計算機アーキテクチャ　68－16
(1987・11・13)

マイクロプログラマブルプロセッサ "Proteus"のアーキテクチャ

THE ARCHITECTURE OF MICRO-PROGRAMMABLE PROCESSOR "Proteus"

朝長 宣央,　村田 浩樹,　山田 剛,　小原 啓義
Takahiro TOMONAGA　Hiroki MURATA　Tsuyoshi YAMADA　Hiroyoshi OHARA

早稲田大学理工学部　電子通信学科
Department of Electronics and Communication Engineering,
School of Science and Engineering, Waseda Universityp

あらまし　本論文では，ハードウェアの拡張性を考慮した計算機アーキテクチャを提案している．このアーキテクチャの目的は必要に応じてハードウェアを拡張することによって各種の専用マシンを構成することにある．
ハードウェアの拡張方法には，小規模な拡張な拡張に用いるExtension Unitと比較的大規模な拡張に用いられるExtension Blockがあり，Extension Blockは並列動作についても考慮している．
また，本論文では併せて試作機である"Proteus" について報告している．

Abstract　This paper proposes a new computer architecture which is suitable for extension of hardware. With adaptive extension of hardware, this architecture aims at building up various kinds of CPU for application oriented machines.
There are two ways to extend hardware in the proposed architecture. One is by "Extension Units", for extending small sized hardware, and another is by "Extension Blocks", for extending large sized hardware. And "Extension Blocks" are available for parallel execution, too.
This paper also presents a prototype processor named "Proteus".

1．はじめに

現在，計算機の種々の方面で特定のアプリケーションを対象にした専用計算機のアーキテクチャに関する研究が行なわれている．これらの専用計算機は相当程度の成果をあげており，評価に値するものといえる．しかし，これらの専用マシンは目的ごとに別途のアーキテクチャ採用しているのが現状である．したがってこのような方針を踏襲する限り，アプリケーションごとに異なるアーキテクチャが存在することになり，新たなアプリケーションの発生に伴って費やされる労力は膨大なものとなる．

そこで，アーキテクチャを初めから強固に定める事なく，必要に応じてハードウェアを拡張することによって目的とするプロセッサを得ることができれば，広範な応用分野に対処しうるプロセッサを作成することができると考えられる．

本稿では，このような考察のもとに以下の章に示す拡張可能な計算機アーキテクチャを提案し，現在製作中の試作機であるProteusについて概説する．試作機Proteusの目的は本稿で示すアーキテクチャの有効性を検証することにあり，

