計算機アーキテクチャ 102－4
（1993．10．21）

拡張性を考慮した マイクロプログラマブルプロセッサ Proteus

石川 明雄* 山西 正一郎* 朝長 宜央** 小原 啓義*

早稲田大学理工学部* 株式会社アプリックス **

本論文では，内部構成モジュールの拡張が容易なプロセッサ Proteus のアーキテクチャに関する研究について述べる．Proteus は，機能単位であるユニットとブロックを内部バスに接続することにより構成される．ブロックはそれ自体が一つのプロセッサとして動作可能な単位であり，各々に対して与えられるマイクロ命令により制御される．また，ユニットはブロックに対して接続され，その各々は ALU やレジスタファイルなどのプリミティブな単位である．ユニットはそれが接続されているブロックに与えられたマイクロ命令により制御される．Proteus は，ユニットの追加により特殊なハードウェア機能の付加，ブロックの追加により並列プロセッサとしての動作が可能である．

An implementation of microprogramable prosessor Proteus : Extensible internal module CPU

Akio Ishikawa* Syoichiro Yamanishi* Takahiro Tomonaga** Hiroyoshi Ohara*

School of Sci. and Eng., Waseda University* Aplix Corporation**

This paper focuses on the processor architecture of Proteus, which is easy to extend it's internal modules. Internally, Proteus is structured in two types of modules that are Units and Blocks connected to Proteus bus. Each Block works as a processor that is controled with a micro-instruction set provided from the exterior. And each Unit is a primitive modules attached to a Block, for example, Arithmetic Logic Unit and Register File. Units can be controled with a bit field in the micro-instruction set given to a Block. Proteus architecture can add new specific functions by adding new Units, and can work as parallel pocessor by adding new Blocks.

1 はじめに

現在，コンピュータの応用分野は多岐にわたり，様々な分野で利用されている．一般的なの汎用プロセッサでは，ハードウェアを単純化し，パイプライン，キャッシュ，スーパスカラといった技術を導入することにより高速な演算を行うものが多い．そのようなプロセッサの場合，ハードウェアの単純化により，実際のアプリケーションの要求とプロセッサの基本命令との間の隔たりは，限られたハードウェア資源を効率よく用いて高速に実行されるコードを生成する最適化コンパイラにより吸収しなければならない．しかし，最適化コンパイラは，開発に多大なコストがかかり，また，プログラムパラダイムとプロセッサの基本命令の間の隔たりを根本的に埋めるものではない．

さらに，汎用プロセッサでは有効に実現できないようなアプリケーションも少なからず存在する．例えばリスト処理や画像処理などのアプリケーションでは，プログラムパラダイムとプロセッサの基本命令の間の隔たりは大きく，これらのアプリケーションを汎用プロセッサで実現しようとしても，そのプログラムパラダイムを効率よくプロセッサの基本命令に変換するとことはできない[1]．そこで，このような分野のアプリケーションを高速に実行するためには，アプリケーションのプログラムパラダイムが要求する機能をハードウェアで処理する能力を持つ専用プロセッサが必要となる．

このような専用プロセッサは，対象とするアプリケーションに有効なアーキテクチャが採用されねばならず，そのプログラムパラダイムに依存しないと効率的な処理を行なうことはできない．そこで設計に際しては，処理系の振舞いを解析し，ボトルネックとなっている機能をハードウェア化し，トップダウン式に設計される．しかし，このような設計方法では，処理目的の変化により処理能力が著しく低下するため，ソフトウェアの柔軟さにハードウェアが追随しないことは明らかである．

また，並列処理システムに適したプロセッサには「演算処理能力と通信能力のバランスが取れている」ことが求められる．しかし，演算処理能力と通信能力のバランスはアプリケーションのプログラムパラダイムにより大きく変化するため，そのアーキテクチャを一般的な形で決定することは難しい．

現在では，プロセッサの基本モジュールの設計は究めて簡便にかつ安価に行なわれ，処理に即したサブプロセッサを作製することは難しくない．しかし，サブプロセッサに対する自由度という点では，基本モジュールに強く依存しており，ハードウェアの拡張性はないといっても過言ではない．本来のコンピュータの在り方からすれば，現在のようにハードウェアが固定されている必然性は全くない．

