解　説

高性能 VLSI プロセッサに適した RISC アーキテクチャ†

高田　正日出††

1. はじめに

命令セットを使用頻度の高い簡単な命令のみに絞る RISC (Reduced Instruction Set Computer) 手法を用いたマイクロプロセッサが，既存の汎用プロセッサより高速処理が実現できるとして開発されている[1)-3)]．この RISC 手法が VLSI プロセッサのアーキテクチャとして主役の座に立つかどうかは不明であるが，現状のマイクロプロセッサの設計法に1つの警鐘を鳴らした点で評価されている[4)]．

一般に VLSI の集積度が増大するにつれて，システムの処理速度が高速になるとは言えない．処理速度を決める要因として次のようなものが考えられる．

高速化要因

（i） ビット数を増して並列・高速処理を実現する．

（ii） パイプライン処理によって速度を上げる．

（iii） レジスタ数を多くしてメモリに対するアクセス頻度を減らし高速化を行う．

（iv） 諸機能のオンチップ化によってチップ間の遅延時間をなくす．

低速化要因

（i） 配線長の増加によって遅延時間が長くなる．

（ii） ゲート数の増加によって一定電力の下では，ゲート当りのドライブ力は低下し動作速度は遅くなる．

（iii） 動作モードの複雑化のため処理速度の遅いクリティカルパスが生じ，サイクル時間を低下させる．

現状の VLSI プロセッサは上記の高速化要因の寄与によって高性能を実現している．しかし，今後更にマイクロプロセッサのアーキテクチャが複雑になるにつれて，ワンチップ化された VLSI システムが全体として高性能になるとは限らない．この対抗手法として登場したのが RISC アーキテクチャである[5)-7)]．

VLSI コンピュータに汎用コンピュータの命令セットをすべて採用すると，使用頻度の少ない命令によってシステムの処理速度が決定され性能が低下する．そこで RISC では，汎用プログラムの中でもっとも必要かつ使用頻度の高い命令セットを選び，これらの命令処理速度が最高速になるように，チップ内のデータパスとタイミングを設定して作られている．使用頻度の高い命令の処理時間を最小にしているため，汎用プログラムの実行時間も主にこれらの命令によって決定され，全体としても高速処理が実現される．

RISC アーキテクチャを用いたマイクロプロセッサは米国の大学を中心に研究されており，特に，カリフォルニア大学では RISC・CPU チップとして，RISCI と RISCII の2つのチップを開発した．これら両チップはレジスタセル及びデータパス構成が若干異なるが，命令処理に関する基本アーキテクチャは同一である．チップの完成度は RISCII の方が高く，命令処理速度も5倍以上 RISCII が速い．

本稿では，RISCII プロセッサ[2),5),8),9)]を中心にそのアーキテクチャ及び汎用プロセッサとの性能比較について解説する．

2. 命令セット[7),9)]

RISC II 命令はすべて1ワード32ビットの固定長であり，命令フォーマットが規則的であるため，簡単かつ高速の命令デコード・データ処理が行える．RISC II プロセッサに用いられた39種類の命令セットを表-1 に示す．命令セットは大きく分けて，レジスタ間演算命令，ロード命令，ストア命令，制御転送命令から成る．

カリフォルニア大学の RISC の特徴として，使用頻度の高いオペランドの高速処理を達成するために，大容量のレジスタファイルをチップに内蔵している．

† RISC Architecture for High-Performance VLSI Processor by Masahide TAKADA (ULTRA-LSI Research Laboratory, Micro-Electronics Laboratories, NEC Corporation).