①．試作機が既存の各分野の専用機と比肩し得る性能をもつこと，
②．ハードウェアコンパイラ等を利用せずとも容易に専用マシンの設計を行えること，

を最終的な目標としている．

2　Proteusの基本構想

Proteusの基本構想について要約すると以下のようになる．

（1）ハードウェアの拡張性の重視

Proteusは応用にあわせてハードウェアを拡張することによって様々な応用分野で高い，性能／ハードウェア量比を得ることを一つの目的としている．

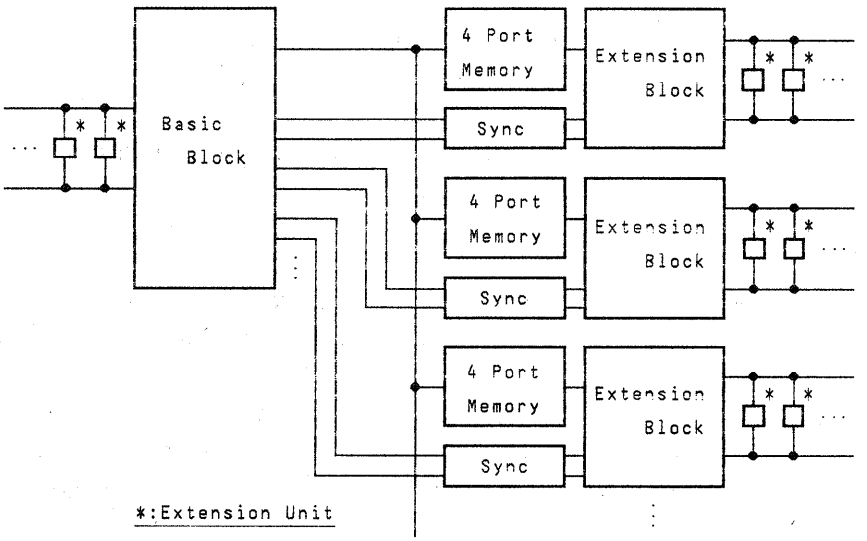


Fig.1　Proteusのアーキテクチャ

Proteusではハードウェアの拡張を効率よく実現するため，ハードウェアの規模によってExtension Unit，Extension Blockの二つの手法を用意している[1]．これらの具体的な内容については次章でのべる．

（2）マイクロプログラムの有効な利用

応用の種類に限定される事なく高い性能を得るため，Proteusではアプリケーションプログラムのほとんどの部分をマイクロプログラムを用いて記述する事を念頭に置いている．したがって，Proteusのマイクロプログラム命令とは通常のプロセッサでの機械命令に相当する．

（3）Register to register指向の命令セット

主記憶への頻繁なアクセスは，プロセッサの高速化に対して致命的であるとの認識から，内部レジスタ間の演算を命令セットの基本においている．このような命令セットをとることにより，ALUの機能強化がプロセッサの高速化にとって効果的な手段となると考えられる．

3　アーキテクチャの概要

Proteusのアーキテクチャの大きな特徴は，複数の独立した制御部をもつプロセッサによって構成されていることである．したがって一種のマルチプロセッサということができるが，通常のマルチプロセッサに比べてプロセッサ間の結合度は強く，直接接続されたプロセッサ間の同期は1命令毎に可能である．

ProteusではこれらのプロセッサをBlockと称しており，Proteus全体の制御を司るBasic BlockとBasic Blockによって制御されるExtension Blockがある．Fig.1に示すとうり，一つのBasic Blockには複数のExtension Blockが接続可能であり，Basic BlockとExtension Blockは互いにマルチポートレジスタファイルを介して接続している．各々のBlockはそれ自身単体で動作可能なマイクロプログラマブルプロセッサであり，シンクロナイザと呼ばれるハードウェアにより命令毎に同期をとることができる．Extension Blockはひとつの Basic Blockに対して，複数接続可能だが，以下の章では簡単のため特にことわらなければExtension Blockは1台として説明する．

各BlockにはProteusのハードウェア拡張法の一つであるExtension Unitの増設が可能であり，これについては3．2で述べる．

また，命令のフォーマットは Register to Register Operationを基本にし，多種のALUを駆動するため高レベルの垂直水平型としている．これについては，4章で述べる．

3．1．　Basic Block

Basic Blockは文字通りProteusの中核を成している．Basic Blockの構成を，Fig.2に示す．内部バス（以下，Proteus Bus）はそれぞれ16bitでSource Bus2系統（S-Bus及びR-Bus），Distination Bus（Y-Bus）1系統からなっており，内部レジスタやALU，メインメモリインターフェース等の各種のExtension Unitはこのバスに接続される．

Basic Blockには次節で述べるExtension Unitを必要に応じ増設する．したがって，Basic Blockのマイクロプログラムシーケンサ等の基本的な部分とExtension Unitの使用方法が異なる場合，プログラマにとって使いづらいものになる恐れがある．そこでBasic Blockの基本的な部分はExtension Unitとの一貫性をたもつため，類似の制御方式をとっている．

なお，試作機ではExtension Blockの設計は，VMEバスとのインターフェースをのぞいて事実上Basic Blockと同一の設計になっている．このため，Extension Blockに関する説明はBasic Blockと異なる部分についてのみ３．３に記す．

