報　告

パネル討論会

RISC は CISC に勝るか

昭和63年後期第37回全国大会†報告

パネリスト
稲吉　秀夫[1]，富田　眞治[2]，日比野　靖[3]
平山　正治[4]，山本　昌弘[5]，**司会**　飯塚　肇[6]

みなさん，大変お暑いところ，本日は情報処理学会のパネル討論会にご参加いただきましてありがとうございました．当会の理事をつとめます NTT データの板倉でございます．ただいまから2時 45 分まで，標題にありますように，「RISC は CISC に勝るか」というテーマで，パネル討論をやっていただきます．司会をお務めいただきますのは成蹊大学の飯塚先生でいらっしゃいます．

それじゃ司会の飯塚さんにマイクをお渡ししたいと思います．先生よろしくお願いします．

司会(飯塚)　みなさん本日のパネルセッションにおいでいただきまして大変ありがとうございます．私，今ご紹介いただきました成蹊大学工学部の飯塚と言います．こういうことにはあまり慣れていないのですが，本日の司会を務めさせていただきます．どうぞご協力をいただきまして活発な討論が進めばいいと考えております．

それでは本日のパネリストですが，詳しいご紹介はおのおのお話をいただくときにいたしますけれども，まずお名前だけご紹介いたします．みなさんのほうから向かって右側から，日立製作所の稲吉さん，九州大学の富田先生，NTT の日比野さん，三菱電機中央研究所の平山さん，日本電気の山本さんです．

本日の予定としては，最初に私が簡単に趣旨などお話させていただきまして，その後パネリストに 15 分ほどご講演をいただきます．ご講演の内容は，おのおのの方の経験をふまえて，RISC と CISC，広くは命令セット・アーキテクチャについての意見を述べていただき，今日の討論の問題提起というようなことをしていただこうと思います．

そして最後に残った 45 分ないし1時間ぐらいは，フロアのみなさんからもいろいろご意見をいただきまして，パネリストとともに討論を進めていただこうというふうに考えております．どうぞよろしくお願いいたします．

CISC と RISC の誕生

それでは初めに，ご存知だと思いますが，RISC と CISC について簡単にお話させていただきます．最近 RISC チップといわれるものがいろいろ出てきて，RISC と CISC の議論が盛んになっております．この RISC にしても CISC にしても命令セットの話でありますので，まず命令セットの設計を歴史的に振り返ってみましょう．命令セットというのはご存知のように，最初はハードウェア主導で決められてきたわけですが，第三世代ごろからソフトウェアも考慮されるようになって，IBM の 360 に代表されるメインフレームの命令セットが完成しまして，一応一段落した形になっていたわけです．その後一時期互換性の問題があって，あまり大きな変化はなかったのですが，だんだんハードウェアが安くなってきまして，ハードウェアが安いのだったらもっとハードウェアをつぎ込んでソフトウェアとのギャップを減らしたらいいだろうという考えが，だんだん広まってきたわけです．その結果ある時期に命令セットが高機能のものになりまして，いわゆる CISC という非常に複雑な命令だけれども，ソフトウェアをサポートするような命令というのがかなり出てきたわけです．

それの代表的なものは，ご存知のように VAX の命令セットだというふうに言われています．ところが，こういう複雑な命令セットというのはどうも命令のデコードをするのに時間がかかるとか，あるいはデコードのロジックが大きくなって LSI にするときに，

† 日時　昭和63年9月14日 (水) 12：30～14：45
場所　立命館大学
1) 日立，2) 九大，3) NTT，4) 三菱，5) 日電，6) 成蹊大

非常にうまくないというような問題が出てきまして，命令セットを逆に簡単にしていったほうがかえっていいのではないかということで，非常に簡単な命令セット，つまりリデュースト・インストラクション・セットはどうだろうかというのが，1980年代になって出てきたわけです．最初は大学などで大いに研究が進められてきたわけですが，だんだんそれがなかなかいいのではないか，かなり作りやすいし，非常に高速だということが分かってきまして，近年ではいくつかのチップ・メーカからたくさんの RISC の商用チップが出てきたということになっております．

ご存知のように，SPARC だとか MIPS だとか AMD の 29000 だとかモトローラからも 88000 でしたか出ていますし，非常にたくさんの RISC のチップが出てきました．なんとなく最近の様子をみていますと，みんな RISC になってしまうのではないかというムードもあるわけです．しかし，一方ではやはり RISC というのは非常に単純だから，それだけソフトウェアの負担も大きいわけでありまして，将来もっともっと集積度が上がれば必ずしも RISC じゃなくて，やっぱり CISC のほうがいいのかなという意見もあるわけです．たぶん，今回こういうパネルをやるということになったのも，そのへんから両者の特徴を中心とした RISC, CISC の議論だけでなくて，将来命令セット・アーキテクチャがどうなっていくかというようなことについて，いろいろ討論をしたらいいだろうということで，計画がなされたのだろうと考えます．