†† 日本電気(株)マイクロエレクトロニクス研究所超高集積回路研究部

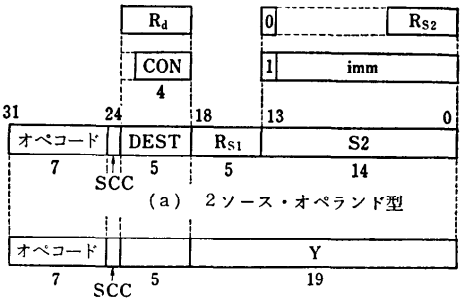
表-1 RISC Ⅱの命令セット

	命令	オペランド	処理
レジスタ間演算命令	ADD	R_d, R_{s1}, S2	$R_d \leftarrow R_{s1}+S2$
	ADDC	R_d, R_{s1}, S2	$R_d \leftarrow R_{s1}+S2+carry$
	SUB	R_d, R_{s1}, S2	$R_d \leftarrow R_{s1}-S2$
	SUBC	R_d, R_{s1}, S2	$R_d \leftarrow R_{s1}-S2-carry$
	SUBI	R_d, R_{s1}, S2	$R_d \leftarrow S2-R_{s1}$
	SUBCI	R_d, R_{s1}, S2	$R_d \leftarrow S2-R_{s1}-carry$
	AND	R_d, R_{s1}, S2	$R_d \leftarrow R_{s1} \wedge S2$
	OR	R_d, R_{s1}, S2	$R_d \leftarrow R_{s1} \vee S2$
	XOR	R_d, R_{s1}, S2	$R_d \leftarrow R_{s1} \oplus S2$
	SLL	R_d, R_{s1}, S2	$R_d \leftarrow R_{s1}$ shifted by S2 (左論理シフト)
	SRL	R_d, R_{s1}, S2	$R_d \leftarrow R_{s1}$ shifted by S2 (右論理シフト)
	SRA	R_d, R_{s1}, S2	$R_d \leftarrow R_{s1}$ shifted by S2 (右算術シフト)
ロード命令	LDXW	R_d, R_x, S2	$R_d \leftarrow M[R_x+S2]$ (32ビット)
	LDRW	R_d, Y	$R_d \leftarrow M[PC+Y]$ (32ビット)
	LDXHU	R_d, R_x, S2	$R_d \leftarrow M[R_x+S2]$ (16ビット符号なし)
	LDRHU	R_d, Y	$R_d \leftarrow M[PC+Y]$ (16ビット符号なし)
	LDXHS	R_d, R_x, S2	$R_d \leftarrow M[R_x+S2]$ (16ビット符号付き)
	LDRHS	R_d, Y	$R_d \leftarrow M[PC+Y]$ (16ビット符号付き)
	LDXBU	R_d, R_x, S2	$R_d \leftarrow M[R_x+S2]$ (8ビット符号なし)
	LDRBU	R_d, Y	$R_d \leftarrow M[PC+Y]$ (8ビット符号なし)
	LDXBS	R_d, R_x, S2	$R_d \leftarrow M[R_x+S2]$ (8ビット符号付き)
	LDRBS	R_d, Y	$R_d \leftarrow M[PC+Y]$ (8ビット符号付き)
ストア命令	STXW	R_d, R_x, S2	$M[R_x+S2] \leftarrow R_d$ (32ビット)
	STRW	R_d, Y	$M[PC+Y] \leftarrow R_d$ (32ビット)
	STXH	R_d, R_x, S2	$M[R_x+S2] \leftarrow R_d$ (16ビット)
	STRH	R_d, Y	$M[PC+Y] \leftarrow R_d$ (16ビット)
	STXB	R_d, R_x, S2	$M[R_x+S2] \leftarrow R_d$ (8ビット)
	STRB	R_d, Y	$M[PC+Y] \leftarrow R_d$ (8ビット)
制御転送命令	JMPX	CON, R_x, S2	$PC \leftarrow R_x+S2$ (CON が真なら)
	JMPR	CON, Y	$PC \leftarrow PC+Y$ (CON が真なら)
	CALLX	R_d, R_x, S2	$R_d \leftarrow PC$, $PC \leftarrow R_x+S2$, $CWP \leftarrow CWP-1$
	CALLR	R_d, Y	$R_d \leftarrow PC$, $PC \leftarrow PC+Y$, $CWP \leftarrow CWP-1$
	CALLI	R_d	$R_d \leftarrow LSTPC$, $CWP \leftarrow CWP-1$
	RET	CON, R_x, S2	$PC \leftarrow R_x+S2$, $CWP \leftarrow CWP+1$ (CON が真なら)
	RETI	CON, R_x, S2	$PC \leftarrow R_x+S2$, $CWP \leftarrow CWP+1$ (CON が真なら)
その他の命令	LDHI	R_d, Y	$R_d\langle 31:13\rangle \leftarrow Y$, $R_d\langle 12:0\rangle \leftarrow 0$
	GTLPC	R_d	$R_d \leftarrow LSTPC$
	GETPSW	R_d	$R_d \leftarrow PSW$
	PUTPSW	R_{s1}, S2	$PSW \leftarrow R_{s1}+S2$