３．２．Ｅｘｔｅｎｓｉｏｎ　Ｕｎｉｔ

Extension Unitは，Basic Block，Extension Block内に拡張されるハードウェアである[1]．これらはFig.3,Table.1に示されるように，その目的や機能によって４種類（Type A～D）に分類できる．また，プログラマの観点から，基本的には内部レジスタの増強に相当するもの（Type B,C,D）とALUの機能増強に相当するもの（Type A）に分類する事もできる[2]．

Extension Unitのようにプロセッサの内部バスにALUや内部レジスタを追加していく方法は容易に実現が可能である．しかし，このようなExtension Unitのみによるハードウェアの拡張ではBasic Blockのように内部バスが一組のみである場合，複数のALUユニットを同時に動作することはできない．したがって，Extension Unitの数が増えると各々のUnitの利用率が低下してしまい，ハードウェアの並列性を十分に生かすことができない．そこで Proteus では Extension Unit以外に次節に示すExtension Blockを提案しこれを併せて用いることによりこの問題を回避し，さらにBasic Block/Extension Blockの並列動作も可能にしている．

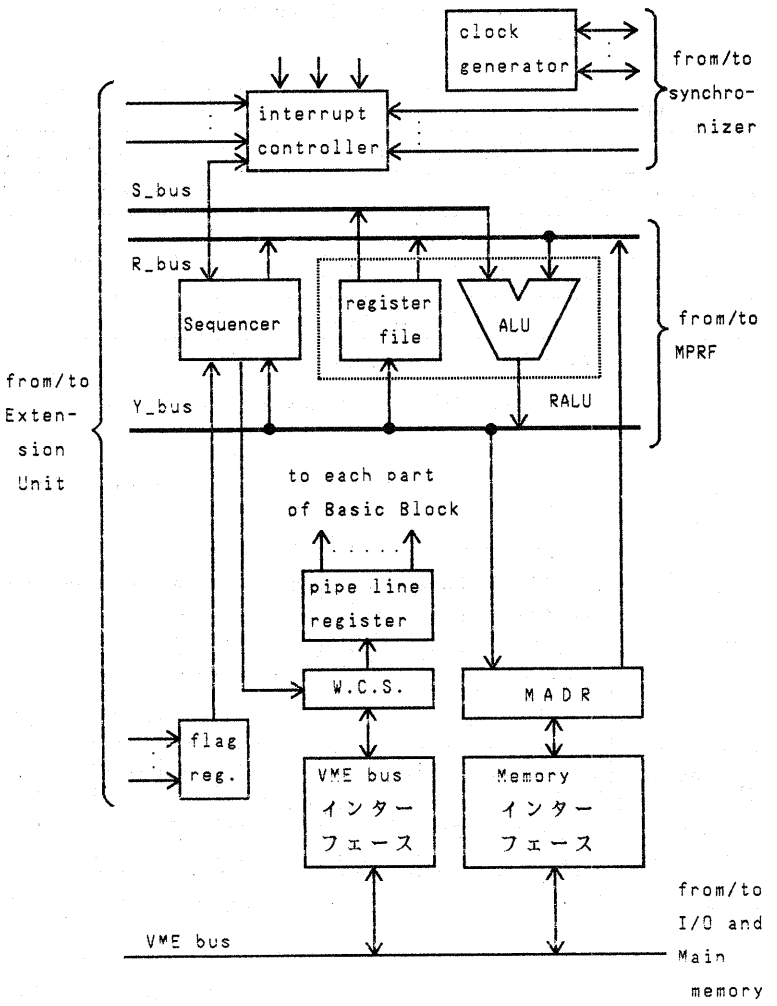


Fig.2 Basic Block構成図

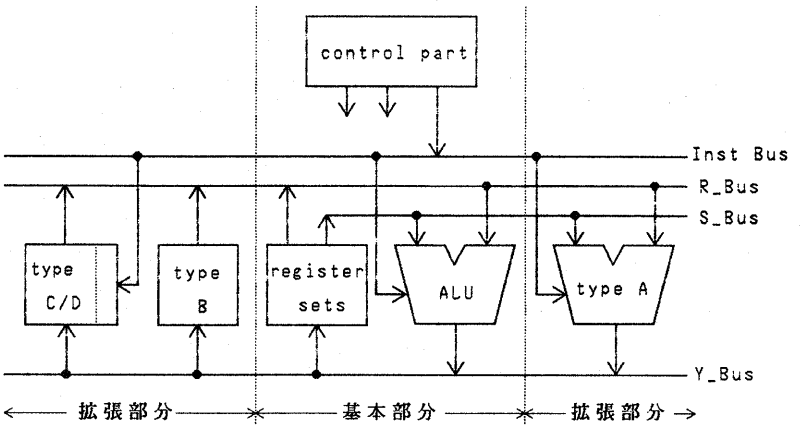


Fig.3 Extension Unitの種類

	Unitの選択	機能の選択	例