私のコーディネートの立場でも，だいたいそういった趣旨でパネリストをお願いするということにしました．したがいまして今回お願いしましたパネリストは基本的には，命令セットのアーキテクチャあるいはチップ，そういったものについて実際に自分で設計をした経験がある方々を中心にお願いいたしました．ですから，必ずしもある方は RISC, RISC ということで，ある方は CISC, CISC ということで対抗するように選んだわけではありません．実際の経験に基づいてお話をしていただいて，本当に RISC がいいのか CISC がいいのか，あるいはその中間がいいのか分かりませんけれども，そういったことについていろいろ議論が進められたらいいと考えています．みなさん方も，おのおのご意見をおもちでしょうし，パネルディスカッションというのは，多少コントロバーシアルといいますか，ちょっと思い切った意見があるほうが面白くなるということもありますので積極的に議論に参加していただけたらよろしいかと思います．

それでは早速各パネリストに 15 分ほどずつお話をいただきたいと思います．それでは一番最初に，稲吉さんにお願いしたいと思います．稲吉さんは，現在日立製作所の武蔵工場マイコン設計部というところの副部長をなさっていらっしゃいまして，最近は有名なトロンのチップの設計に関わっていらっしゃいます．そういう意味でそのへんから命令セットについていろいろ問題点を提示いただけるのではないかと思います．では，お願いいたします．

CISC アーキテクチャを有効に用いた TRON チップ

稲吉 日立製作所の稲吉と申します．よろしくお願いします．

今，私どもの会社では，トロンのチップを出荷しております．これがそのチップでございます．CMOS の 1 ミクロン・プロセスというのを使いまして，13.5 ミリ角です．中を簡単にご説明いたします．ここが命令のデコードユニットです．外から取り込まれた命令は，この PLA のデコードユニットを通りまして，このマイクロ・プログラムの ROM にアクセスします．このチップのトランジスタ数は，全部で 73 万トランジスタなんですが 50 万トランジスタ分くらいはこのデコードユニットの PLA と，マイクロプログラムの ROM で使っています．

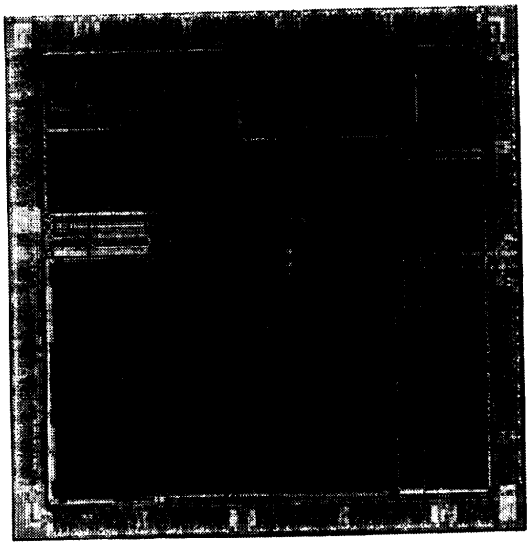

図-1

それから制御回路と実行ユニットがあります．このなかに演算器だとか，レジスタ類，バレルシフタなど入っています．それからこれはインストラクションのプリフェッチユニットでして，ここには命令のキャッシュだとか，あるいは分岐予測のテーブルだとか，プリフェッチアドレスのゼネレータだとかが入っています．

それから，ここが入出力のコントロールユニットでして，バスの IO の制御をしていまして，データキャッシュなんかもここにもっています．それからここにありますのがメモリ管理ユニットでして，アドレス変換のための TLB を内蔵しています．

本日の RISC, CISC のディスカッションに関係しそうな部分というのは，このデコードユニットとマイクロ・プログラム・エリアであると思います．それ以外の部分，命令のフェッチユニット，キャッシュや制御，こういったところは CISC, RISC のアーキテクチャには関係のないところですね．

それから入出力のユニット，データキャッシュ，メモリの管理ユニット，ここも関係ないところです．それからプロセッサの性能をどこが決めているかということなんですが，プロセッサの性能を決めているのは，ここの実行ユニットになりまして，この演算器をいかにウェイトサイクルなしに効率的に動かすかということが，プロセッサの性能が高いか低いかということを決めるわけです．ここも関係ありません．制御ブロックは，これらのコントロール・ポイントをいろいろ制御するため信号線を作っているところでして，ほとんど RISC, CISC には関係ないところです．このなかの半分ぐらいは例外のハンドリングのために使っております．したがいまして RISC, CISC にアーキテクチャ的に関係があるというところは，このマイクロ ROM，それから PLA のデコードユニットおよび若干の制御ブロックくらいかと思います．