以上のことから，拡張に関する自由度が高いプロセッサは，ハードウェアの拡張によりプログラムパラダイムとプロセッサの基本命令の間の隔たりを吸収し，効率的なアプリケーションの実行を可能とするものである．そこで我々は，プロセッサの構成モジュールの変更・追加が容易なアーキテクチャに関する研究を行い，ハードウェアの拡張性を考慮したマイクロプログラマブルプロセッサ Proteus の設計，開発を行っている．本稿では，Proteus の基本構想，概要，及び開発状況について述べる．

2 Proteus アーキテクチャの基本構想

- ハードウェア拡張性の重視

  ハードウェアの拡張が容易なように，単純な基本構造，簡明な命令セットを持つプロセッサとし，制御回路を簡潔化する．アプリケーションを効率よく処理するために必要な機能は，ハードウェアを追加することにより実現する．これが，Proteus アーキテクチャの最も重要，かつ特徴的な構想である．

- マイクロ命令の有効利用

  追加したハードウェアを効率よく制御するために，マイクロプログラム制御方式とする．マイクロ命令は WCS(Writable Control Storage) に格納されハードウェアを制御する．Proteus のマイクロ命令は，追加されたハードウェアを制御する自由度を持っており，プログラムパラダイムとプロセッサの基本命令との隔たりを吸収する．また，追加したハードウェアに対し作成されたマイクロ命令のライブラリをインクルードすることにより，効率的なアプリケーション開発が実現できる．

- Register to Register 指向の命令体系

  主記憶装置への頻繁なアクセスは，処理効率の低下を招く．そこで，内部レジスタ間の演算を命令セットの基本に置く．これにより，ハードウェアの追加による性能向上の妨げとなる主記憶装置とのデータ転送を押えることができる．

3 Proteus アーキテクチャの概要

3.1 ハードウェアの拡張

Proteus は，目的の機能を実現するいくつかの単位から構成される．各単位は，外部に開放された内部バス (Proteus Bus) 上に接続され，全体を構成する．

ハードウェアはユニット，ブロックという 2 種類のハードウェア単位で追加する．

- ブロックによる拡張

  ブロックとは，それ自体が一つのプロセッサとして動作可能なハードウェア単位である．ブロックには基本ブロックと拡張ブロックがある．基本ブロックは Proteus の最もプリミティブな構成要素であり，Proteus 全体の制御を司る．拡張ブロックは，基本ブロックに対して追加さ

れるブロックである．試作機では，基本ブロックに対し最大3つの拡張ブロックが追加される．

Proteusでは、命令コードをデータと分離して格納するハーバードアーキテクチャを採用している．従って、命令コードは各ブロックごとに与えられ，データは基本ブロックのみが外部との転送を行う．

基本ブロックと拡張ブロックは，MPRF(Multi-Port Register File)とSync.(Synchronizer)で接続される(図1)．

MPRFは基本ブロックと拡張ブロックの間とのデータの通信を行なうためのマルチポートレジスタ郡である。

Sync.は基本ブロックと拡張ブロックの同期を取るための機構であり，これにより拡張ブロックごとに基本ブロックとの同期を制御する．この制御はマイクロ命令で指定する．

- ユニットによる拡張

  ユニットとは，レジスタやALU(Arithmetic Logic Unit)といったプロセッサ内部の構成要素である．各ブロックにはプロセッサとして最低限必要な機能が基本ユニットとして与えられており，この基本ユニットに対して拡張ユニットが追加される．拡張ユニットは，それが付加されているブロックに与えられたマイクロ命令によって制御される．