Type A	funcフィールド	funcフィールド	Built in ALU,Barrel Shifter
Type B	S_bus/R_bus/Y_busのアドレス	———	内部レジスタ(Accumulator)
Type C	funcフィールド	funcフィールド	Tag Jump Table
	S_bus/R_bus/Y_busのアドレス	———	
Type D	Y_busのアドレス	funcフィールド*/Y_busのデータ	シーケンサ

*:イミディエート命令のときだけ，利用できる．

Table.1 Extension Unitの種類

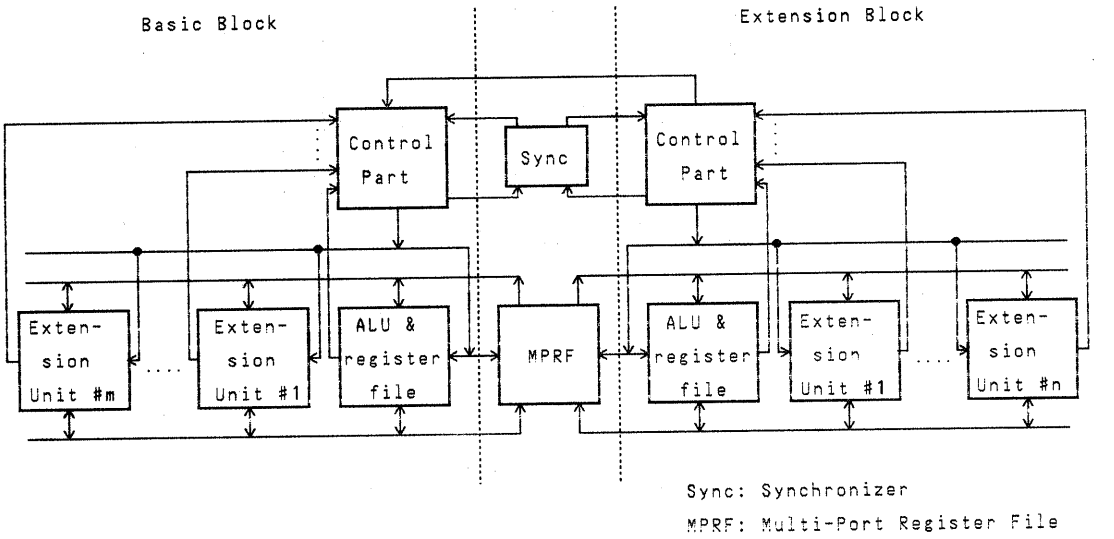


Fig.4 Proteusの構成（基本部分のみ）

3．3 Extension Block

Extension Unitを複数同時に動作させるためには内部バスを複数設けるだけではなく，複数のExtension Unitに命令を同時に与える必要がある．

このため，PuroteusではFig.4に示すようにマイクロプログラムシーケンサと制御記憶をBasic Block用とExtension Block用の多重構成とし，両Blockの間にマルチポートレジスタファイル（Multi-Port Register File，以下MPRF）をインターフェースとして配している．MPRFは両方のBlockからは内部レジスタとして利用できるためBasic BlockとExtension Blockの内部バスを分離することに役だっている．また，Extension BlockはBasic Blockとは別に，独自に制御部を持っているので，各々のBlockは独立して命令の実行が可能である．

