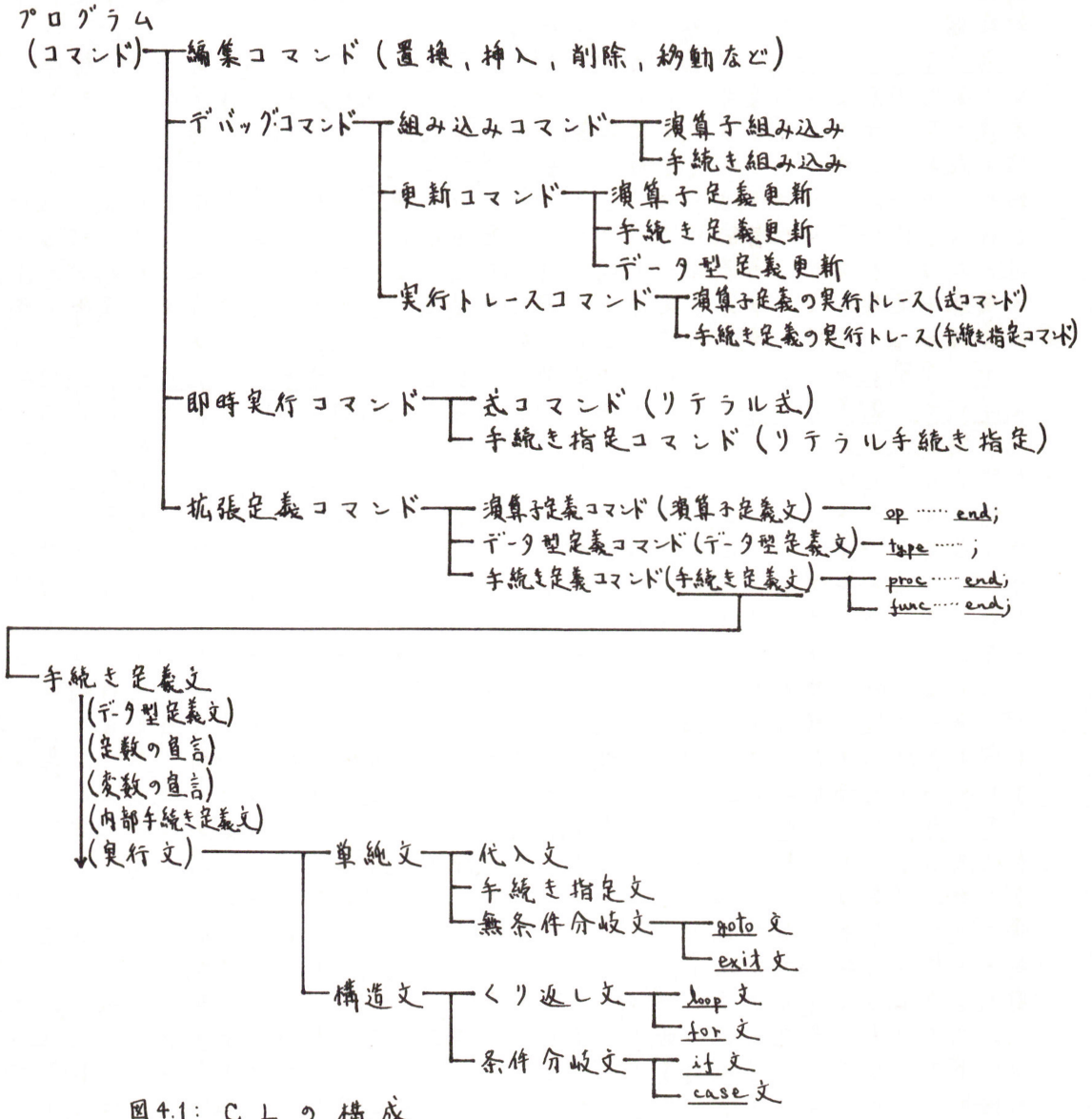

図4.1：CLの構成

は，プログラムエディタの機能をもつもので，ユーザが定義したりースプログラムをコンソールを介してシステムと会話しながら編集するために用意されている。

手続きとその定義：手続きは，〈手続き定義コマンド〉として，〈データ型定義文〉，〈定数の宣言〉，〈変数の宣言〉，〈内部手続き定義文〉，〈実行文〉を書き下すことによって定義される。定義を構成する各文に対しては，この定義入力時にトランスレータによってＩＬ列への変換が行なわれる。そして，〈即時実行コマンド〉として入力する〈手続き指定コマンド〉や，手続きの実行中に現われた〈手続き指定文〉の実行時にそれらＩＬ列が解釈実行される。手続きは，パラメータを使用して手続きの外部とデータの受け渡しをするほか，手続き名が値をもつ関数手続きも許される。又，手続きの定義中には１レベルまで〈内部手続き定義文〉を含むことが許される。手続き定義で宣言するデータ型，定数，変数の有効範囲は，その定義中のみに限定される。

演算子とその定義：ＣＬの標準データ型をもつデータ間において，標準の算術・論理演算が標準演算子によって指定できる。〈文〉の一部となる〈式〉は，変数の値へアクセスするためと演算子を作用させて新しい値を作るための計算手順を表わし，オペランド（変数，定数，関数手続き指定）および演算子を組み合わせて作る。式の評価は左から右へとなされ，演算子間の優先順位は設けない。これは，拡張可能な言語においては，演算子順位の設定が逆にユーザに対する負担となり，システムの柔軟性を妨げる要因ともなることを考慮したためである。〈演算子定義コマンド〉を使用する演算子定義は，定義する演算子用記号を指定するほかは，関数手続きの定義と同じ形式で行なわれる。