3.1.1 ブロックの構造

各ブロックは基本ユニットと拡張ユニットから成る(図3)．

基本ブロックの基本ユニットは，三つのデータパスを持ち，RF(Register File)，ALUの他に，シーケンサ，割込みコントローラ，クロックジェネレータ，フラグレジスタを持つ．さらに，命令コードを格納するWCS，パイプラインレジスタ，外部バスとのデータの授受を司るMADR(Memory Address and Data Register)がある．外部バスにはVMEbus(Versa Module Europe[2][3][4])を使用し，これを通じてデータメモリ，I/Oなどと接続される．また、命令コードはデータと分離してWCSに格納する。

拡張ブロックの基本ユニットは，ほぼ基本ブロックのそれと同じ構造を持つが，外部とのデータの授受は基本ブロックが行うため，MADR，VMEbusインターフェイスユニットはない．

3.1.2 ユニットの構造

各ブロックは，基本ブロックに拡張ユニットを追加することにより拡張される．拡張ユニットは，その構造や目的によって，表1のように四つに分類される．

- Type A

  ALUの機能を追加するユニットで，マイクロ命令のファンクションフィールドによって制御される．このユニットにより，基本ユニットが持つALUにはない機能を追加する．例えば，乗算機能を持つALU，特定の演算を行なうALUが挙げられる．

表1: Extention Unitの種類

	Unit Select	Function Select	Example
Type A	Func (Sel)	Func (Inst)	ALU
Type B	Bus Address		Register File
Type C	Func (Sel) Y-Bus Address	Func (Inst)	Sift Register Register Stack
Type D	Func (Sel) Y-Bus Address	Func (Inst) Y-Bus Data	Sequencer

- Type B

  レジスタを追加するユニットで，マイクロ命令のアドレスフィールドによって制御される．

- Type C

  複雑な機能を持つレジスタを追加するユニットで，ファンクションフィールドとアドレスフィールドによって制御される．例えば，シフトレジスタやレジスタスタック(レジスタウィンドウ)が挙げられる．

- Type D

  Type Cと同様に複雑な機能を持つレジスタを追加するユニットであるが，特にユニットにシーケンサを持ったユニットである．このユニットは、起動されると基本ユニットとは独立して処理を行い，処理の終了を割込みによって知らせる．例えば，VMEbusインターフェイスユニットが挙げられる．

3.2 マイクロ命令による制御

図4にProteusのマイクロ命令のフォーマットを示す．

Type 1が通常用いられるフォーマットである。以下に主なビットについて説明する。

- Clk

  Proteusでは，可変長クロックを用いている．このビットによって，8通りのクロックパターンを選択することができる．可変長クロックを用いることにより，処理速度の異なるユニットを付加した場合でも，全体の効率を落すことなく処理することが可能である．

- Sync

  基本ブロックと拡張ブロックの間の同期をとるためのビットである．この値がSyncの時には同期がとられ，Asyncの時には同期はとられない．基本ブロックでは，この4ビットが各拡張ブロックに対応する．また，拡張ブロックでは，自分のビットが基本ブロックに対応する．あるブロックの同期ビットがSyncであった場合，相手のブロックの同期ビットがSyncになるまで待ち状態に入り実行を中断する．

- Func

  各演算ユニットの機能を指定する.

- Y-bus

  Y-bus 上のデータを格納するレジスタを指定する.

- S-bus, R-bus

  S-bus, R-bus にデータを出力するレジスタを指定する.

Type 2 は即値データを使用する場合に用いる. このデータフォーマットでは, Imm0 と Imm1 のビットを結合した値が Y-bus 上に出力される.

3.3 Proteus の特徴

図 5にブロックとユニットによる拡張が行われた Proteus の全体の構造を示す.

拡張ブロックを追加することにより, Proteus は一種の並列処理プロセッサとして動作する. また, 各ブロックに拡張ユニットを追加することにより, その特徴や性能を多岐にわたって変化させることができる.

Proteus の同期機構には, 連続同期モードと非連続同期モードの二通りのモードがある.

- 連続同期モード

  連続同期モードでは, 各ブロックにおいて常に同期ビットを Sync にして動作させる. この場合, 各ブロックはクロックサイクルを完全に同期させて命令を実行する.