Basic BlockとExtension Blockの情報の授受は，両ブロック間の引き数／返り値の渡しに用いられるマルチポートレジスタファイル，ブロック間の同期をとるシンクロナイザ，Extension BlockからBasic Blockへの割り込みにより行われる．

シンクロナイザによる両ユニット間の同期制御は必要に応じて連続同期モード型・非同期モード型の２通りの方法があり次章で具体的な使用法を述べる．

4 シンクロナイザによる同期機構

シンクロナイザはBasic BlockとExtension Blockの間の同期をとるのに利用される．Basic Block とExtension Block用マイクロ命令には，それぞれシンクロナイザを用いてブロック間の同期をとるために同期ビットがあり，Sync/Asyncのいずれかの値をとる．各ブロックのシーケンサは自分のブロックのマイクロ命令中の同期ビットがSyncであった場合，他方のブロックの命令中の同期ビットがSyncになるまで待ち状態に入り実行を中断する．同期ビットがAsyncの場合は各シーケンサは独立して命令の実行を進める．

以下，同期ビットを利用したProteusのExtension Unitの制御方式である連続同期モード・非同期モードの利用法について述べる．これら２のモードは実際には，厳密な使用法の区別がある訳ではなく，どちらのモードかはっきり区別しにくい場合も存在すると思われるが，プログラマがExtension Blockを使用したい時の一つの指標として定めている．

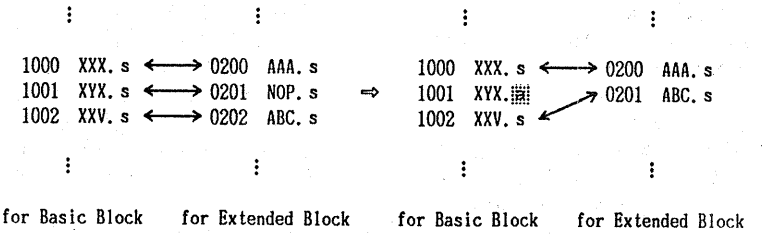
4．1 連続同期モード

連続同期モードでは基本的に両方のブロックとも同期ビットをSyncにしたままで動作させる．これによって両ブロックのクロックサイクルが完全に同期して各命令を順次実行する．このとき，両ブロック内のALUは同時に動作することが可能になるため，ハードウェアの並列性を有効に利用できる．また，Extension BlockがMPRFを介して渡されたデータを処理するのと同時に，Basic Blockが次のデータを用意するといったように，両ブロックをパイプライン的に利用することもできる．

このモードにおいては，両方のBlockが常に演算を行っていることは必ずしも多くはないと想定され，したがってNo Operation命令が頻出する可能性がある．このような場合，No operation命令をいちいち挿入していたのでは，無駄が多い上，相対するブロックでの命令数にNo operationの数を合わせなくてはならず，非常に不便である．そこでどちらか一方の処理ブロックが使用されずNo Operationになる場合は，連続同期モードにおいても他方のブロックの同期ビットをAsyncにすることにより不要な No Operation命令を抑制している（Fig.5）[11]．