R_d: デスティネーションレジスタ，またはストア時のソースレジスタ
R_{s1}: ソースレジスタ1
S2: ソースレジスタ2 (R_{s2}) またはイミーディエット定数 (imm)
R_x: インデックスレジスタ
Y: オフセット
PC: プログラムカウンタ
CWP: カレント・ウィンドウ・ポインタ
PSW: プログラム・ステータス・ワード

外部メモリ中のオペランドは，(i)レジスタへのフェッチ，(ii)処理，(iii)結果のメモリへのストアの3段階で処理されるが，パイプライン処理を用いて性能劣化を防いでいる．

RISC Ⅱはアドレスを32ビットまで取れるため，4Gバイトの仮想アドレス空間を有する．データは外部メモリ中でバイト・半ワード・ワードのいずれかの型でストアされるのに対し，レジスタファイルの中ではすべてワード単位でストアされる．以下に RISCⅡの命令セットの特徴を説明する．



(b) PC 相対型

図-1 RISCⅡ命令フォーマット

2.1 命令フォーマットの共通化

命令フォーマットを共通化することによって，命令コードのデコードを簡単かつ高速化できる．図-1 に基本となる2種類の命令フォーマットを示す．(a)は2つのソース・オペランドを有する命令に使い，(b)は PC (プログラムカウンタ)相対命令に用いる．プロシージャ当りの汎用レジスタとして32ワードのレジスタを使うため，オペランドのソース (R_{s1}, R_{s2}) とデスティネーション (R_d) アドレスに5ビットずつ使う．オペコードに7ビット使っているが，実際には39種類の命令しかない．SCC ビットは条件コード設定用に使う．DEST は分岐命令の場合には4ビットの条件コード (CON) を，その他の命令では R_d のアドレスを示す．ソース・オペランド S2 はレジスタ R_{s2} のアドレスまたはイミーディエット定数 imm として用いる．

2.2 レジスタ指向構成

コンピュータによるプログラム処理においてオペランドへのアクセスが処理の大半を占めるので，処理速度を上げるには高速のオペランドアクセスが必要である．このため RISC は CPU のレジスタ内に多くのオペランドを蓄えて高速処理させる．

RISC Ⅱ の基本命令は次に示す3オペランドのレジスタ間演算である．

$R_d \leftarrow R_{s1}$ OP S2

使用できる演算 OP は次のとおりである．

(i) 整数加算 (キャリ付きを含む.)
(ii) 整数減算 (〃)
(iii) 整数逆減算 (〃)
(iv) ビット単位のブール演算 (AND, OR, XOR)
(v) 左右論理シフト，右算術シフト (シフト数任意)

上記以外の汎用演算も次に示すように処理できる．

MOVE	$R_d \leftarrow R_s + R_0$
INCREMENT	ADD+1
DECREMENT	ADD−1
COMPLEMENT	$R_0 - R_s$
NOT	R_s XOR "−1"
CLEAR	$R_d \leftarrow R_0 + R_0$
COMPARE	$R_0 \leftarrow$ ·····
TEST	(命令実行時に条件コードをセットする.)