データ型とその定義：ＣＬの標準データ型には，整数型，実数型，論理値型，文字列型，ポインタ型が用意されている。そして，新たなデータ型の定義は〈データ型定義文〉によって，これら既定義のデータ型を組み合わせて構造的なデータ型を作成することで行なう。〈構造データ型〉の作成方法には，(1)異なるデータ型をもつデータをフィールドと呼ぶランダムな要素として許し，要素へのアクセスはそのフィールド名によって行なう〈レコード型〉；(2)１個のデータ型のみのデータをインデックス付けした要素として並べ，その要素へのアクセスは計算されたインデックスによって行なう〈配列型〉；の２種がある。又，ファームウェアインタプリタの特長を生かしたダイナミックなデータ空間管理を目標とすることから，配列型とレコード型とが混在する様な複雑なデータ型，実行時にインデックスの範囲が決まる動的配列などの使用を許した。データ型の定義は，それを含む手続き定義内で有効であり，この手続き定義時に行なわれる手続き定義のＩＬ列への変換処理過程において，データ型定義文のもつ情報が使用される。

ＣＬからＥＬへの拡張：〈拡張定義コマンド〉によって定義された演算子や手続きは，〈実行トレースコマンド〉や〈即時実行コマンド〉を利用してデバッグされ，必要ならば〈組み込みコマンド〉によってＥＬに拡張される。又，〈手続き定義文〉中に現われる〈データ型定義文〉によって定義されたデータ型は，その手続きのみに有効なものであるが，新しく定義したデータ型をＥＬとして共用しようとする場合には，〈データ型定義コマンド〉によって拡張定義すればよい。このため，演算子や手続きのＥＬへの組み込みの際に使用する〈組み込みコマンド〉にはデータ型に対するものはなく，この〈データ型定義コマンド〉がその機能を実行する。ユーザ個別のＥＬは，もともとＣＬから拡張されたものであ