- 非連続同期モード

  非連続同期モードでは, 基本的に各ブロックを同期させずに動作させる. ブロック間で同期が必要となった場合にのみ, 同期ビットを Sync にして同期させる. この場合, 全体の動作はレジスタ結合のマルチプロセッサとなる.

4 ユニットの開発

本章では, 既に設計が終了しているユニットについてその概要を説明する.

4.1 VMEbus インターフェイスユニット

Proteus では, 外部とのデータの授受に VMEbus を用いる. VMEbus 上のデータメモリ, I/O 等とのデータの授受を行うユニットが VMEbus インターフェイスユニットである. このユニットは基本ブロックのみに接続される. VMEbus インターフェイスユニットでは以下の機能を実現する.

- VMEbus を介して接続されているデータメモリ, I/O とのデータ転送機能
- 割込み要求機能
- 割込み応答機能

4.2 特徴的な機能

- MADR

  VMEbus インターフェイスユニットでは, 外部とのデータの授受を行うレジスタとして MADR(Memory Address and Data Register) を用いている. これは, MAR(Memory Address Register) と MDR(Memory Data Register) を共用するレジスタである. Proteus では, アドレスをデータと同様に扱うことができ, これを用いてデータメモリから読み込んだデータをそのままダイレクトにアドレスとして出力することが可能である. これにより, Lisp 等のリスト処理の多いアプリケーション実行時に高速なアクセスが可能となる.

- アクセスエラーの検出

  VMEbus へのアクセスで発生し得るエラーの大部分をハードウェアで検出し, Proteus に対し割込み行う.

- ロード遅延スロットへの対応

  VMEbus インターフェイスユニットは, Type D の拡張ユニットとして設計されている. つまり, 内部にシーケンサを持っている. そのため, データのアクセスが完了するまでの間, Proteus 側とは独立に処理を行う. これにより, VMEbus とのアクセスの間に Proteus 側で他の処理を行うことが可能である.

4.3 拡張ユニット／レジスタ・ウィンドウ

レジスタウィンドウはプロシージャコールを高速化する拡張ユニットである. プロシージャコールでは, スタック操作に伴う主記憶装置へのアクセスが増すため, そのオーバヘッドが大きい. レジスタウィンドウは, プロシージャ間のデータの授受を支援するハードウェアである. 試作機 Proteus のアーキテクチャを考慮し, 以下の特徴を持つレジスタウィンドウを設計した (図 6).

- in/out 領域, local 領域を明確に定義せず, ユーザが必要に応じて設定できる.
- フレームの移動量が可変である. これにより, 未使用レジスタをなくし, 効率よくレジスタを利用できる. また, スワップ時に効率のよいレジスタの保存, 復帰ができる.
- 現在のフレーム開始位置を保存するレジスタとして CWP(Current Window Pointer) があり, これを用いたレジスタ相対アドレッシングにより目的とするレジスタをアクセスする. フレームの移動は, CWP の値を変更することにより行なう.
- フレームの移動量は, プロシージャコール時にソフトウェアが申告する. リターン時は, ハードウェアがフレームをコール前の位置に復帰させる.

- スワップの判定を行うレジスタとして4ビットから成るWIM (Window Invalid Mask) がある．レジスタウィンドウは128個のレジスタから成り，それらは32レジスタごとに4ブロックに分けられている．WIMの各ビットは各ブロックに対応しており，そのブロックが有効か否かを示す．

- スワップを検出はハードウェアが行う．スワップを検出した場合には，Proteusに割込みをかける．

- 割込み処理ルーチンでは，ソフトウェアによりレジスタの退避を行う．この処理が可能なように，CWPをユーザの操作が可能なレジスタとする．

- スワップは，1ブロック32レジスタ単位に行う．

5 ソフトウェア環境

5.1 アセンブラ PASM(Proteus Assembler)