ただし，R_0 は常に0論理に結線されているレジスタを示す．

2.3 アドレス・モード

ロード・ストア命令はレジスタと外部メモリ間でオペランドを受け渡しする．

$R_d \leftrightarrow M[R_x + S2]$, または $M[PC + Y]$

外部メモリのアドレスは R_x (インデックスレジスタ) +S2, または PC+Y (オフセット) で与えられる．RISCⅡ で使用できるアドレスモードは次のとおりである．

アドレス・モード	実効アドレス
絶対または直接アドレス	R_0+S2
レジスタ間接アドレス	$R_x + R_0$
インデックスアドレス	R_x+S2
相対アドレス	PC+Y

RISCⅡ では外部メモリのアドレスはバイトアドレスを示す．他方，レジスタファイル中ではデータはワード単位で処理されるため，ロード命令に際して外部メモリのバイト及び半ワード・データはレジスタ内では LSB 側にシフトして蓄えられる．ストア命令に際しては，外部メモリシステムが4バイトワードの任意バイトを選択できると仮定している．

2.4 ディレイド分岐

図-2 にディレイド分岐処理を示す．分岐命令 I1 は常に1命令 (I2) 遅れて分岐先の命令 I3 を行う．命令 I2 は常に実行されるため，分岐命令 I1 の直前

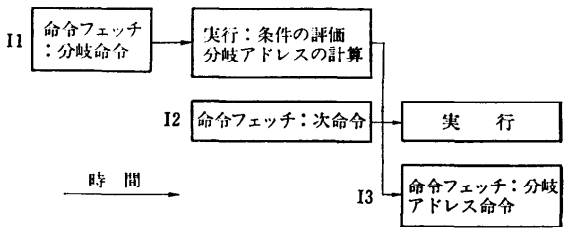


図-2 ディレイド分岐

のプログラム命令（分岐条件に影響を与えない場合），または NOP 命令（直前の命令が分岐条件に影響を与える場合）がコンパイラによって命令 I2 として与えられる．

2.5 高級命令の欠除

RISC は文字処理，乗算，浮動小数点演算のハード機能を備えていない．文字処理に関しては高級言語レベルで標準化されたフォーマットがないので除かれた．乗算，浮動小数点演算に関しては，CPU チップ内に集積化する場合にキャッシュメモリの集積化の方が利点が多いとしていずれも除かれた．むしろ，ファミリプロセッサとして別チップ化して性能の劣化を防ぐことを狙っている．

3. 多重ウィンドウ式レジスタファイル[7),10)]

RISC は高級言語によるプログラミングを前提としている．高級言語は階層化プログラミングが一般的であるのと，RISC 自体が複雑な命令をサブルーチン処理する発想のため，RISC ではプロシージャ呼び出しが非常に多い特徴がある．このプロシージャ呼び出しは一般にローカル変数をレジスタファイルと外部メモリ間で受け渡しするので長時間を要するが，RISC II では多重ウィンドウ式レジスタファイルを用いてコール・リターン処理を高速化する．

RISC II のレジスタファイルは全部で 138 個のレジスタから成っているが，1つのプロシージャ当り使えるレジスタ数は 32 個であり，これが1つのレジスタウィンドウを構成する．**図-3** に RISC II のレジスタファイル構成を示す．1つのレジスタウィンドウは 10 個のグローバルレジスタ（すべてのウィンドウ共通に使われる），10 個のローカルレジスタ，2つのオーバラップレジスタ群（2つのウィンドウに属する 6 個のレジスタ）から成る．1つのプロシージャに使用するレジスタ数が足らない場合には外部メモリを使う．オーバラップレジスタはコール・リターン命令に際して，レジスタ変数の受け渡しを自動化するのに用いる．