り，その拡張されたＥＬを用いてさらに新たなＥＬを構成するブートストラップ法によって言語の問題適応化を図る。そして，この時に生じるオーバヘッドをできるだけ小さく押える為にファームウェアを利用して，システム全体の問題適応化をめざす。このための手法には，(1)ＣＬやＥＬを使用して演算子や手続きを定義しているソースプログラムから，トップダウンに最適化されたマイクロプログラムオブジェクトを生成するマイクロプログラムコンパイラを使用する；(2)演算子や手続きの定義時に変換されているＩＬ列を一旦マイクロプログラムに展開して，それからボトムアップに冗長部分を削除したり，並列処理可能な部分を発見したりして，単一のＩＬに対応するマイクロルーチンに再合成する；の２種のアプローチがある。FELPSでは，その設計思想を良く反映できる(2)の手法を採用する予定であるが，いずれの方法にしてもこの処理においては多少時間を要しても，最終的に実行効率の良いマイクロプログラムが生成できれば良い。このことから，QA-1のアーキテクチャを生かしたマイクロプログラムの最適化戦略の開発は別途に行なっている。

5. 機能分担によるシステムの実現

QA-1上に実現する高級言語計算機は，高級言語の簡単な変換過程（CL，EL→ＩＬ）を含んだ直接実行型である。そして，ＣＬやＥＬをインタプリタ(QA-1)への入力となるＩＬへ変換するトランスレータは，豊富なI/O機能を備えたミニコン側が分担する。（図5.1参照）

(1) ＩＬ列の作成（トランスレータ）：コンソールから入力されるソースプログラムを処理して，対応するＩＬ列に変換する。すなわち，空白，コメントなどの，以後の処理に不要な区切り記号を除去して，識別名，予約語，演算子や残された