Proteusのソフトウェアを開発するためのアセンブラには，ハードウェアの拡張に対応できる能力が求められる．アセンブラとして次の3点が問題となる．

- 全体が複数のブロックにより構成され，制御記憶部が独立している点

- ブロック内のハードウェア構造が変化する点

- マイクロ命令の形式が変化する点

これらの問題点を解決するために，PASMは次の機能を持つ．

- ターゲットシステム全体の構成を認識する機能

  ターゲットとなるプロセッサの持つ命令形式，ハードウェア構成をファイルに定義し，それを読み込むことで，ハードウェアの変更に対応する．

- プログラムの属するブロックを明示する機能

  これにより，複数の記憶装置を持つプロセッサに対応する．

これらの機能を実現するための命令として以下の命令がある．

- ターゲットシステムの定義に関する擬似命令

  ターゲットプロセッサに関する全ての情報をソースとして与える．この命令により，命令形式，各々のレジスタ，オペコード，ユニットの機能，記憶装置のタイプ，ブロック，システム全体の情報をトップダウンに定義する．

- プログラムを記述する領域 (セグメント) に関する擬似命令

  PASMでは，プログラムを複数の領域 (セグメント) に分割して記述することが可能である．同一の記憶装置にロードされるプログラムは，各セグメントに付けられた記憶装置名，アドレス属性，配置属性などに従って結合される．

PASMは，上記の問題点を解決し，ターゲットシステムの定義を行なうことでさまざまなプロセッサに対応するアセンブラとして作成された．

6 おわりに

本論文では，ハードウェアの拡張が容易なプロセッサProteusの概要について述べた．Proteusは，ユニットやブロックを追加することにより，プログラムパラダイムとプロセッサの基本命令との隔たりを吸収し，目的の機能を持つプロセッサを簡単に構成することが可能である点，また，追加されたモジュールの制御が自由度の高いマイクロ命令により制御される点が特筆すべき特徴である．

ハードウェアに関しては，現在FPGA(LCA[5])を用いてインプリメントを行なっており，近日に完成の予定である．また，特定のプログラムパラダイムを効率よく処理するための拡張ユニットや拡張ブロックについての検討も行っている．

ソフトウェアに関しては，並列性の検出，最適化，ユニットの選択などの処理を行なう高級言語コンパイラについて検討している．PASMではハードウェア構造を記述したシステム定義ファイルによりハードウェアの変更に対応した．高級言語コンパイラでも同様のシステム定義ファイルを用い，Proteusに対応させる予定である．現在，基本ブロックの基本ユニットに対するCコンパイラは作成済みであり，Proteus Architectureにマッチした最適化，並列性検出，ユニット選択などの処理を行なう機能をより高度に付加する予定である．

参考文献

[1] Microprogrammable Processor Proteus 記号処理への応用，朝長 宜央，小原 啓義，情報処理学会第31回全国大会予稿集 4D-2，Oct.，1985，pp. 121-122.

[2] VMEバス・システム完全マスタ，インターフェイス，No. 117，Feb.，1987.

[3] VMEbus アーキテクチャ・マニュアル Revision C. 1 VME MEMBER，Oct.，20，1986.

[4] VMEbus アーキテクチャ・マニュアル，日本モトローラ株式会社半導体事業部，Sep.，1，1984.

[5] XILINX プログラマブル ゲートアレイ データブック，大倉商事株式会社，1991.