トロンチップはご存知の方もみえてるかと思いますが，さきほど言われました VAX よりはるかに複雑な命令セットをもっております．簡単にご紹介いたしますと，命令のタイプとして3種類あります．プロセッサは 16 本の汎用レジスタをもってます．まず第1はレジスタ間の演算命令です．ソースとデスティネーションがレジスタの命令です．第2はレジスタとメモリの演算，これも1命令でサポートされています．RISC は，レジスタ間演算主体でメモリとのインタラクションというのは，ロード命令とストア命令だけです．トロンのほうはこの第2のメモリ・レジスタという形式がかなり多くのインストラクションでサポートされている上に，第3のメモリ・メモリ間演算という命令もサポートされています．これは，メモリ上の二つのオペランドを演算して，メモリにストアするという命令です．何がいいかといいますと，たとえば，メモリ上の二つのオペランドの加算の場合，サイズは違ってもいいんですが，たとえばバイトサイズとワードサイズの二つのオペランドを，一つの命令で加算します．これをバラバラにしますと，まずメモリオペランドをレジスタにロードしまして，それからサイン拡張あるいはアンサイン拡張し，それからデスティネーション・オペランドをロードして，加算して，最後にストアする．5つ命令が必要になるわけですね．これに対して，一つの命令で処理することができるということが良い点じゃないかと考えております．

それからさらにトロンのアーキテクチャで特徴的なものは，高機能命令です．まず，トロン系の OS で実際に使われる命令なんですが，キューの操作命令というのがあります．リソース管理などのためのキューの操作に用いられます．まず第1はある値のキーをもったエントリを探せという命令です．たとえばプライオリティ・ナンバ4番のエントリを探せという命令です．

それから，エントリをキューの中に入れたり出したりする，インサート命令，デリート命令という命令があります．これらはかなり高速にマイクロプログラムで実行されます．

それからストリング命令というのがあります．文字列などを扱う命令です．ストリングムーブ命令は，一つの命令でメモリ上のデータ・ブロックをデスティネーションに転送します．これはキャラクタであってもいいですし，それ以外のデータ・ブロックであってもいいです．Cなどでも構造体を表現するのに有効に使える命令です．さらにストリング命令系にはサーチ命令，それからストリング要素を入れなさいという命令，あるいはブロック同士をコンペアしなさいというような命令が用意されています．

普通こういう複雑な命令はコンパイラでは使われないから，あっても意味がないということになるわけなんですが，実際にこれらの命令は使われております．私どもで今開発しておりますコンパイラは，このストリング命令を使いましてライブラリ関数をインラインで展開しております．

この命令はどのくらい速いかという点ですが，もしこのストリングのサーチ命令がなければ，これだけの命令をループさせるわけです．このループをプロセッサのなかでマイクロプログラムで実行しますので，命令のバイト数，それから実行ステップ数とも圧倒的にストリングサーチ命令を使ったほうが有利です．これを RISC 的命令でやりますともう少しこれらの命令がバラバラになります．ロードしてレジスタのコンペアをしてロードしてというオペレーションが入りまして，それぞれ4バイト命令と仮定しますと，28バイトぐらいのコードサイズになります．

それからこのストリング系の命令のもう一つの特徴として，1バイトコード，2バイトコードをミックスして取り扱うことができるということがあります．文字列の表現として，中断コードというのを設定できます．たとえばA，B，それから漢字キャラクタ，さらにCという文字がありますと，これはこのように展開すればいいということになります．

このようにトロンのほうでは，コンパイラあるいは OS で使われない，あるいは使われても遅くなるというものは，高機能命令にはしておりません．実際に OS やコンパイラ中で使われて，かつ，単純な命令の組み合わせよりも速いというものを選び，マイクロプログラムで高速に実行するということをやっております．

それから，命令ばかりでなく，例外処理やメモリ管理も厳格に規定しています．これは例外ハンドリングのベクタのテーブル一覧表です．数多くのタイプの例外処理を区分けしておりまして，一つ一つ違うベクタをアサインしています．それから例外が起きたときのスタック上の情報としても，可能なかぎり多量のプロセッサ情報をダンプするようにしておりまして，たとえばメモリ・アクセスに関連する例外が起こりますと，この IO インフォメーションというところに，エラー情報をコード化して積みます．

さらにメモリ管理につきましても，多重仮想記憶をサポートし，2段階のページングシステムをとっています．それからおのおののページテーブルにはもちろんアクセス情報が入っておりまして，プロテクションもこれだけのタイプをページ単位に割りつけることができます．

私どもはトロンチップを最初に作って売り出しているわけなんですが，その立場から CISC としてのトロンとはこういうふうになっているということを中心にご紹介させていただきました．ありがとうございました．