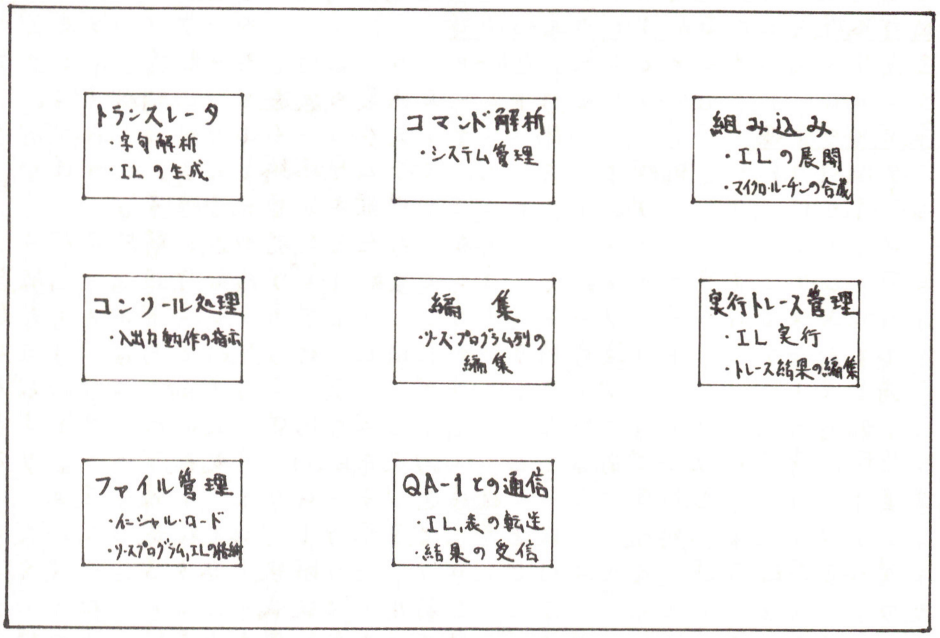

図5.1：トランスレータの機能

区切り記号（カッコ，コンマ，ピリオドなど），リテラルを抽出し，各々対応するＩＬに変換する。Ⓐ識別名に関しては，手続き名，データ型名，定数名，オペランド名，仮パラメータ名，変数名，ラベルを識別する処理を実行してＩＬオペレーションコードを生成し，識別名表へのポインタをＩＬオペランドとする；Ⓑ予約語に関しては，各予約語の意味解釈ルーチンへのリンクコードをそのＩＬオペレーションコードとし，必要ならば簡単な構文解析を同時に実行して，その際に生成された情報（例えば，条件文における条件成立時の分岐先アドレスなど）をＩＬオペランドとする；Ⓒ演算子と区切り記号に関しては，対応するＩＬオペレーションコードに変換するだけである；Ⓓリテラルに関しては，そのデータ型を解析してＩＬオペレーションコードとし，文字形式のデータを内部表現の数値データに変換して，それをそのままＩＬオペランドとする；以上の処理は主に，定義文中の実行文に対するトランスレータの機能である。変換後のＩＬは元のＣＬやＥＬとの対応が簡単にとれる高レベル言語であり，ＱＡ-１のファームウェアインタプリタはこのＩＬを直接に解釈実行する。これに対して，同じ定義文中に現われるが，その解釈を定義入力時に完了させてしまうことのできる，定数や変数の宣言やデータ型の定義などの部分の処理は，トランスレータによってほとんど実行され，ＣＬやＥＬは実行時にインタプリタに渡される識別名表などの一部に変換されるため，インタプリタによって解釈実行されるＩＬとしては陽に表現されない。

(2) コマンドの解析とその実行管理：ＣＬの一部として入力されるコマンドを解析し，各々の機能を実行するために，エディタ，トレーサ，トランスレータなどを管理する。Ⓐエディタは，コンソールを使用してユーザがシステムと会話を進めながら行なうソースプログラムの編集機能を実現する；Ⓑトレーサは，インタプリタを管理して，必要なトレース結果を編集出力するなどの処理を行なう。

(3) 入出力処理およびＱＡ-１との通信制御：コンソールやプリンタなどの入出力動作を直接に指示するとともに，ＱＡ-１へのＩＬ列，表の転送，インタプリタのイニシャルロード，ＱＡ-１からのトレース結果の返送などを制御する。

(4) ファイルの管理：ミニコン側にはディスクファイルが付加されており，ＣＬやユーザ個別のＥＬを処理するためのシステムが格納される。このほか，ミニコンやＱＡ-１のバックアップメモリとしても機能する様に管理する。

ＱＡ-１は，トランスレータによって生成されたＩＬ列や表を解釈実行するファームウェアインタプリタである。ミニコンとＱＡ-１との機能分担は，高級言語処理システムにおけるハード／ファーム／ソフトウェアのトレードオフを考慮して，コンパイルとインタプリトの総合的な効率を向上させるためである。すなわち，(1)高級言語レベルにおいて，プログラムのデバッグを実行すること；(2)ポインタ型データ，動的配列，再帰的手続き呼び出しなどの処理のために，ダイナミックなメモリ空間の管理が必要であること；(3)多次元配列，可変長データ，リスト構造，構造型データなどの複雑なデータ構造をファームウェアで操作すること；などのインタプリタによる処理の特徴を考えて，ＩＬのレベルを設定した。だから，ＩＬのレベルを高級言語レベルに引き上げて，その解釈に要するオーバヘッドをＱＡ-１のファームウェアとハードウェアを利用して軽減することをねらい，コンパイラのもつ特長とインタプリタのもつ特長とを兼ね備えた高級言語処理システムの実現を図った。

6. むすび

FELPSは，高級言語レベルでマクロ化された手続きを，ユーザがさらにファームウェア化できることを特長とする高級言語処理システムである。高級言語レベルでプログラムをマクロ化あるいはサブルーチン化できる機能は，ソースプログラムを重複して記憶することを避けてメモリの有効利用を図るためだけでなく，プログラミング機能やドキュメントとしてのプログラムの価値を高めたりすることを目標として，ソフトウェアコンパイラなどに付加されている。しかし，FELPSの様にマクロ化された手続きをさらにファームウェア化するシステムは，ソフトウェアの生産性を向上させるだけにとどまらず，その実行効率の向上をもめざす点で，ソフトウェアだけにたよるシステムとの本質的な相違がある。

ソフトウェア蓄積の影響が大きい中・大型機では，アーキテクチャの大幅な変更を必要とする高級言語計算機の普及は困難であるとされている。しかし，使い捨ての，あるいは移植性を余り必要としないソフトウェアを作成することの多い小型機においては，今後もハードウェアコストの低減とともに計算機のパーソナル化が進むと予測される。だから，問題適応性をソフトウェアのみならず，ファームウェア，ハードウェアでサポートして実行効率の低下を招くことのない高級言語処理システムを構成することで，ユーザの多様なニーズに応えることは重要である。