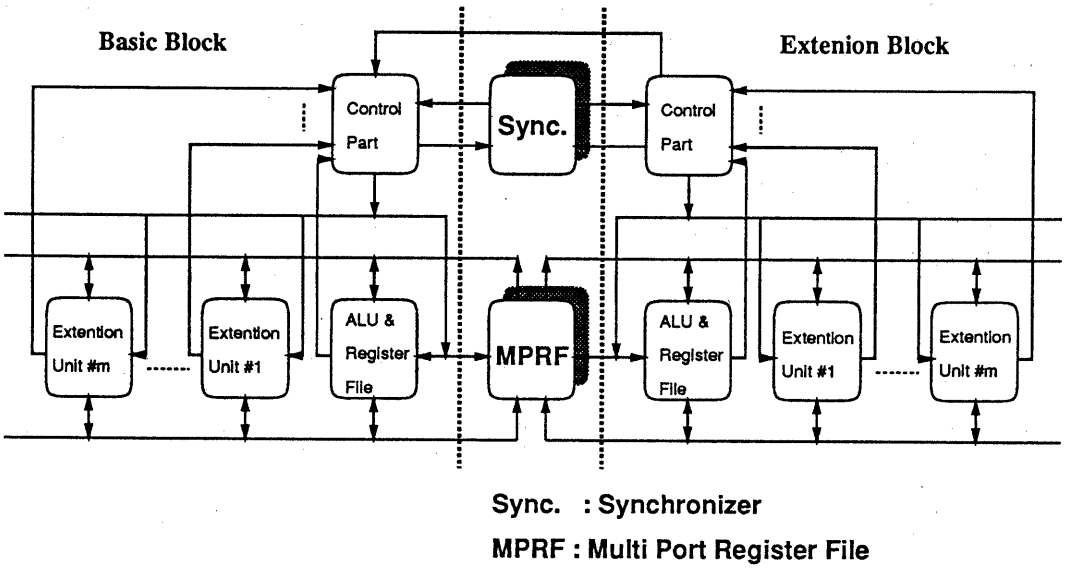


図 1: ブロックの拡張

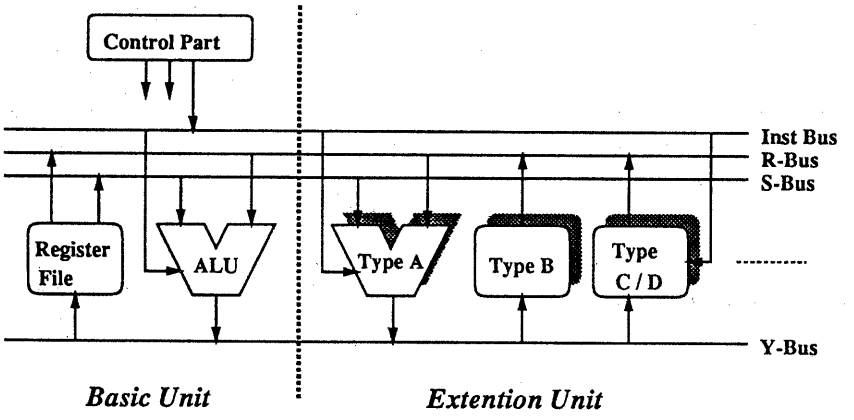


図 2: ユニットの拡張

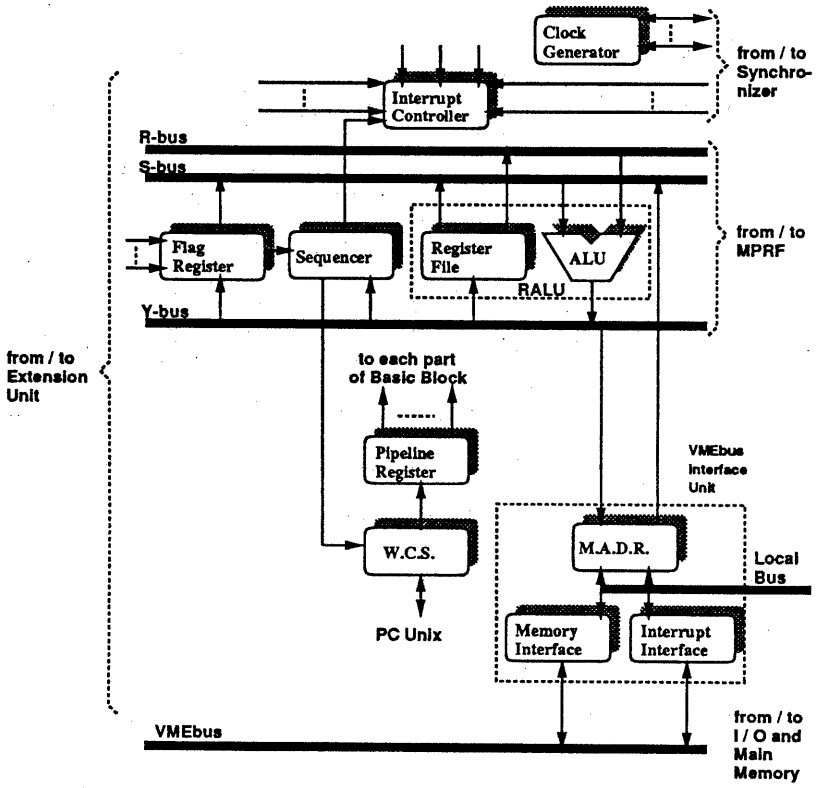
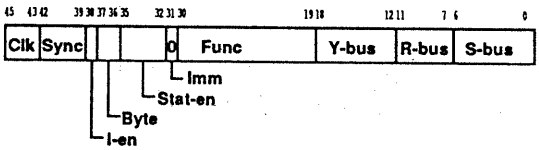
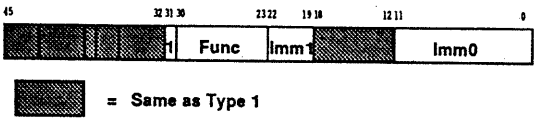