司会 ありがとうございました．トロンのチップを実際に設計した立場からお話いただきました．簡単にまとめさせていただきますと，トロンのチップというのは CISC のタイプに属するかと思いますが，高機能の命令もやみくもに入れるのではなくて，適当な命令をうまくコンパイラで使えるようなものを選んでやればそれはそれなりに非常に有効であるというお話だったと思います．

それから，その高機能命令をインプリメントすることも現在のチップの技術だとマイクロ・プログラムを使って相当うまくできるということ，だいたいそんなところかと思います．ありがとうございました．

それでは，初めに申しあげるのを忘れたのですが，今日は企業の方にもいろいろパネリストをお願いしておりまして，今の稲吉さんも含めて，いろいろ企業でおやりになっているお話も出てくるかと思いますが，基本的にはこれは企業の意見ではなく，各パネリストの個人的な意見とご了解いただきたいと思います．そういうふうにお願いしてございますので，今日こういう話があったから，その会社がそうだろうというふうにはお考えにならないようにお願いします．その代わり，個人的な意見ということで，自由にいろいろ言っていただくというようにお願いしてございます．

それでは続きまして，次は九州大学大学院総合理工学研究科情報システム学の教授をしていらっしゃいます富田眞治先生にお願いします．富田先生はご存知のように，元京都大学におられまして，京都大学の時代に QA-1，QA-2 といったマイクロプログラムのマシンあるいは，もしかしたら RISC の一種の変型と言えるかどうか分かりませんが，VLIW，すなわちベリー・ロング・インストラクション・ワードマシンについて実際に設計あるいはソフトも作った経験をおもちで，またアーキテクチャ全般に詳しい方ですので，その経験をふまえて面白い話が伺えるかと思います．よろしくお願いします．

RISC と CISC の違いは何か

富田 今日のタイトルは，「RISC は CISC に勝るか」ということなんですが，一言でいいますと，大型機の立場から見ますと，“目くそ鼻くそを笑う

がごとし”ということになるんじゃなかろうかと思います．その理由の一つとして非常によく考えられた命令セットというようなものは高々 20〜30% の違いしかないだろうというふうに，私は思います．50% を超えますと，もうそういうプロセッサというのはなくなるというふうに，私は思っております．

それから，もう一つ，20〜30% の差というふうなところでありますと，OS とか IO とか，その他の要因がいろいろ効いてきて，もうプロセッサの単体性能というのは隠れてしまうのではないかというふうにも思います．

それから，プログラムの互換性の問題というのがあって，蓄積されたプログラムをどうしても使っていくという問題がありますので，こういった問題からも五十歩百歩だというふうにも思います．

その一つの例として IBM 370 アーキテクチャというのが幅をきかせているわけなんですが，その IBM の 370 が CISC であるとか RISC であるとか，どっちなんだというふうな議論というのは聞いたことがない．そういったことを尋ねられたこともないと思います．だが厳然として，IBM 370 アーキテクチャは，1964 年からずっと存在するということであります．

それからもう一つ，デバイス技術というのが非常に発展しているということで，それが一定しておれば，RISC, CISC といった議論というのができるんじゃないかというふうに思うんですが，なにぶんにも日進月歩，進歩が激しく追いつ抜かれつといったところで，まともな議論ができない面があります．

そういうことで，RISC は CISC に勝るかというふうな答として，私は目くそ鼻くそを笑うぐらい程度の差しかないんじゃないかというふうに思います．

さて，機械命令セットをみてみますと，直接実行型高級言語計算機の超々 CISC，間接実行型高級言語計算機の超 CISC，で iAPX 32 などの CISC, G-MICRO & MC 68030 などの CISC ふう，MC 88100 などの RISC ふう，SPARC などの RISC，学生実験作成計算機の超 RISC, Connection Machine の要素プロセッサのような超々 RISC など非常に広いスペクトラムがあります．

このようなスペクトラムのなかで，CISC と RISC というのをどこでどう分けたらいいのかといったような話があるかと思います．RISC マシンのもつ属性といったことで，機械命令の数が少ないとか，アドレッシングモードが非常に少ないとか，それからワン・サイクルで演算が全部終わるとかいったようなこと，それからロード・ストアのアーキテクチャであるとか，それから大容量のレジスタがあったり，さらにこれにウィンドウが付いていたり，高級言語を指向したりといったようなことがいろいろ言われておるんですが，どの RISC あるいは RISC ふうのものを見ても，こういったところを全部満足するものはないんですね．ないので RISC ふうといったようなことを言っていると思うんですが，そこでエイヤッと分類すると，命令セットの複雑さの尺度ということで CISC と RISC を分けるポイントというのを，とにかくマイクロ制御をやっているかどうかと，マイクロ制御が必要なほど命令セットが高いのか，マイクロ制御がいらないほど単純であるのかといったところで分けてしまったらどうかというふうに思います．

