第２章　高性能計算機アーキテクチャ

§2.1　緒言

近年のワークステーションの普及は、その高性能化によるものが大きい。初期のワークステーションでは、汎用のCISCマイクロプロセッサが主に採用されており、処理能力は数MIPS(Million Instructions Per Second)というものであった。80年代後半にRISCプロセッサが誕生すると、その処理能力は飛躍的に向上し、命令の互換性の壁をものともせず市場の多くを占めるに至った。また、プロセッサの動作周波数は年々上昇し、近年では数百メガヘルツにまで達している。

しかしながら、RISCプロセッサにより1クロック1命令の実行能力は達成できたが、1命令のみを逐次的に処理する方法ではこれ以上の性能向上は困難になりつつある。そこで、高性能計算機アーキテクチャとして、同時に動作可能な命令を並列に処理し、クロックあたりの処理命令数を1以下にすることにより処理性能を向上させる命令レベル並列処理方式が広く一般に使用されつつある。

本章では、まず計算機設計プロセスと関連づけた新しい計算機アーキテクチャの定義を示す。次に現在の高性能プロセッサの基本となっているRISCアーキテクチャについて述べた後、代表的な命令レベル並列処理方式であるスーパースカラ、VLIW、およびスーパーパイプラインについて、実例を示しながら述べる。また、主に数値計算処理において高性能を発揮するベクトル処理方式についても述べる。

§2.2　新しい計算機アーキテクチャの定義法

計算機アーキテクチャという言葉は、ＩＢＭ３６０システムに対し、はじめて使用された[Amdahl 64]。当時、それは計算機の構造を意味するものとして大変分かりやすいものと思われていた。それ以来、この言葉は計算機システムのハードウエアの構成を表すものとして、いたるところで使用されている。今日、計算機アーキテクチャという言葉は計算機の設計における基本的な概念や考え方としてとらえられている。

計算機アーキテクチャは大きく５つのレベルに分けて考えることができる。それらは、ゲートレベル、マイクロプログラムレベル、命令セットレベル、プロセッサレベル、ＯＳレベルであり、それぞれ計算機の機能レベルに対応している。普通、

単に計算機アーキテクチャと呼ぶときには命令セットアーキテクチャを指す。

ゲートレベルは物理的にみていちばん基本となるものである。ゲートレベルアーキテクチャによって様々な機能のゲートがデジタル回路を用いて実現される。

２番目のレベルはマイクロプログラムレベルである。このレベルの機能はマイクロプログラムそれ自体によって実現されている。このレベルはそのひとつ上のレベルである命令セットレベルときわめて密接な関係がある。

３番目のレベルは命令セットレベルである。単に計算機アーキテクチャと言うとき、普通はこの命令セットレベルを指す。なぜなら、全ての計算機は主にこのレベルのアーキテクチャでその構成が決定されるからである。

４番目のレベルはプロセッサレベルである。このレベルで考慮する構成要素はプロセッサ、メモリ、入出力装置、相互結合網である。プロセッサはメモリをアクセスし、また他のプロセッサと通信を行う。有名なFlynnの計算機システムの分類法[Flynn 66]はこのレベルに関するものである。

最上位のレベルはＯＳレベルである。ここでのアーキテクチャはアプリケーションプログラムからみたＯＳコール命令に関するものとなる。また、ＯＳそれ自体も含まれる。

機械工学や建築学における設計プロセスには、概念設計、基本設計、詳細設計、生産設計という４つのステップがある。このプロセスを計算機の場合について考えると、これらのステップは概念的な設計と論理的／物理的な構造の設計の二つのカテゴリに大別することができる。

計算機アーキテクチャの定義においては、まず、計算機の設計プロセスにおける考え方と関連づけて考えることが必要である。まず、計算機アーキテクチャは計算機の概念設計における概念に対応するものと考えられる。機械工学や建築学における設計プロセスによれば、概念設計とは機能の要求を満たすような動作や構造の概念を決定することである。そして、論理的／物理的構造の設計は基本設計、詳細設計、生産設計からなるものと考えられる。

本研究では、計算機アーキテクチャは、計算機の設計プロセスの中の計算機の概念設計における概念であると定義する。従って、設計する計算機が持つべき機能の要求を満たすような動作及び構造の概念を決定することが概念設計となる。この考え方を用いれば、これまで曖昧に使われていた計算機アーキテクチャの持つ意味が

明確になる。

表2-1に、これまでに示した計算機アーキテクチャと計算機設計プロセスの関係をまとめたものを示す。表中の左側が各レベルにおけるアーキテクチャ、右側が各レベルにおける設計プロセスとその内容を示している。また、図2-1に計算機設計プロセスの進行を図式化したものを示す。上位のレベルから各プロセスを順に螺旋状に経て下位のレベルまで詳細化していく過程が示されている。

表2-1　計算機アーキテクチャと計算機設計プロセス

<table>
<tr><th colspan="2">COMPUTER ARCHITECTURE</th><th colspan="4">DESIGN PROCESS</th></tr>
<tr><td>Architecture Level</td><td>Architecture
(The concept itself of conceptural design of computers)</td><td>Conceptual Design For Architecture</td><td>Basic Design</td><td>Detailed Design</td><td>Production Design</td></tr>
<tr><td rowspan="2">OS Level</td><td rowspan="2">Behavioral/Structural concept of OS call instructions and machine instructions</td><td rowspan="2">Decision of OS level architecture</td><td colspan="3">Logical Structure(Software)</td></tr>
<tr><td>Total system of OS</td><td>Relations between OS call instructions and OS</td><td>Details in programming</td></tr>
<tr><td rowspan="2">Processor Level</td><td rowspan="2">Behavioral/Structural concept of processors</td><td rowspan="2">Decision of processor level architecture</td><td colspan="3">Physical Structure(Hardware)</td></tr>
<tr><td>Computer system design</td><td>CPUs, memories, I/O devices, Interconnection Networks</td><td>To basic design of instruction set level</td></tr>
<tr><td>Instruction Set Level</td><td rowspan="2">Behavioral/Structural concept for Instruction set level
(Also for microinstruction)</td><td rowspan="2">Decision of instruction set level architecture</td><td colspan="3">Logical/Physical Structure(Software-Firmware/Hardware)</td></tr>
<tr><td>Including Micro-program Level</td><td>Processor design (Instruction sets etc.)</td><td>Microprogrammed (Hardwired)</td><td>To basic design of register transfer level/gate level</td></tr>
<tr><td rowspan="2">Register Transfer Level</td><td rowspan="2">Behavioral/Structural concept of register transfer</td><td rowspan="2">Decision of register transfer level architecture</td><td colspan="3">Physical Structure(Hardware)</td></tr>
<tr><td>Word gates, Registers, Multiplexers, Decoders, Encoders, Arithmetic elements etc.</td><td>To basic design of gate level</td><td></td></tr>
<tr><td rowspan="2">Gate Level</td><td rowspan="2">Behavioral/Structural concept of gate</td><td rowspan="2">Decision of gate level architecture</td><td colspan="3">Physical Structure(Hardware)</td></tr>
<tr><td>Combinational/ sequential circuits</td><td>Semiconductor devices
Transistors
Registers
Diodes</td><td>VLSI design</td></tr>
</table>