図 3: 基本ブロックの基本ユニット

Type 1 (for Normal Instruction)



Type 2 (for Immediate Data Load)



16bit Immediate Data is given as a connection of Imm1 and Imm0

図 4: Proteus Micro Instruction Format

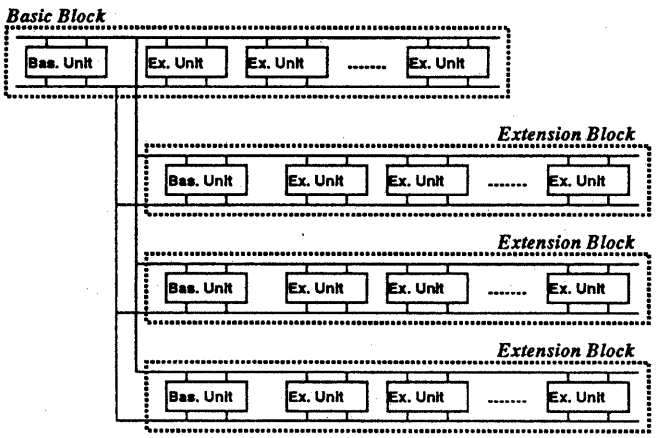


図 5: ハードウェアの拡張

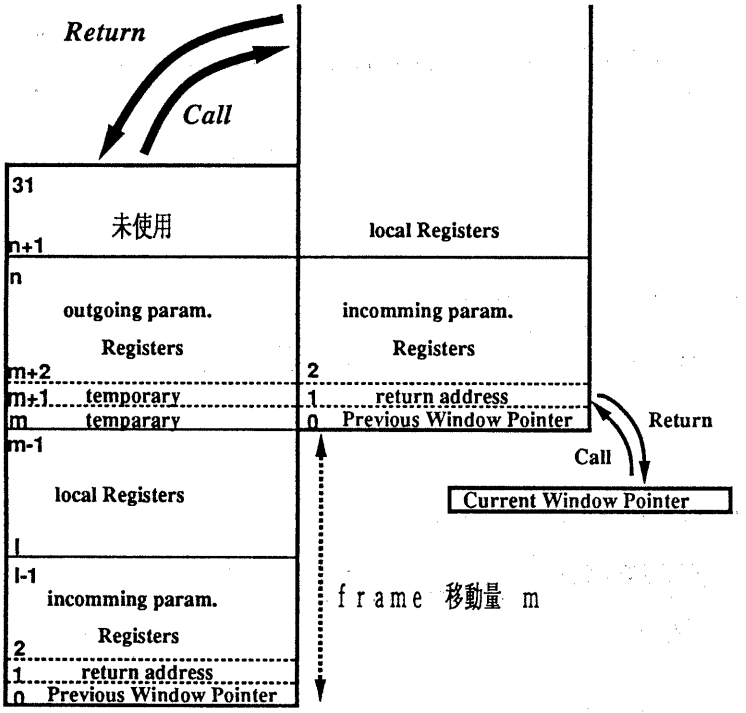


図 6: Proeus レジスタウィンドウ