VLSI プロセッサでは，高速化を図るためには，有限チップのなかに何を入れたらいいかという話になるわけで，まず制御記憶を入れたもの，これが CISC であります．制御記憶の代わりに，たとえば大きなレジスタを入れる，あるいはレジスタの代わりにメモリのキャッシュあたりのところを入れる，あるいは浮動小数点演算器を入れるといった方式が RISC と言えます．

RISC というのは，もともとどのようにして生まれたのかと言いますと，もともとプロセッサの速度と主記憶の速度のバランスという面で考えてみますと，プロセッサの速度のほうが非常に速かった．そうすると主記憶のなかにかなりレベルの高い命令を入れておきまして，それを垂直型のマイクロでインタプリットするといったような感じになって，両者がバランスするため，CISC に向いていたわけです．

そうするうちに，非常にプロセッサ速度と主記憶の速度というのが，コンパラブルなオーダになってきたということで，機能レベルの高い命令を出してもプロセッサ側の垂直タイプ命令の実行速度が遅いものですから，主記憶のほうが非常に遊ぶことになるわけです．それで遊ぶのだったら，この垂直型のマイクロ命令を機械命令とみなして主記憶にインライン展開して用いることになったわけです．これが RISC です．制御記憶のオフチップ化の一形態なわけです．制御記憶はチップ面積の半分ぐらい占めますから，RISC では大きな領域が空くことになり，パターソンなんかがやった RISC I でのオーバラップレジスタウィンドウのようなものも実装できますし，モトローラ，インテ

ルの RISC のシステムのように浮動小数点演算器の実装などがオンチップで可能となります．今の技術ではこれらのもののオンチップ化でチップは満杯となっています．

チップに集積できるトランジスタ数が日進月歩増大していくので，RISC ではトランジスタを将来使い切れないのではないかといった危惧があります．これに対してはわれわれの研究室でやっている多重命令パイプライン方式でスケーラビリティを保ちながら十分対処できると考えています．

CISC のほうはどうかといいますと，現在の状況ではプロセッサの速度と主記憶の速度がコンパラブルのオーダですから，このメモリとプロセッサ系のバランスをとった設計をしたいということになりますと，どうしてもプロセッサのなかのマイクロ命令は水平型にならざるをえない．たとえば，この高機能命令が RISC 二つ分であるとしますと，マイクロ命令が二つ必要であるとしますと高速化はできないわけで，RISC 二つ分の命令を一つの水平型のマイクロのなかにうまく埋め込んでやるということができて，はじめて CISC というものが RISC よりも速くなるということになるわけです．また，水平型マイクロにするとピンネックでオフチップ化ができない．

ここで考えてみる必要がありますのは，CISC の命令でも高速にできないものがあるということです．CISC 操作のなかに本質的な逐次性があるようなもの，繰返し演算でデータパスが一つしかないもの，などがあると CISC の機能のレベルはきわめてレベルが高いのだけれども，それをマイクロのレベルで高速にインタプリテーションできないということになるわけです．

それからもう一つ，CISC 命令は本当に複雑なのかといった点があるかと思います．制御記憶というのは，チップの半分ぐらいを占めているというふうに言われているんですが，本当にチップの半分を占めさせるほど CISC の命令というのは，本当に高いレベルにあるのかといったことが，ちょっと疑問です．マイクロプログラム制御というのは，経験の集積でありまして，マシンの設計をするときに，昔のマイクロプログラム制御方式をそのまま引き継いでくるような傾向があります．そういう伝統のなかで，設計者はあまり考えないでマイクロ制御をやっているんじゃないかという気もします．最近 CAD というのが非常に進んでおりますから，もう 1 回原点にかえってみて，ワイヤード・ロジックで全部作ってみたらどうなるか．その上で RISC と CISC の比較なんかやってみたら，もうちょっと面白い議論ができるんじゃないかと思います．

昔の汎用型計算機なんかは，ワイヤード・ロジックで作っておりましたから，今日の CAD が進歩したところでかなり簡単にいけるんじゃないかといったような気も若干いたします．

それからもう一つ，やっぱり CISC というのはどうしてもやっぱり応用指向のほうにいかざるをえないんじゃなかろうかと思います．現在の機械命令セットは機能レベルが低いということで，マイクロあるいは演算系のところを生かし切っていないような気がちょっとします．ですから，応用指向の命令をユーザに自由に作らせるということではないんですが，もうちょっと開放して，命令を定義できるようなかっこうにして，マイクロを生かしきるような手段を考えたらどうかと思います．まあそういうことを考えますと，これは昔からありまして，東大の坂村健先生なんかがマシン・チューニングという話を若いときにやっておられましたが，そういった考え方をもう 1 回持ち込んで，応用指向の機械命令を定義できるようなかっこうにすべきじゃないか，それで高速化を図っていったらどうかというふうに思います．