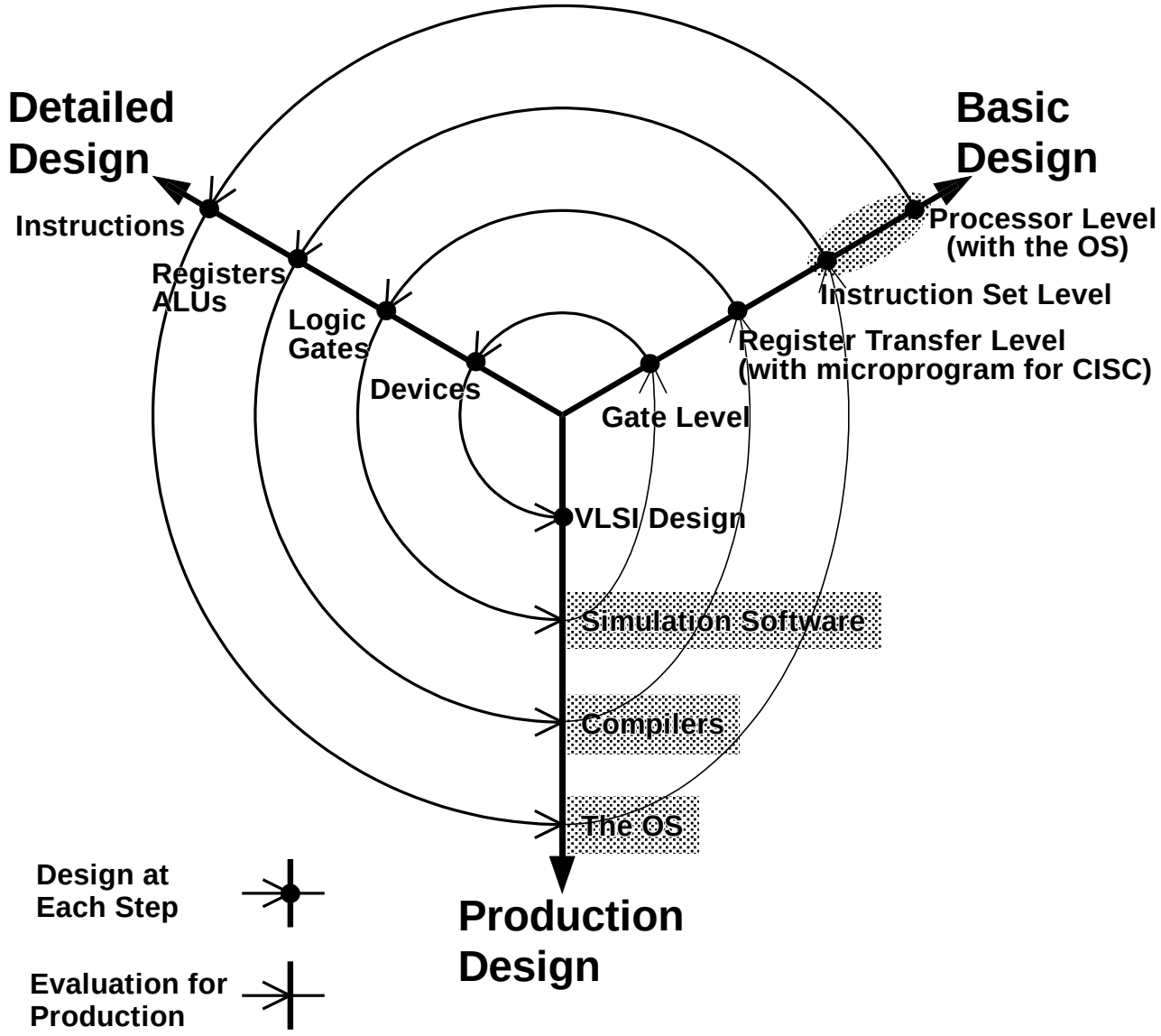


図2-1　計算機の設計における基本設計から生産設計に至るプロセス

§2.3　RISCアーキテクチャ

以前、コンパイラ等のソフトウエア技術が現在ほど進歩していなかった頃は、CやFortran等の高級言語のセマンティクスと機械語とのギャップを埋めるべく、より高機能な機械語命令を用意しようとしていた時代があった。このような考え方に基づいて設計された計算機は、配列アクセス等にも柔軟に対応可能な多くのアドレッシングモードを持ち、１命令で手続き呼び出し等をサポートするなどの高機能命令を備えていた。これらの計算機の命令は一般に可変長でかつオペランドの種類も多く存在し、そのデコードは複雑化した。また、演算のオペランドにメモリ内の値を使用したりするために実行に多数のクロックを要することが多かった。さらに、複雑な命令の実行のための制御回路は非常に複雑になりがちで、その設計には多くの時間と労力を費やすこととなった。そして、実行制御にマイクロプログラムが用いられたのもこれらの計算機に多い。

しかし、このような計算機システムにおいて、必ずしも高機能命令が実際のプログラム中で使用されているわけではなく、実行時間の大半が基本的な命令の実行に費やされているという報告[Patterson 82]がなされ、その結果命令セットをこれらの基本的な命令に限定し、それらを高速に処理することによって全体として高性能化をめざすという考え方が現れた[Radin 83]。これがRISC(Reduced Instruction Set Computer)アーキテクチャである[Radin 83;Hennesy 81,84;Patterson 82,85]。なお、RISCアーキテクチャの研究者は従来の複雑な高機能命令を持つアーキテクチャのことを、CISC(Complex Instruction Set Computer)アーキテクチャと呼んで区別している。