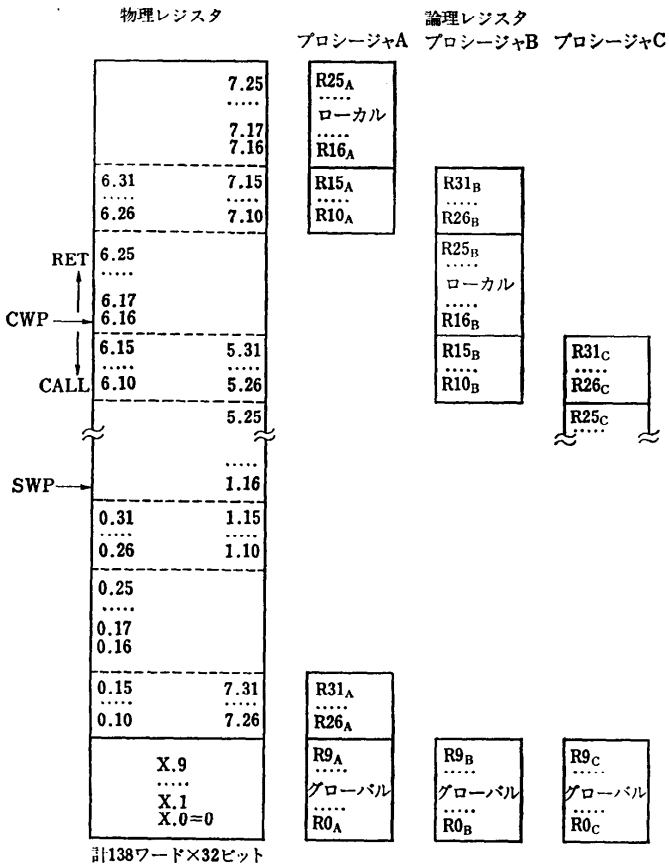


図-3 多重ウィンドウ式レジスタファイル

ウィンドウ内のレジスタ番号は命令コード中のレジスタアドレスによって決まるのに対し，ウィンドウ番号はプロシージャごとに1ずつ加減算される．ウィンドウは回転スタック構成で，隣接ウィンドウ同士がプロシージャのメイン・サブルーチン関係を作る．プロシージャのネスティングの深さは見かけ上無限である．実際にはネスティングが深くなりすぎて，レジスタファイルのウィンドウを全部使い切る（図-3 では，ウィンドウ数は 8 個ある）と，一番最初に蓄えられたウィンドウレジスタの内容は外部メモリに掃き出されて，レジスタファイル内に新しいウィンドウが作られる．ネスティングを解除する場合にはウィンドウを逆方向に順々に戻るが，レジスタファイル中のウィンドウを戻り切ると，先ほど外部メモリに掃き出されたレジスタウィンドウがレジスタファイル中に戻され，元の状態に戻ることになる．このレジスタファイルと外部メモリ間のウィン

ドウ転送は，CWP（アクセス中のウィンドウポインタ）と SWP（前回に外部メモリに蓄えられたウィンドウポインタ）を用いて制御する．

4. パイプライン構成

大部分の RISCⅡ 命令は次のパターンで実行される．

（ⅰ） R_{s1} と R_{s2} の読み出し，あるいは，PC または imm の取り込み．

（ⅱ） オペランドの演算処理．

（ⅲ） 結果の R_d への書き込み．

ロードとストア命令はデータメモリアクセスのためにさらに1ステップ増える．

図-4 に RISCⅡの基本パイプライン（a），及びメモリアクセス時のパイプライン（b）を示す．RISCⅡ は基本的には3段のオーバラップ処理を用い，（1）命令フェッチ，（2）命令の実行，（3）結果のレジスタへの書き込み，の各処理を1サイクルごとに行う．レジスタセルとして，nMOS スタティック RAM でよく用いられる6トランジスタ型の2パスメモリセルを用いている[11]ので，2つのソースオペランドの読み出しと前命令結果の書き込みを同時に行うことはできない．RISCⅡ のデータパスの動作を図-5 に示す．1サイクル内のレジスタファイルの動作は，（1）読み出し，（2）書き込み，（3）プリチャージの3ステップから成る．前命令の実行結果をオペランドとして用いる場合には，書き込み用データラッチからデータを引き出して用いる．この後レジスタへの書き込みが続いて行われる．ALU (Arithmetic Logical Unit) とシフタの演算は，レジスタファイルへの書き込みとプリチャージに並行して行う．1サイクル内の ALU，シフタの動作は（1）プリチャージ，（2）演算の2ステップから成る．