そういう点で，機械命令セットとしては非常に汎用なものと非常に専用なものを共存させるといったようなことが，これからの CISC では必要じゃないかというふうに思います．

司会 ありがとうございました．

わざわざまとめる必要もないくらい整理していただきましたので，あとで議論するときいろいろ参考にしていただけるかと思います．

CISC や RISC も単に速いとか遅いとか批判されているけれども，そんな差はないのではないかとか，CISC は本当に複雑なのかとか，応用指向もいろいろ考えられるのではないかとか，いろんな面白い問題提起をいただいたかと思います．反論あるいは賛成の意見もあるかと思いますが，のちほどいただくことにいたしまして，次のプレゼンテーションに移らせていただきます．

CISC は RISC に勝てない？

次は，NTT ヒューマン・インタフェース研究所言語メディア研究部の研究グループリーダをされており

ます日比野靖さんにお願いします．日比野さんは，ELIS（エリス）というマシンの設計に携わっておられましたので，そのへんからいろいろお話を伺えるのではないかと思います．どうぞ．

日比野 今ご紹介がありましたように，私は NTT の研究所で ELIS という名前の LISP マシンをずっとやっております．実は昨年からそれを販売させていただいております．さきほど富田先生のほうから RISC，CISC の論争は，「目くそ鼻くそを笑うがごとし」というお話がございましたが，たとえば，私どものように商売をさせていただいている立場から言いますと，笑ってばかりもいられないことで，死活問題でございまして，まずそのへんの気持からお話させていただきたいと思います．

まず，高級言語マシンを開発しているものの立場からの気持を述べさせていただきます．

今日は RISC は CISC に勝るか，というテーマなんですけれども，私は，あれ！ちょっとテーマが違っているのではないかなと思いました．どういうふうに違っているかと申しますと，CISC はもう RISC に勝てないのではないかという気がしていたんです．そのくらい私どもの立場から申しますと，切実な問題がございます．

今日，お話したいことは三つあります．まず RISC があればというのか，もう RISC になってしまえば高級言語マシンは要らないんではないかというような見方があるのに対して，どう考えているかということです．

それから，2番目は，高級言語マシン，さきほどの富田先生のお話では超々 CISC，あるいは超 CISC というアプローチからみますと，CISC というのはなんとなく中途半端なのではないかということ．3番目は，さらにそれを先に進めまして，今 UNIX のワークステーションあるいはCという言語が非常に多く使われておりますけれども，こういった OS あるいは言語を前提とした RISC のブームでございますが，いつまでもそんなことでいいのだろうかということを，お話をさせていただきたいと思います．

まず RISC であれば高級言語マシンは要らないかと，もう少し具体的に言いますと，SUN ワークステーションがあれば LISP マシンは要らないかと，こういうことでありますと，どんなことになるのであろうか．このことは私どもの立場から言いますと，毎日のようにベンチマーク競争で勝った負けたと言って，一喜一憂しておりますので，なかなか大変なところであります．

まず RISC とそれから CISC という今日の話題なんですけれども，高級言語マシン（ハイレベルランゲージ・マシン）と RISC とを比較してみようと思います．マシンのなかのゲートをコントロールしているのは，実際どこがどういうもので動いているかという立場からみますと，RISC の場合は，これはいわゆるマシン命令ということでございまして，1サイクルに1操作が行われています．一方，高級言語マシンの場合では，さきほどの富田先生の定義によれば，超々 CISC あるいは CISC ということでございまして，マイクロプログラム制御によって行われるということで，1サイクルの間に複数の操作が行われるということであります．この複数分の効果をどう勘定するかということになります．マシンサイクルが同じであれば，だいたい同じぐらいの処理ができる，あるいは高級言語マシンのほうがやや有利なはずではないかということになります．

それからもう一つ，コントロールしている情報の所在です．これはさきほどの富田先生の整理では，どこに情報を置くかということになろうかと思います．RISC ではキャッシュに置いてあり，これが今オンチップになっております．普通のマイクロプロセッサでもキャッシュに置かれるようになってきています．一方，高級言語マシンの場合ですと制御メモリに置かれるということになると思います．

これが本格的な超高級言語マシンになりますと，まだ現在の技術では，オンチップにいくわけにはいかなくて，外付けになってしまうというところであります．

それから制御情報，すなわちプログラムをどうやって作るかという立場から申しますと，RISC については普通の高級言語のコンパイラということでございましょうし，高級言語マシンは言語レベルに合わせるための内部制御のプログラムを，マイクロプログラムで，人海戦術によって実現することになります．アセンブラで書く．あるいはマイクロコンパイラを使うということだろうと思いますが，なかなかこれを作るのは大変だということでございます．