RISCアーキテクチャはCISCアーキテクチャに比べて命令が基本的で単純な固定長（一般には32bit）の命令に限定されているため、そのデコードや実行に複雑な回路やマイクロプログラム制御等が不要になり、その結果クロック周波数を向上できる。また、図2-2に示すような実行処理のパイプライン化も容易であり、命令を各サイクル毎に実行可能とすることが容易である。その結果実行の高速化ができる。また、制御回路の占める面積が小さくてすむことから、それにより生じたチップ上のスペースを利用して多くのレジスタを持たせることも可能となる[Patterson 82,85]。

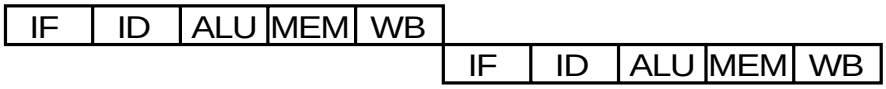
PattersonによるRISCアーキテクチャの一般的な定義を以下に示す[Patterson 82]。

・簡単な命令のみによる１クロック１命令実行
・32ビット固定長命令によるデコードの単純化

○典型的なRISCの命令実行プロセス

IF: **命令フェッチ（メモリからの命令読み出し）**
ID: **命令デコード（命令種類の解読、レジスタの読み出し）**
ALU: **演算（読み出した値の間で演算）**
MEM: **メモリアクセス（**LOAD/STORE**命令のみ）**
WB: **演算・ロード結果のレジスタへの書き込み**

○パイプライン化されていない場合



１命令あたり５サイクル必要

○パイプライン化した場合

１命令あたり１サイクルで実行

図2-2　**命令パイプライン処理の概念**

・メモリアクセスをロード、ストア命令のみに限定、演算はレジスタ間で行う
・高級言語とそのコンパイラの使用を前提とし、直接アセンブラによるプログラミングは行わない

RISCアーキテクチャの起源は、IBMで開発された801ミニコンピュータ[Radin 83]にさかのぼることができるが、カリフォルニア大バークレイ校にてPattersonらを中心に開発されたRISC-I/II[Patterson 82,85]や、スタンフォード(Stanford)大にてHennessyらを中心に開発されたスタンフォードMIPS[Hennesy 81,84]といった大学におけるRISCアーキテクチャCPUの開発プロジェクトによって広く世間に知られることとなった。

これらの２つの初期のRISC CPUは、基本的な思想は同一であるもののレジスタ構成においては大きく異なっている。

バークレイのRISCでは、プロシージャコールを高速に実行できるように、プロシージャ間での引数の受け渡しをサポートするためのレジスタウインドウ機能を持つレジスタファイルを採用した。その概念図を図2-3に示す。レジスタはR0～R137の計138個存在するが、一つのプロシージャから見えるレジスタは32個で、それらはR0～R9のGLOBALレジスタ、R10～R15のLOWレジスタ、R16～R25のLOCALレジスタ、R26～R31のHIGHレジスタに分けられている。GLOBALレジスタはすべてのプロシージャで共通して使用される。LOCALレジスタは各プロシージャの局所変数として用いられる。HIGHとLOWレジスタは呼び出し元と呼び出し先との間で共有されており、この部分を利用してプロシージャ間の引数を受け渡すことが可能である。現プロシージャがどのウインドウを使用しているかは、PSW(Program Status Word)内のCWP(Current Window Pointer)により示される。このレジスタウインドウを用いることにより、従来メモリ上に確保していたスタック領域を用いて引数を受け渡す場合と比較して、より高速にプロシージャコールを実行できるようになっている。

一方、スタンフォードMIPSでは、このようなレジスタウインドウは採用されていないが、16個の汎用レジスタを持っている。また、MIPSにおいてはハードウエアによるパイプラインインタロック機能は持たず、すべてソフトウエアによりインタロックが生じないようにスケジューリングされる。