RISCⅡ は外部メモリとの情報転送に単一ポートのみを用いる．したがって，ロード・ストア命令の実行時にデータメモリのアクセスと命令フェッチを同時に行えないので，図-4（b）に示すようにパイプラインが一時的に中断する．もし，命令用にキャッシュメモリをオンチップ化できれば，メモリアクセスによる中断のないパイプライン方式が可能となる．しかし，RISCⅡ ではレジスタウィンドウ方式を使って，ロード・ストア命令の頻度を減らしているので，パイプラインの中断による性能劣化は小さいと考えられる．

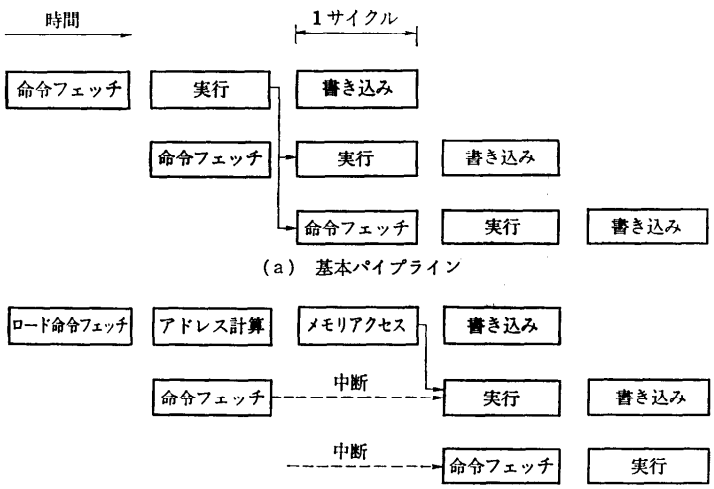


図-4 RISC Ⅱのパイプライン方式

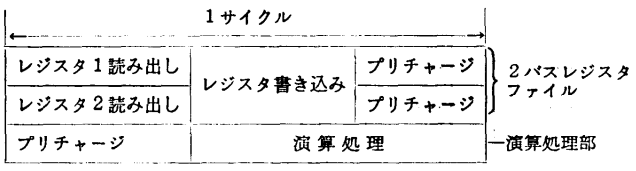


図-5 RISC Ⅱデータパスの動作

5. 汎用プロセッサとの比較

RISC プロセッサと汎用プロセッサとを，高級言語への適合性，命令コードのコンパクト性，性能について比較してみる．

5.1 高級言語への適合性

RISC 命令は一般のマイクロプログラムマシンのマイクロ命令と類似点が多い．

（ⅰ） すべての命令コードのビット幅は同一であり，各フィールドは決まったビットサイズを有し，かつ，その位置は固定されている．

（ⅱ） ロードとストア以外の命令の処理時間はすべて同一である．

（ⅲ） 命令はデータパス中で同一パターンで処理される．

（ⅳ） ディレイド分岐が使わ

れる.

(v) 命令コードのデコードが非常に簡単である.

したがって,1つの高級言語命令は一般に数個の RISC 命令にコンパイルされる.しかし,RISC を高級言語を用いてプログラミングしても,命令ステップ数は極端には増加しない.12種類の C プログラムの命令ステップ数が RISC と VAX-11/780 で,167対100になった例[12]がある.実行時間に関しては,個々の RISC 命令の処理が命令ステップ数の増加を打ち消すに十分,汎用プロセッサに比べて高速であればよく,2倍程度の高速化は容易に達成できる[13].