それで，こういったものを今度性能をどうやって向上させていくかという立場からみますと，RISC につ

いては最適化の技術，最適化コンパイラをどうやって作るかという問題になろうかと思いますし，高級言語マシンの立場から言いますと，応用指向になっていくだろうということとも直接関係がありますが，主要な関数のマイクロコード化を行って，性能を上げていくことになりましょう．アプリケーションに応じてこういうことをやっていけば，アプリケーションの専用になるということにもなろうと思います．また，特定言語という立場からいえば，その言語のなかでよく使われるプリミティブを，マイクロ化していくということだろうと思います．

それで，こういった手法の適用性というか有効性といいますか，そういったものをみてみますと，RISC の場合には，やはりコンパイラの最適化手法ということで，普遍的な方法で効果が全体に及ぶという有利な点があろうと思います．高級言語マシンについては，チューニング手法ということで，部分的あるいは個別的になってしまうというところに差が出てくるということだろうと思います．

それで，こういった性能向上の手法に関しての難しさということに関して比較してみると，性質は異なりますが，難しさという点においては，やはり両方とも難しいであろう．逆に言いますと，同じ程度にやさしいとも言えるかもしれません．RISC の場合でも，最適化と申しましても，いろいろ情報によりますと必ずしもうたい文句どおりの MIPS 値はアプリケーションでは出ていないという話もございます．それから高級言語マシンの場合も，こういったチューニングをしようとしましても，人手によりましてアセンブラで書いているというようなことではうまくいきません．性能のいいマイクロコンパイラのようなものを作らなければならない．ところが，これはこれでまた難しい問題でございます．

そういう立場でこれらを比較しますと，原理的な問題といたしまして，ある意味で性能という面からいえばそう差が出てこないのではないかという気がいたします．

RISC の将来性は少ない？

2番目の話といたしまして，さきほど来，CISC というのがいったい何かということが問題になっておりますけれども，さきほどの富田先生のお話の分類で言えば，私がここで CISC といっているのは CISC ふうというのに当たるかと思いますが，今の CISC ふうのプロセッサというのは，なんとなく中途半端ではないだろうかという気がいたします．つまり，さらになんらかのことをすればより性能を上げられる可能性，あるいはメリットが出てくる部分があるのか．逆に今の CISC であるとすれば，RISC とあまり差がないのではないかというような意味合いでございます．

さきほど TRON のチップの話がございました．LSI のテクノロジの進歩の恩恵を享受する．アーキテクチャはこれを享受することによって，具体的なプロセッサ・デザインをするということになるわけです．みなさんよくご存知のように，スケール則というのがございまして，プロセスの技術が上がりまして，S 分の1に長さが縮まりますと，速さが S 倍になって，素子数は S^2 倍入ってくるという大変素晴しい性質を集積デバイスはもっています．これがハードがどんどん進歩しているのに，ソフトが進歩していない原因なのであります．このメリットをいったいどう使うかということだろうと思います．それで RISC の場合，これをレジスタファイル，RAM ですね．それからキャッシュメモリを増やすということでやってまいりました．さていったいその後はこれを何に使うのでしょうか．というのが一つの疑問であります．

一方，今の CISC でありますと，たしかにいろいろ複雑な命令があっても速くできるというお話もございました．素子数を増やして頑張ろうということであると思います．しかし，やはり RAM を増やし，キャッシュを増やしていくということで，その後どうするのかということになろうかと思います．

それでもう一つ，今度別の観点からみますと LSI 化されたプロセッサの性能のネックということを，さきほど GMICRO のお話のところでは，エグゼキューション・ユニットのところをどれだけ速く作るかでプロセッサの性能を決めている，というお話がございました．たしかにそれが一つの性能を決めておりますけれども，もう一つのネックは，チップに命令あるいはデータをどうやって供給するか，この供給するスループットをどうやって与えるかということがございます．これは一つはピンの数，あるいはそのピンをとおして入力しうるデータの速さ，最小どのくらいの時間で入れられるかということに関係しているわけであります．こういう面で RISC と CISC というのをみますと，データに関しては，これは操作対象のデータに関しては本質的には同一条件であるはずです．無駄な書き込みがないとすれば，本質的には同じである．

一方命令について言いますと，本来は CISC が有利なはずである．つまり，高機能な命令になればなるほど1命令によって，たくさんのデータを処理するということになるわけですから，命令のフェッチの回数が少なくなるということであります．本来はこうであるはずだということであります．ところが，さきほども富田先生のお話のなかに出てきたように，結局は待ちが生じてしまって，なかなか速くならないよという話があります．しかし，こういう性質があるということで，私が言いたかったのは，まず RISC についていえば，将来工夫の余地がもうあまりないのではないかということと，それからもう一つ CISC についても，やはりそれだけでは中途半端であって，もう一歩先に進む必要があるのではないだろうかという気がしているということであります．