これらの２つの初期のRISCアーキテクチャは、後に実用化され、それぞれSun

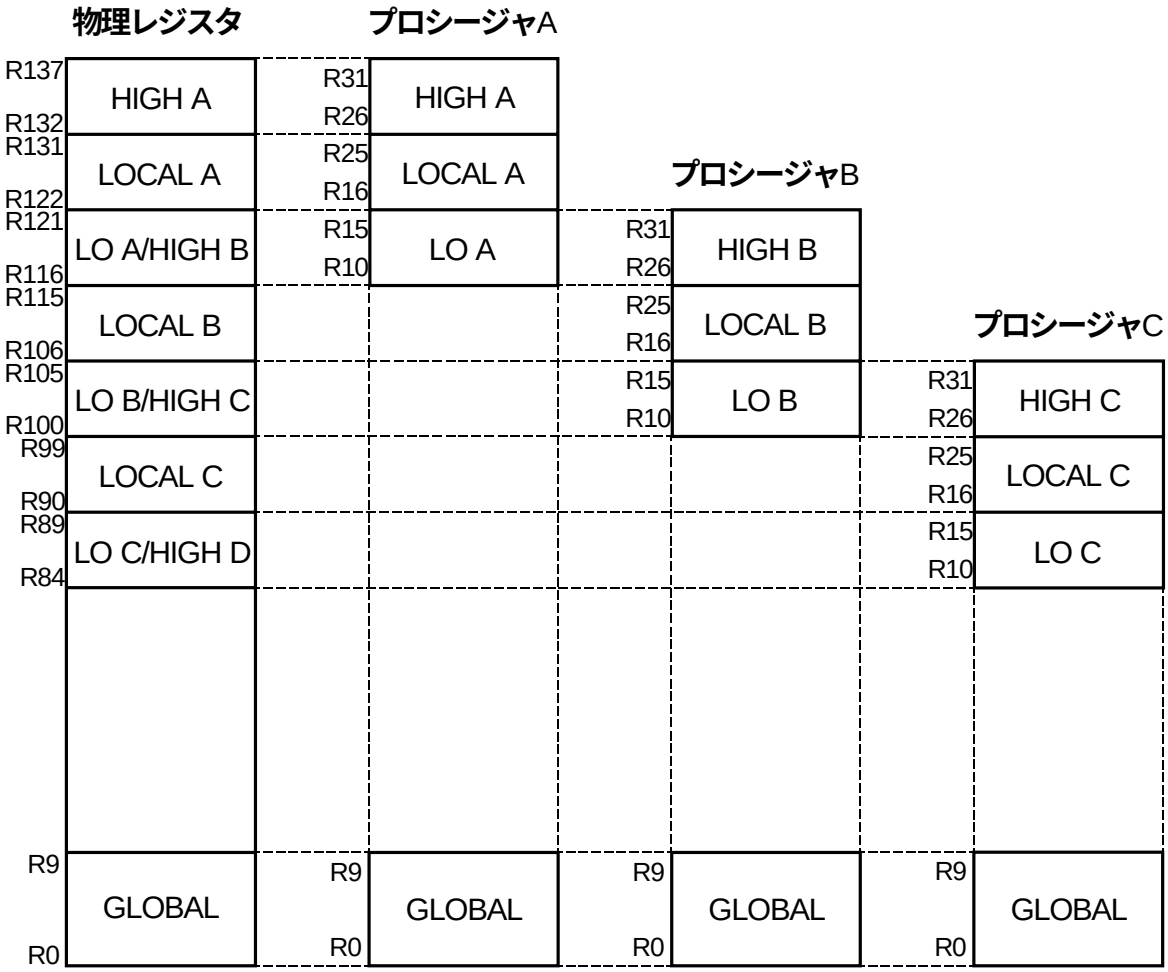


図2-3　**バークレイ**RISC**のレジスタウインドウ**

SPARCアーキテクチャとMIPS R2000/R3000アーキテクチャとなった。Sun SPARCアーキテクチャは、バークレイRISCのレジスタウインドウを採用しているが、HIGH,LOWレジスタはそれぞれin,outレジスタと呼ばれている。また、それぞれのレジスタ数も若干変更されており、global,in,local,outがそれぞれ８個づつとなっている。また、MIPS R2000/R3000では、レジスタ数は32個となっている。

これらのアーキテクチャによりRISCが一般的に使用され始めるにつれ、それまで各種のCISCアーキテクチャCPUを製作していた各社からもRISCアーキテクチャのCPUが発表されるようになった。代表的なものには、モトローラのM88100、インテルの80960/80860、AMDのAm29000などがある。

§2.4　命令レベル並列処理アーキテクチャ

RISC CPUの登場によりCPU性能は飛躍的に向上したが、ユーザインタフェースの改善や応用プログラムの計算量の増大に伴うソフトウエア規模の拡大もとどまることを知らず、性能向上への要求は常に途絶えることはない。

あるプログラム中に含まれる命令数をN(個)、CPUのクロック周波数をf(Hz)、CPUの１命令あたりのクロックサイクル数をCPI(Clock per instruction)としたとき、そのプログラムのCPU時間は次式で表される[Patterson and Hennessy 90]。

$$CPUtime = N \times (1 / f) \times CPI \quad (sec.)$$

上の式より、同一のプログラムを実行する場合、CPU性能を向上させるにはクロック周波数fを増加するか、もしくはCPI数を減少させることが必要となることがわかる。しかしながら、シリコン技術を用いる限りこれ以上のクロック周波数fの上昇は限界があり、またたとえばガリウム砒素などの高速な素子技術を用いた場合には集積度の問題や周辺回路との接続および速度差の問題などの課題がある[Milutinovic 91]。従って、CPIを減少させることにより性能向上をはかる必要があるが、RISCアーキテクチャではパイプライン処理やキャッシュメモリ等の採用によりすでにCPIは1に近づいている。従って、これ以上の性能向上のためには、1サイクルに複数の命令を実行することによりCPIを減少可能な命令レベル並列処理アーキテクチャの採用が必要不可欠である。現在、各種の命令レベル並列処理アーキテクチャが存在し、それらに対する評価も数多く行われている[Jouppi 89;Hennessy

91;Stone 91;Johnson 91;Saitoh 94]。

本節では、代表的な命令レベル並列処理アーキテクチャについて紹介する。

2.4.1 スーパースカラ(Superscalar)

スーパースカラ・アーキテクチャは、各命令間の依存関係を検知するハードウエアを用いて、並列に動作可能な命令を選択し、複数の処理ユニットを制御して命令レベル並列処理を行う方式である。その概念図を図2-4に示す。もし、命令間の依存関係、もしくは命令自体は並列動作可能であるが演算ユニット数等の制限により並列実行が不可能な場合は、逐次的な実行が行われる。この方法では従来の命令セットとの互換性を保つことができるため、現在使用しているソフトウエアを再コンパイルせずに処理の高速化を行うことが可能である。しかしながら、命令間の依存関係を検知するハードウエアは非常に複雑かつ高コストになりがちである。また、並列動作させる命令をその処理に対応した演算ユニットに分配する回路も複雑になる。

現在実用化されている高性能プロセッサの場合、命令の互換性が非常に重要であるため、スーパースカラが主に高性能化の手法として使用されている。RISCベースのスーパースカラ・プロセッサは同時実行可能な命令数は２～３、演算ユニットは整数と浮動小数点の２つを持つものが多い[Saitoh 94]。また、従来より使用されてきたCISCタイプのマイクロプロセッサに対しても、スーパースカラ技術が応用されてきている。

現在、市販されているスーパースカラ・アーキテクチャのCPUを、表2-2に示す。これらのCPUは主に高性能ワークステーションのために用いられているが、Pentiumなど一部はパーソナルコンピュータにも使用され始めている。SuperSPARCやMC88110のようにRISCプロセッサベースのものが多いが、現在パーソナルコンピュータのCPUとして圧倒的なシェアを持つ86系のCPUのスーパースカラ版であるPentiumは、CISCプロセッサベースのスーパースカラの数少ない例である。これは、スーパースカラが従来のプログラムを再コンパイルすることなしに高速化することが可能であるという利点を生かしたよい例である。しかしながら、Pentiumの場合、同時に実行できる整数命令が限定されている点や、浮動小数点命令と整数命令とを並列に実行できないなどの制限がある[Seya 93]。