5.2 命令コードのコンパクト性

命令コードのビット幅の少ない命令フォーマットは,メモリ量が少なくて済む,命令デコードが高速になる,バス幅が少なくなる等,システムの高性能化に都合がよい.しかし,命令コードを少なくして性能を上げる方式にも限界がある.命令フェッチと実行は一般にパイプライン化されるので,命令実行時間よりフェッチを高速にしてもメリットはない.また,命令コード幅をバス幅より小さくしても,命令フェッチに常に1サイクル必要なのでメリットは少ない.これらを考慮して RISCII の命令フォーマットは1ワード32ビット幅に決められた.命令はメモリのワード番地にストアされるため,VAX-11/780 に比べてコードサイズは30%大きくなる.ただし,RISCII 命令のうちの7種の命令に半ワードコードを採用すると,コードサイズは30%程度節約でき,VAX-11/780 並みになることが確かめられている[14].

5.3 性能比較

一般にノイマン型コンピュータは高速回路技術と並列処理を組み合わせて高性能化される.並列処理のやり方として,(i)一つの命令を使用頻度の高い一連の基本処理(マイクロ命令)から作り,これら複数の基本処理を並列に実行させる,(ii)命令を1つの基本処理だけから成る簡単な命令とし,命令の実行に際してデータパスの大半がビジーになるように,命令処理をパイプライン化する,の2方式がある.(i)は CISC (Complex Instruction Set Computer) の設計法に,(ii)は RISC の設計法に対応する.CISC 方式は命令のフェッチ回数が少ない利点があるが,CPU 内に複雑な制御回路を必要とする欠点がある.

RISC 方式は並列処理に柔軟性があり制御も簡単であるが,命令が基本命令に分解されるので命令フェッチの回数が多い欠点がある.

表-2 RISC II と汎用プロセッサの処理速度の比較

プロセッサ	クロック周波数	レジスタ間加算演算時間	11種類のプログラムの平均実行時間(8 MHz RISC II で規格化)
RISC II	12 MHz	330 ns	0.67
RISC II	8 MHz	500 ns	1.00
VAX-11/780	5 MHz	400 ns	1.7±0.9
PDP-11/70	7.5 MHz	500 ns	2.1±1.2
M68000	10 MHz	400 ns	2.8±1.4
Z8002	6 MHz	700 ns	3.3±1.3

表-3 RISC II と汎用マイクロプロセッサの制御部占有面積の比較

	RISC II	M68000	Z8000	iAPX-43201
素子数	41K	68K	18K	110K
チップ面積 A (mm²)	59.6	44.6	38.6	66.3
制御部面積 B (mm²)	6.0	22.6	23.9	43.1
A/B(%)	10%	51%	62%	65%

RISCII と他の CISC プロセッサの処理速度の比較を表-2 に示す.ベンチマークとして用いた11種類のプログラムのうち,5種類はコール・リターン処理を含まない.また,RISCII のマシンサイクル時間 330 ns は試作結果の値である.RISCII の高速性はコール文を含まなくても成り立つが,特に,コール・リターン処理があると,性能は著しく向上する[9),15].

RISC と CISC の命令コードサイズの差はあまり大きくないのに対し,制御回路の複雑度と占有面積の差は非常に大きい.表-3 に RISCII と汎用マイクロプロセッサにおける制御部のチップ占有面積の比較を示す[16].例えば,M68000 の制御回路部はチップ面積の約50%を占めるのに対し,RISCII では10%以下である.RISC ではマイクロプログラム ROM よりは,むしろ命令キャッシュ RAM を優先する.命令キャッシュはダイナミックに使用頻度の高い命令を保持するのに対し,マイクロプログラム ROM はスタティックに使われる命令を保持し,時には使用頻度の低い命令を保持するため,チップ面積が有効利用されない欠点がある.

6. むすび

RISC アーキテクチャは次に示す一般的なプログラム処理の性質を前提としている.

(i) 整数型演算及びブール演算,プロシージャ呼び出し,比較及び分岐命令が非常に多い.