4．2 非同期モード

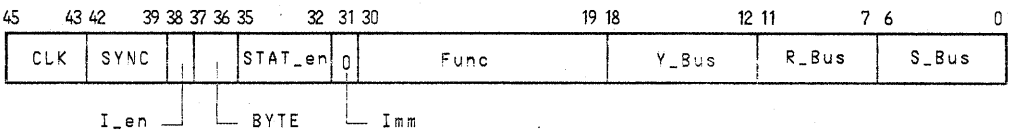
非同期モードでは基本的には両方のブロックとも同期ビットをAsync にしたままで別々のプロセスとして動作させる． これは単純な演算ユニットでは処理しきれないサブプログラムや，並列アルゴリズムが明快に与えられた処理を行う際に，Extension Block側をBasic Blockから切り離して独立のプロセッサとして利用するものである．Basic BlockとExtension Blockは命令中のSyncフィールドによって同期を取り合う，レジスタ結合された一種のマルチ



但し，XXX, XXV, AAA, ABC などは通常命令を
NOP はNo operationを表し，s，aはそれぞれ
Sync, Async を表す．また ⇔ は同期している
命令同志を示している．

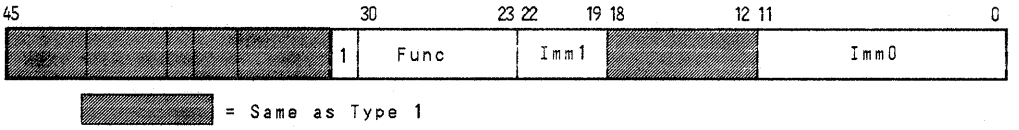
Fig.5 プログラムでExtension Block 内のＮＯＰ命令を抑制する方法

Type 1 (For normal instruction)



S_Bus,R_bus : Source Address
Y_Bus : Destination Address
FUNC : ALU Function Select
Imm : Instruction Type Select
STAT_en : Status Registers update enable
BYTE : Byte or Word Calc width
I_en : Interruot enable
SYNC : Sycronization for other Block
CLK : Machine Cycle Period Control

Type 2 (For Immidiate Data Load and To drive a Type-D unit)



16 bit Immidiate Data is given as a concatenation of Imm1 and Imm0

Fig.6 Proteus Instruction Format

プロセッサとなる.

5. Proteusの命令

5. 1 命令の形式

前章までに述べたように, Proteusにはさまざまな形態のハードウェア (Extension Unit) が付加される. このため, ともするとそのインストラクションフォーマットも非常に多種にわたってしまう可能性がある. Proteusでは命令を高度の垂直水平型とし, フィールドの割付を固定し各フィールドをデコードしている. このため, 各フィールドの意味を限定して, Extension Unitの設計を容易にしている.

Proteusの試作機の命令は, Fig.6のように基本的に2種類の命令に分類される*1.

5. 2 命令中のフィールド

Fig.6の各フィールドのうち, S_bus, R_busはそれぞれ読み出しバスに接続されたレジスタ類を指定するのに用いられ, 一方Y_busは書き込みバスのレジスタ指定である. FuncはALUの機能を選択するのに利用され, このフィールドをデコードする事によってType-AのExtension Unitの選択を行なう.

両種の命令で共通なフィールドのうち, Syncフィールドは Extension Blockとの同期をとるために用いられる[1]. 試作機では, Syncフィールドは4bitとられており, 従って接続可能なExtension Blockの数は4となる. なお, Extension Blockの命令ではSync BitはBasic Blockに対する1bitのみ利用される.

そのほか特徴的な事項としては, 割り込みを受け付けるか否かのビットをI_enとして設けていることがあげられる. Proteusではプログラム中に, いわゆるきわどい部分が頻繁に発生することが予想される. そこで, 水平型のマイクロ命令をもつマイクロプログラム計算機にしばしばみられるように割込みの禁止/許可をEI,DI*2などの独立の命令で処理するのではそのオーバーヘッドは無視できないものになると考えらるため, 専用のフィールドを割り付けている.

脚注1. 文献[2]の発表時より, 各フィールドの名称のほか, 内部レジスタのアドレスに対するインデックス修飾を破棄, ステータスレジスタのアップデートを制御するフィールドを増加が変更されている.