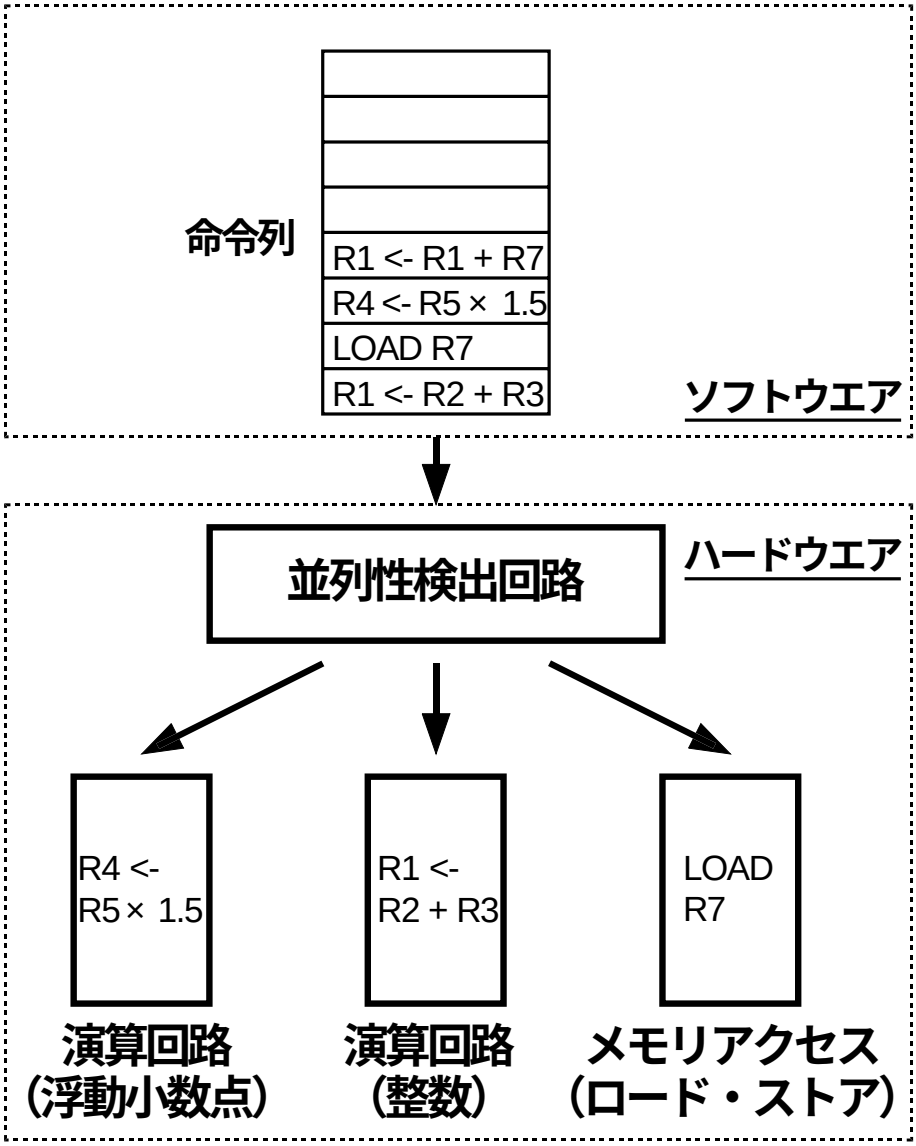


図2-4　**スーパースカラ・アーキテクチャの概念**

表2-2　商用スーパースカラCPUの例

開発元	Sun/TI	DEC	Motorola	IBM/Motorola	INTEL
製品名	SuperSPARC	Alpha	MC88110	PowerPC 601	Pentium
演算ユニット数	IU3,FPU1	IU1,FPU1	IU2,FPU1	IU1,FPU1	IU2,FPU1
最大並列実行命令数	3	2	2	2	2
内蔵キャッシュ構成	I:20KB D:16KB	I:8KB,D:8KB	I:8KB,D:8KB	ID共用8KB	I:8KB,D:8KB

2.4.2 VLIW

VLIWアーキテクチャは、多数の並列動作可能な処理ユニットを語長の非常に長い一命令によりそれぞれ独立かつ直接に制御することにより並列処理を行う方法である。その概念図を図2-5に示す。この長命令語(Very Long Instruction Word)がアーキテクチャ名の由来である。RISCが垂直型マイクロプログラムを命令セット化したものに例えられるのに対して、VLIWは水平型マイクロプログラムを命令にしたものと考えられる。命令語の語長は256ビットから1024ビットといったものがある。

このアーキテクチャでは、命令語の構成はプログラムのコンパイル時に行われる。ソースプログラム中の並列性はコンパイラにより検知され、並列に実行可能な演算は処理ユニットへの制御命令として長命令語の中に埋め込まれる。また、命令の実行順序まで考慮した命令語を生成するために、コンパイラはハードウエア構成に応じてそれ専用のものを使用する必要がある。従って、明らかにプログラムは再コンパイルされねばならず、従来との互換をとることは不可能であるため、広く使用されるには至らず、いくつかの実用化されたマシン[Colwell 88]も商業的には失敗に終わった。しかしながら、実行時に並列性を検出するための複雑なハードウエアが不要なため、スーパースカラに対して単純な構成をとることが可能である。

実用化されたVLIWの例として、TRACEアーキテクチャについて述べる[Colwell 88]。TRACEアーキテクチャでは図2-6に示すI boardとF boardのペアを単位として、それらが１組から４組の構成をとることが可能である。I-Fペアの数に応じた256～1024ビットの長さを持つ長命令語を用いて同時に7～28個の操作を制御して並列処理を行う。図2-7に、一つのI-Fペアを制御する256ビット長の命令語のフォーマットを示す。なお、長命令語によるコードサイズの増大をできるだけ抑えられるように、命令コードはメモリ上ではnopの部分が圧縮されており、命令キャッシュに格納されるときに展開される。ソフトウエアにおいては一般的なVLIWと同様に、トレーススケジューリング[Colwell 88;Ellis 86]等の手法を用いて通常のコードから並列性をできるだけ引き出し、長命令語に構成できるようにしている。

2.4.3 スーパーパイプライン(Superpipeline)