(ii) オペランドへのアクセスは非常に多いが，プロシージャ当りに使用されるローカル変数の個数は大半が1ケタである．

(iii) プロシージャのネスティングの深さは大半のプログラムで8以下である．

これらの性質への対処を次に述べる方法で行う．

(i) 簡単な命令のみを用いる．

(ii) 制御回路・データパスを簡単化し，マシンサイクルを高速化する．

(iii) データパス・ハードウェアの使用効率を高める．

(iv) 命令処理用の制御回路の減少で生じた空スペースに大容量の多重ウィンドウ式レジスタファイルを置く．

(v) プロシージャ呼び出しに際して，ウィンドウ間のオーバラップレジスタ部に受け渡しされるローカル変数を蓄え，コール・リターン処理を高速化する．

以上の方法で実現される RISC プロセッサは，一般に整数演算と高級言語プログラム処理を他の CISC プロセッサよりも高速に実行する．また，RISC ゆえの特徴として，VLSI チップの設計・レイアウトを早めると同時に，デバッグ量を減少させ VLSI アーキテクチャの TAT (Turn Around Time) を早めることも重要なメリットであろう．

RISC アーキテクチャの現状の問題点としては，

(i)命令セットの簡単化にともなうソフトウェアへの重荷の皺寄せ，(ii)浮動小数点演算，仮想記憶方式，マルチプロセッサ構成等の高級機能への未対処が上げられている．(i)に関しては，RISC はすべて高級言語を用いてプログラムを組むことを前提としており，流通ソフトの互換性を含めて高級言語と RISC 命令を効率よく結合するコンパイラの開発が望まれる．(ii)に関しては，RISC プロセッサに高級機能を備えた場合，RISC 特有の簡単かつ高速の制御系を実現できるかどうか今後の課題として残っている．

参 考 文 献

1) Radin, G.: The 801 Minicomputer, Proc. Symp. on Architectural Support for Programming Languages and Operating Systems, pp. 39-47 (1982).
2) Sherburne, R. W. et al.: A 32b NMOS Microprocessor with a Large Register File, ISSCC. Dig. Tech. Papers, pp. 168-169 (1984).
3) Rowen, C. et al.: A Pipelined 32 b NMOS Microprocessor, ISSCC. Dig. Tech. Papers, pp. 180-181 (1984).
4) 田中他：命令制御を単純化して，高性能マイクロプロセサを簡単に作る RISC 手法，日経エレクトロニクス，No. 308, pp. 123-142 (1983).
5) Patterson, D. A.: A RISCy Approach to Computer Design, Proc. COMPCON Spr. 82, pp. 8-14 (1982).
6) Hennessy, J. et al.: The MIPS Machine, Proc. COMPCON Spr. 82, pp. 2-7 (1982).
7) Patterson, D. A. and Sequin, C. H.: A VLSI RISC, IEEE Computer, Vol. 15, No. 9, pp. 8-21 (1982).
8) Katevenis, M. G. H. et al.: The RISC II Micro-Architecture, VLSI'83 Conf., Norway (1983).
9) Katevenis, M. G. H.: Reduced Instruction Set Computer Architectures for VLSI, Computer Science Technical Report No. UCB/CSD 83/141, U. C. Berkeley (1983).
10) Tamir, Y. and Sequin, C. H.: Strategies for Managing the Register File in RISC, IEEE Trans. Comput., Vol. C-32, No. 11 (1983).
11) Sherburne, R. W. et al.: Datapath Design for RISC, Proc. Conf. on Adv. Research in VLSI, pp. 53-62 (1982).
12) Ditzel. D. and Patterson, D. A.: Retrospective on High-Level Language Computer Architecture, Proc. 7 th Annual Symp. on Computer Architecture, pp. 97-104 (1980).
13) Patterson, D. A. et al.: RISC Assessment: A High-Level Language Experiment, Proc. 9th Annual Symp. on Computer Architecture, pp. 3-8 (1982).
14) Patterson, D. A. et al.: Architecture of a VLSI Instruction Cache, Proc. 10 th Symp. on Computer Architecture, pp. 108-116 (1983).
15) Patterson, D. A.: RISC WATCH, Computer Architecture News, Vol. 12, No. 1, pp. 11-19 (1984).
16) Fitzpatric, D. T. et al.: VLSI Implementation of a Reduced Instruction Set Computer, Proc. CMU Conf. on VLSI System and Computers (1981).

(昭和59年10月2日受付)