現在使用されている高級言語の大部分は，問題の処理手順を連続的に書き下す手続き向き言語のため，その処理における並列性は少ないことが予想される。しかし，QA-1の特長はレジスタレベル並列処理方式を採用していることであり，これまでの研究結果[2]から判断しても，高級言語の処理をファームウェアによる処理レベルにまで分解すれば，充分にQA-1アーキテクチャの有効性が示されることを期待できる。この点についての詳価分析により，高級言語処理システムにおけるファームウェア/ハードウェアのトレードオフを再検討する必要がある。

拡張可能な言語というとALGOL68やPASCALなどが著名であるが，現在使われている高級言語には何らかの形で拡張性が装備されていることが多い。プログラミング言語の拡張能力は，処理システム全体の問題適応能力を大きく左右するが，これも計算機側からもサポートする試みが本研究の主眼である。今後は，CLの拡張能力を検討しつつ，ソフトウェアエンジニアリングを重要視する立場からも充分な考慮をはらったシステム製作を進めていく予定である。

謝辞

QA-1開発プロジェクトの中心メンバーであった小柳 滋氏（現・東芝）には有益な助言をいただき，深謝いたします。又，本研究に関して熱心に討論していただいた本学大学院生 北村俊昭 君をはじめとする萩原研究室の皆様に感謝いたします。

参考文献

[1] 小柳，柴山，富田，萩原："マイクロプログラム制御計算機QA-1のハードウェア構成"，信学会論文誌，Vol.61-D，No.1（Jan.，1978）。

[2] 柴山，富田，萩原："QA-1のファームウェアによる低レベルリストプロセッサ"，信学会技研報，EC78-27（Oct.，1978）。

本 PDF ファイルは 1979 年発行の「第 20 回プログラミング・シンポジウム報告集」をスキャンし、項目ごとに整理して、情報処理学会電子図書館「情報学広場」に掲載するものです。

この出版物は情報処理学会への著作権譲渡がなされていませんが、情報処理学会公式 Web サイトの https://www.ipsj.or.jp/topics/Past_reports.html に下記「過去のプログラミング・シンポジウム報告集の利用許諾について」を掲載して、権利者の捜索をおこないました。そのうえで同意をいただいたもの、お申し出のなかったものを掲載しています。

過去のプログラミング・シンポジウム報告集の利用許諾について

情報処理学会発行の出版物著作権は平成 12 年から情報処理学会著作権規程に従い、学会に帰属することになっています。

プログラミング・シンポジウムの報告集は、情報処理学会と設立の事情が異なるため、この改訂がシンポジウム内部で徹底しておらず、情報処理学会の他の出版物が情報学広場 (=情報処理学会電子図書館) で公開されているにも拘らず、古い報告集には公開されていないものが少からずありました。

プログラミング・シンポジウムは昭和 59 年に情報処理学会の一部門になりましたが、それ以前の報告集も含め、この度学会の他の出版物と同様の扱いにしたいと考えます。過去のすべての報告集の論文について、著作権者（論文を執筆された故人の相続人）を探し出して利用許諾に関する同意を頂くことは困難ですので、一定期間の権利者捜索の努力をしたうえで、著作権者が見つからない場合も論文を情報学広場に掲載させていただきたいと思います。その後、著作権者が発見され、情報学広場への掲載の継続に同意が得られなかった場合には、当該論文については、掲載を停止致します。

この措置にご意見のある方は、プログラミング・シンポジウムの辻尚史運営委員長 (tsuji@math.s.chiba-u.ac.jp) までお申し出ください。

加えて、著作権者について情報をお持ちの方は事務局まで情報をお寄せくださいますようお願い申し上げます。

期間：2020 年 12 月 18 日 〜 2021 年 3 月 19 日
掲載日：2020 年 12 月 18 日

プログラミング・シンポジウム委員会

情報処理学会著作権規程
https://www.ipsj.or.jp/copyright/ronbun/copyright.html