図2-8に示すようなスーパーパイプライン・アーキテクチャは、前述した２種のアーキテクチャとは若干異なり、命令パイプラインのピッチを細分化し、クロック周

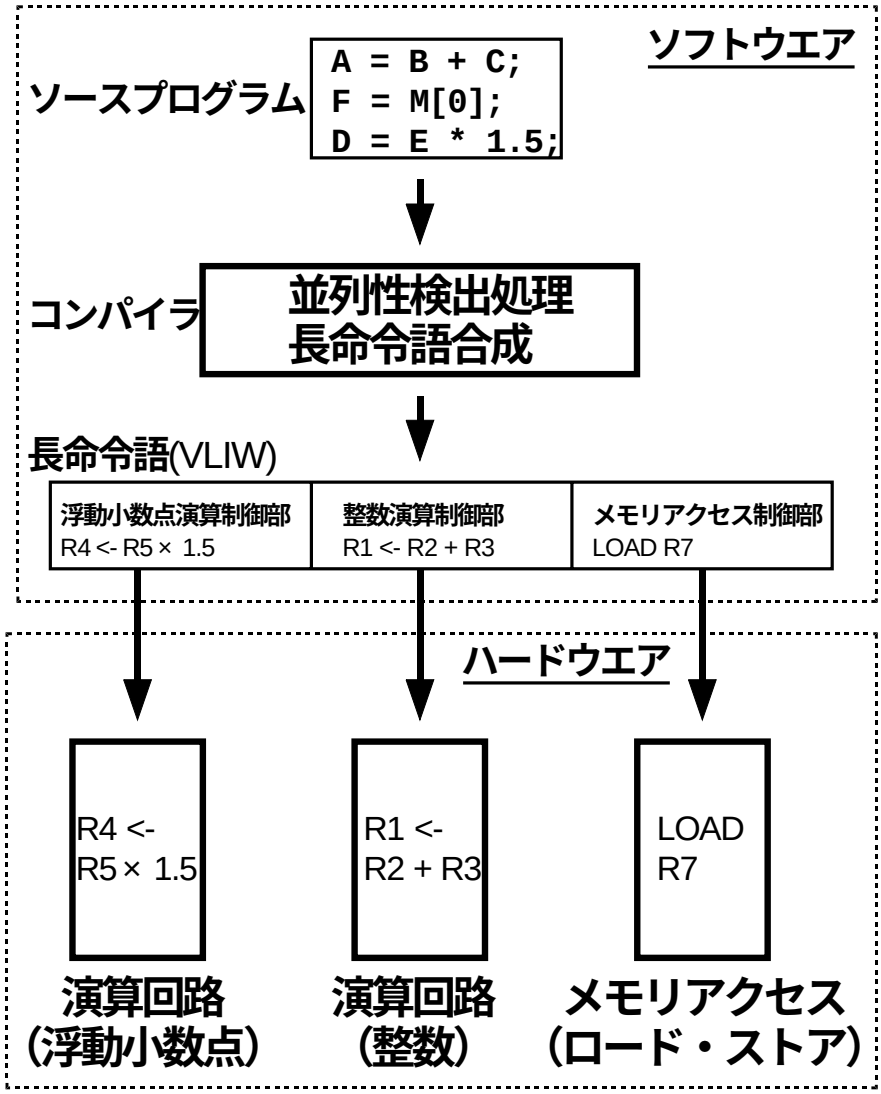


図2-5　VLIW**アーキテクチャの概念**

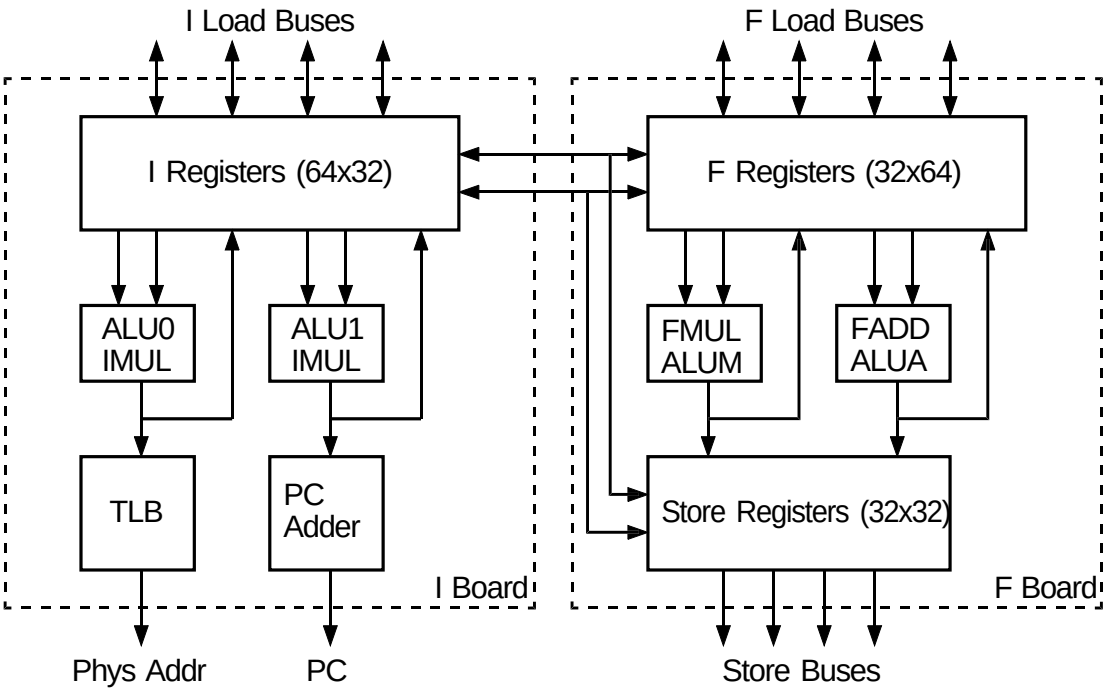
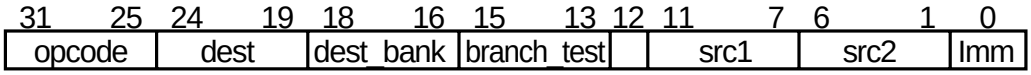
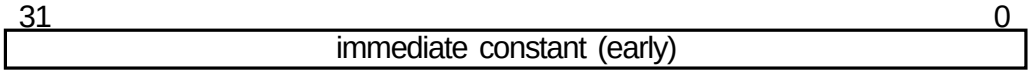


図2-6　TRACE**アーキテクチャの**I Board/F Board**のブロック図**

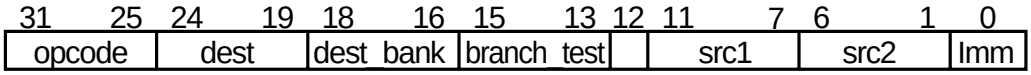
Word 0: I0 ALU0 early beat



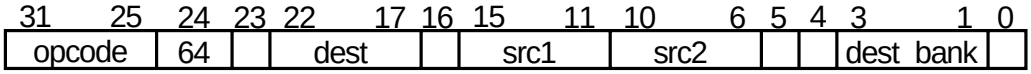
Word 1: Immediate constant 0 (early)