脚注2. Enable Interruot,Disable Interrupt

Type 1の命令は，一般の演算に用いられる命令であり，先に述べたようにExtension Unitおよび，内部レジスタを選択するフィールドを持っている．

Type 2の命令は基本的にはイミディエートデータをレジスタに書き込む命令である．このタイプの命令ではFuncフィールドは通常用いられることはないが，Type DのExtension Unitに命令を与える必要がある場合に利用される．プロトタイプにおける条件分岐命令はこの方法で実現されている．

尚，試作機では後述するようにRALUとして74AS888[3]を使用しているが，このRALUに上記のType 1または2の命令に分類できない特殊な形式の命令が幾つかある．これらは，74AS888に固有な命令であるがここではType 1の特別な場合としている．

6．プロトタイプシステムの構成

プロトタイプの構成をFig.7に示す．設計に際し，使用する半導体プロセスはAdvanced Shottokey TTLないしFast-TTLで実現可能な範囲としており，市販のLSIを中心に使用している．

また，Proteusはi8086（MS-DOS，移植中）によるサービスプロセッサの管理下で，バックエンドプロセッサとして動作するようになっている．サービスプロセッサは，入出力やユーザインターフェース，Proteusのプログラムのダウンロード，ソフトウェア開発に利用するUnixシステムとの通信を担当するほか，デバッグ時にはProteusの診断を行う．

プロトタイプではBasic BlockとExtension BlockはVMEインターフェースの有無など，一部を除き互いに互換性があるため，以下ではBasic Blockのみについて説明しExtension Blockに関しては必要な箇所でその差異についてのみ説明する．

6．1　制御記憶とシーケンサ

制御記憶はTable.2のように64bit／語（うち実際の命令には46bitを使用し，残りはデバッグ及び各種のデータ観測用），64K語となっている．命令の実行シーケンスは１レベルのパイプラインとなっており，プロトタイプでは100～250*3nS/Cycleで実行される．1マシンサイクル／命令で各命令は実行されるため，この数値の逆数がそのままMIPS値となる．プロセッサ内部の割り込み処理はマイクロプログラムレベルで行なわれる．

脚注３．　ほとんどの命令は100～150nSの範囲で実行され，この範囲を越えるものはごく特殊なもののみである．

項目	spec.
ALU	74AS888×2
内部レジスタ	16bit，最大128 （S_busは最大32個）
制御記憶	64K語
制御記憶語長	48bit+18bit(for future use)
マシンサイクル時間	100ns～250ns
主記憶，I/O インターフェース	VMEバス
割り込み	マイクロプログラムレベル ／8レベル

Table.2　Spec. of Proteus

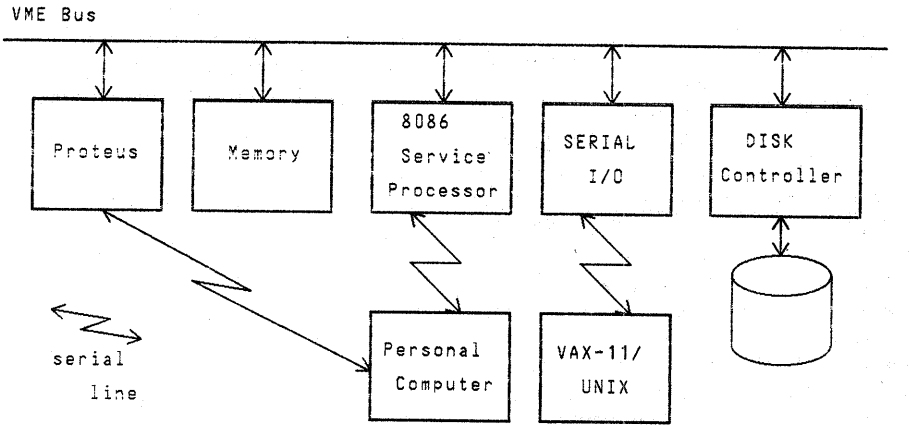


Fig.7　Proteus　システム構成図

プログラムからのシーケンサの制御は，マイクロプログラム制御の計算機にしばしば見られるように，シーケンサ制御のための専用のフィールドを割り当てることはあえて行なっていない．これは，予め組み込まれたExtension Unitとしてシーケンサを設計していることと，命令のフォーマットが高度の垂直水平型となっていることによる．

６．２　ＲＡＬＵ

汎用のアキュームレータとしてのレジスタファイルとALUには，74AS888　8bit sliceを用いている．74AS888のレジスタファイル及びALUの駆動方式もExtension Unitと同様の制御を行ない予め組み込まれたExtension Unitとしており，アーキテクチャの一貫性をもたせている．

６．３　VMEバスインターフェース

VMEバスとのインターフェースは，データ転送バス・インターフェースとVMEバス上の優先割り込みバス・インターフェースからなる．

データ転送バス・インターフェースの最も特徴的な点は，メモリレジスタがMADR(Memory Address/Data Register)として構成されている点である．MADRはELISプロセッサ[5]のMGR(Memory General Register)の成功にヒントを得たものである．しかし，MGRはハードウェア量も多いことから，機能をMAR(Memory Address Register)/MDR(Memory Data Register)の共用にとどめている．

優先割り込みバス・インターフェースは，VMEバスを通じてProteusと接続されている入出力装置からの割り込みを優先順位をつけて受け付け，Proteus内部に割り込みを発生する機能をもつ．

VMEバス・インターフェースは現在最終的な設計を行っている段階であり，Type DのExtension Unitとして動作する．

７　まとめと今後の課題

以上，ハードウェアの拡張によって，専用機を構築することを目的とする計算機のアーキテクチャ，及びそのプロトタイプについて述べた．

ハードウェアを付加する方法として，実現が容易なExtension Unitと並列性を考慮したExtension Blockの２通りの方法を示した．Extension Unitの形態を整理することによって，一応の標準化が図られたと思う．

試作機Proteusについては，現在，製作を行っている段階であり，一号機の具体的なインプリメント対象としてCommon　Lispマシンを決定している．Lispのインプリメントの際，重要なガーベジコレクションについてExtension　Blockを用いた並列アルゴリズムなどの検討も併せて行っている．これについてはシミュレーションの結果を評価の上，別の機会に述べたい．

又，従来の専用計算機で用いられている専用ハードウェアをPuroteusで実現する方法については，いくつかの例について検討を加え実現可能であるとの結論を得ているが[4]，拡張されるハードウェアの形態は，本研究で示した方法で十分なのかについては，まだまだ検証が不十分である．

謝辞

本稿作成にあたり，ご協力頂きました大江和久氏を始めとする小原研究室の皆様に深く感謝いたします．

参考文献

[１] 西田　明宏，朝長　宜央，山田　剛，小原　啓義，”マイクロプログラマブルプロセッサ“Proteus”のアーキテクチャとハードウェア構成”，情報処理学会第32回全国大会予稿集5S-5，Mar. 1986，pp.261-262.

[２] 村田　浩樹，朝長　宜央，山田　剛，小原　啓義，“マイクロプログラマブルプロセッサ“Proteus”のアーキテクチャとハードウェア構成”，情報処理学会第34回全国大会予稿集7P-7，Mar. 1987

[３] “SN74AS888，SN74AS890 Bit-Slice Processor User's Guide”，Texas Instruments Corp. 1985.

[４] 朝長　宜央，”ハードウエ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に関する研究”，早稲田大学理工学部電子通信学科　昭和６１年度修士論文Feb. 1987，

[５] 日比野　靖，渡辺　和文，大里　延康，“LispマシンELISのアーキテクチャメモリレジスタの汎用化とその効果—”，情報処理学会記号処理研究会，Jun.16,1983，pp.1-8.

盛光印刷所