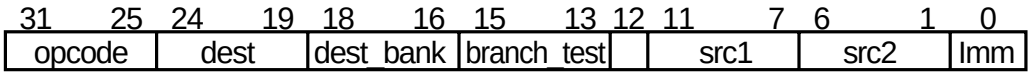
Word 2: I0 ALU1 early beat



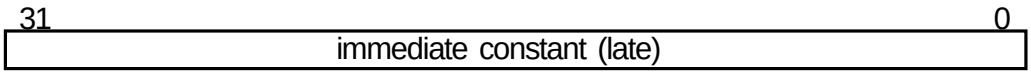
Word 3: F0 FA/ALUA control fields



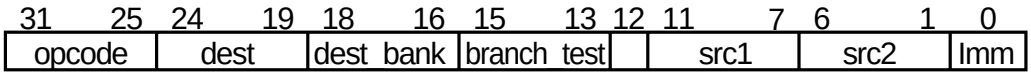
Word 4: I0 ALU0 late beat



Word 5: Immediate constant 0 (late)



Word 6: I0 ALU1 late beat



Word 7: F0 FM/ALUM control fields

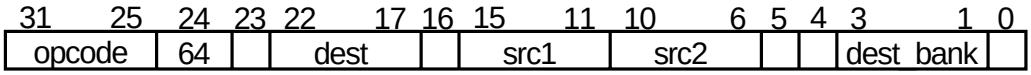
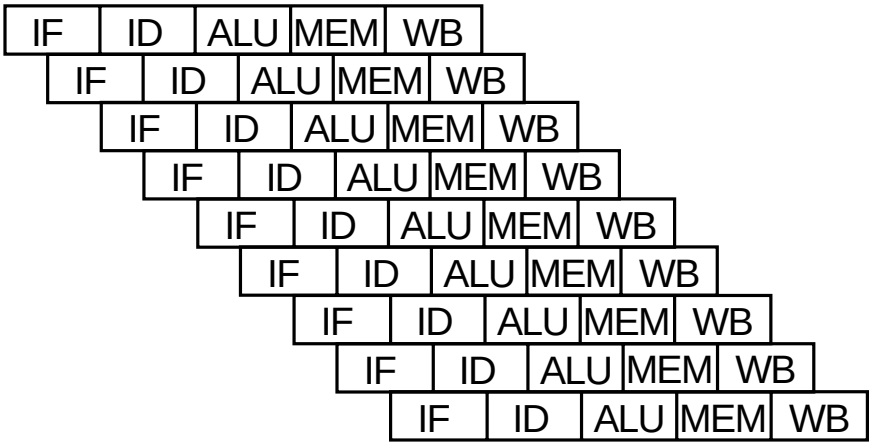


図2-7　TRACE**アーキテクチャの命令語フォーマット**(for one I-F pair)

○通常の命令パイプライン

○スーパーパイプライン方式



パイプラインのピッチを短縮
→　動作周波数の向上による高速化

図2-8　**スーパーパイプライン方式の概念**

期を短縮しかつより多くの命令を同時並列に処理しようとする方法である[Jouppi 89]。この方法では、他の２種よりもハードウエア量の増加を低く抑えながら、性能向上を図ることができる利点があるが、命令間の依存やブランチの際のペナルティが増加するために、たとえばパイプラインピッチを半分にしたからと言って２倍の性能が得られるわけではない。また、パイプライン各ステージ間のラッチ数の増加や、クロック周期の短縮に伴うクロックスキューといったハードウエア設計上の課題も存在する。

実用化されている代表的なスーパーパイプライン・アーキテクチャの例としては、MIPS R4000があげられる。その命令パイプラインの概念図を、図2-9に示す[Saitoh 94]。R4000の基礎となっているR3000のパイプラインは、図2-9(a)に示すような５段のパイプライン構成をとっているが、R4000では同図(b)に示すような８段の構成にパイプラインステージが細分化され、より多くの命令が同時に実行されるようになっている。なお、R4000とR3000の間にはバイナリレベルの互換性がある。

§2.5　ベクトル・アーキテクチャ

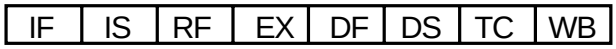
現在高速な数値演算を目的として実用化されているスーパーコンピュータは、ベクトル・アーキテクチャに基づいて設計されている[Karaki 84;Jippo 87;Nagashima 92]。また、マイクロプロセッサレベルにおいてもベクトル処理を目的としたアーキテクチャの研究・試作が一部で行われている[Imori 93][Watanabe 94]。

ベクトル・アーキテクチャを用いたスーパーコンピュータでは、パイプライン化された演算器に、各クロックサイクルごとにデータが送り込まれる。演算器は数段階のステージに分かれており、流れ作業的に処理が行われる。あるデータが演算器に入ってから結果が得られるまでにはステージの数だけ時間が必要であるが、見かけ上は各クロックサイクルごとに結果が得られることになる。計算機上での浮動小数点数の表現方法と、それに基づきパイプライン化された浮動小数点演算装置の概念図を、図2-10に示す[Hayes 88]。

パイプライン化された演算器に対してデータを効率よく送り込むために、スーパーコンピュータにはベクトルレジスタと呼ばれる一時記憶装置が装備されている。演算はベクトルレジスタの内容に対して実行され、結果もベクトルレジスタに格納される。ベクトルレジスタと主記憶間には数個のベクトルロード・ストアユニット

IF: Instruction Fetch
ID: Instruction Decode
EX: Execution
MEM: Memory Access
WB: Write Back

(a) R3000**のパイプライン**



IF: Instruction Fetch, First Half
IS: Instruction Fetch, Second Half
RF: Register Fetch
EX: Execution
DF: Data Fetch, First Half
DS: Data Fetch, Second Half
TC: Tag Check
WB: Write Back

(b) R4000**のパイプライン**

図2-9　**スーパーパイプラインプロセッサ** R4000**のパイプライン**

○浮動小数点数の表現

符号	S	指数部(Exponent) E	仮数部(Mantissa) M

浮動小数点数 $N = (-1)^S \times 2^{E-127} \times (1.M)$

(IEEE 754 standard 32bit format**の場合、**E:8bit, M:23bit)

○浮動小数点演算パイプライン（加算の例）

EC	ALI	ADD	NRM			
	EC	ALI	ADD	NRM		
		EC	ALI	ADD	NRM	
			EC	ALI	ADD	NRM

EC: **指数部比較**(Exponent Comparison)

ALI: **仮数部桁合わせ**(Mantissa alignment)

ADD: **仮数部加算**(Mantissa addition)

NRM: **正規化**(Normalization)

毎サイクル演算開始可能

図2-10　**浮動小数点パイプライン（加算の例）**

があり、演算と並列してデータの転送が可能になっている。このようなベクトルスーパーコンピュータの概念図を、図2-11に示す。ベクトルスーパーコンピュータにおいて、そのハードウエアを効率よく利用し、最大性能を引き出すためには、プログラムに対してベクトル化という処理を行い、プログラム中でベクトル演算命令を使用できる部分を抽出する必要がある。現在ベクトル化処理はコンパイラによって自動的に行われている[Yasumura 85;Nagashima 92]。

このようなベクトルレジスタとパイプライン化された演算器を利用して数値演算を高速に行うタイプのスーパーコンピュータの始まりはCray-1であり、これにより高性能が示された後は相次いでこのタイプのスーパーコンピュータが各社から発表され、その高性能化が競われることとなった。表2-3に現在の代表的なスーパーコンピュータの仕様を示す[Uchida 92]。いずれの機種もECLのような高速の素子を用い、数nsのサイクルタイムで動作している。また、大容量のデータを高速に入出力できるように、半導体による高速な拡張記憶装置を装備している。最近ではマルチプロセッサ並列処理システムの各処理要素にベクトルプロセッサを用いることによって、さらに高性能化をはかろうとしているアーキテクチャもある[Iwashita 94]。

ベクトル方式に基づく現在のスーパーコンピュータは、プログラムをベクトル化してパイプライン方式の演算器を効率よく動作させることにより高速な演算能力を得ているが、もしプログラム中でベクトル化できる部分の割合（ベクトル化率）が低い場合は、演算器が本来持つ性能を発揮することが不可能であるという問題点がある。

§2.6　結言

本節では、はじめに計算機アーキテクチャという言葉の定義について検討し、計算機アーキテクチャの各レベルや計算機設計プロセスとの関連を考慮に入れ考察した結果、計算機アーキテクチャとは計算機の概念設計における概念であるという新しい考え方を示した。次に、次章以降で提案する新たな命令レベル並列処理方式アーキテクチャの基礎となる、現在研究および実用化が行われている高性能計算機アーキテクチャについて、その基本となるRISCアーキテクチャと数種類の命令レベル並列処理方式アーキテクチャ、および数値演算処理用のベクトル・アーキテクチャについて、実例を挙げながら概観した。

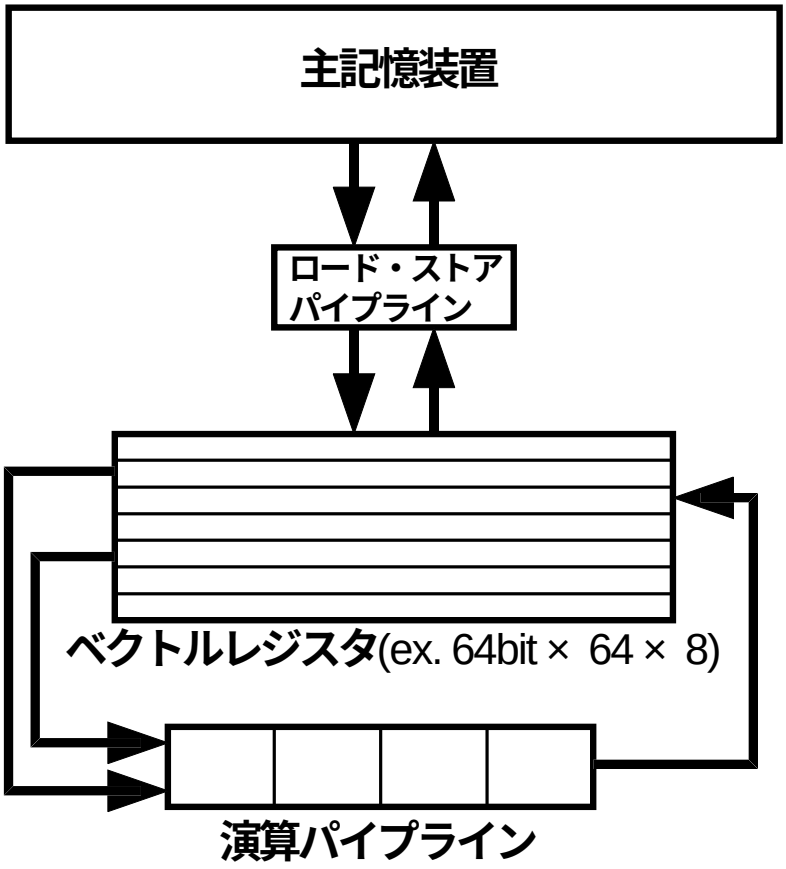


図2-11　ベクトル・アーキテクチャの概念

表2-3　代表的なスーパーコンピュータの仕様

開発元			Cray Research	日本電気	日立	富士通
製品名			Cray Y-MP8E/8256	SX-3 model44	S-820/80	VP2600/20
マシンサイクルタイム			6ns	2.9ns	4ns	3.2ns
処理装置	論理素子	素子タイプ	ECL	CML	ECL	ECL
		集積度(ゲート数)	2500	20000	2000/5000	15000
		ゲート遅延時間	300〜350ps	70ps	200/250ps	70ps
	メモリ&論理素子	集積度		40kbit+7000ゲート	7kbit+2500ゲート	64kbit+3500ゲート
		アクセスタイム		1.6ns	2.5ns	1.6ns
主記憶		メモリ素子	64kbitDRAM	256kbitSRAM	64kbitBiCMOS SRAM	1MbitSRAM
		アクセスタイム	15ns	20ns	20ns	35ns
		最大容量	1GB	2GB	0.5GB	2GB
拡張記憶(半導体記憶)		メモリ素子	DRAM	1MbitDRAM	1MbitDRAM	4MbitDRAM
		最大容量	16GB	16GB	12GB	32GB
		最大転送速度	5GB/s	2.75GB/s	2GB/s	2GB/s
冷却方式			伝導液冷(ふっ化炭素)	伝導液冷(水冷)	強制空冷	伝導液冷(水冷)