それで，最後に表題にちょっと直接関係ないような感じもするのですが，いつまでも UNIX と C でいいかと，こういうことでございます．なぜ UNIX というようなことが突然出てきているかといいますと，今 RISC のチップというものがいろいろたくさん出ておりますけれども，いずれも，UNIX という，いわばパブリックなオペレーティングシステムがあるということが前提になっておりまして，それがあるためにシステムの開発が非常に簡単にできる．逆にいえばチップを作り，ハードウェアを作ってしまえば大きなソフトウェアは全部動いてしまうという利点がありますので，それによって，新しいアーキテクチャマシンがどんどん出てくるというベースがあるだろうということでございます．もう一つは言語という点からいいますと，Cというのは全盛になっておりますが，これは言語屋の立場から言いますと，あまりにもレベルが低いのではないかというような気がしております．

そのへんを高級言語マシンの立場からちょっと見取図的にみてみますと，このゲートレベル，これはどのやり方に対してもまったく同一レベルであるということに対しまして，いわゆる RISC というのはこのゲートを直接操作するレベルで命令が定義されているということで，言語レベルから最適化コンパイラで，ここに落としてやろうということであろうと思います．従来の計算機すなわち CISC というのは，ゲートレベルに対して，ある程度マイクロコードで命令をサポートしてやって，ここにコンパイラで落とすということであります．高級言語マシンの場合はさきほどの富田先生のお話では，超々 CISC に当たるのかもしれませんけれども，言語レベルのすぐ下に命令レベルがございまして，ここから下はドーンとマイクロコードでやろうということであります．こちらのアプローチをずっと進めていきますと，これをさらにずっと上に引き上げていけるのではないかという気がしております．

そういう意味で，RISC と CISC で論争しているというのではなくて，高級言語マシンのほうにいくのが本筋ではないだろうかということなんですけれども，どうも最近すっかり下火になっておりまして，なかなか残念でございます．以上でございます．

司会 ありがとうございました．今は高級言語マシンとそれから，とくに RISC との対応ということでお話いただきました．

今の高級言語マシンと RISC については，どちらもいろいろ難しいところがあって，性能的にはそんな差はないのではないか，ただ RISC のコンパイラで使われるいろんな性能の向上の技術は普遍的に使えるような利点がある．それから，今の CISC は中途半端じゃないか．集積度がだんだん大きくなったら，その余った部分は何に使うのかも問題だ，RISC のほうが使う可能性が CISC より少ないのではないか．というお話があったと思います．

それから RISC では C 言語のプログラムの実行が非常に速い速いという話がよく出てくるわけですが，本当に C がいつまでも使えるわけではなくて，もっと高レベルの言語が使えるようになる可能性もあって，そういうときには，やはり RISC よりずっと高いレベルの命令が使いいのではないか，まあそういったところの話をいただいたわけです．

それでは次にはたぶん RISC の立場でのお話が出てくると思いますが，三菱電機中央研究所のシステム研究部主事をなさっておられます平山正治さんからお話をいただきます．平山さんは RISC を使って，チップの設計などをおやりになった経験がありますので，そのへんからお話をいただけるのではないかと思います．お願いします．

設計が容易な RISC を用いて応用指向のチップを作る

平山 ご紹介いただきました三菱電機の平山です．今日は“RISC は CISC に勝るか”という話題なんですけれど，私どもは RISC 方式に基づいた

Prolog の高速実行用 μ プロセッサの開発をしておりまして，RISC の良いところはどこにあって，それをどういうふうにチップの開発に活かしたかというお話をさせていただきたいと思います．

私どもはこのチップを "Pegasus" という愛称で呼んでいるんですが，2 μm の CMOS で作られていまして，レジスタ・ファイルの部分は手書きですけれども，それ以外の部分は標準セル方式で設計されています．手書きの部分は約 40,000 トランジスタ，標準セルの部分は約 7,300 ゲートぐらいで，ほぼ 1 cm 角の小規模な μ プロセッサです．

では，このチップがどれくらい RISC 的かというお話を次にさせていただきます．このチップのすべての命令は，①フェッチ，②デコード，③レジスタ読出し，④演算，⑤無動作，⑥レジスタ書込みという 6 ステージの動作に分解され，2 ステージごとの 3 段階のパイプライン動作をするように共通化されています．また，6 個のロード／ストア命令以外はすべてレジスタ間演算を基本とするロード／ストア・アーキテクチャを採っています．このチップの制御回路はすべて専用の組み合わせ回路によって実現されています．命令数は 42 で比較的少ないんですが，実は複数の機能を修飾子で指定する演算命令を 1 個と数えていたり，後で述べますが，Prolog を高速に実行するためのいくつかの命令が用意されているため，実際の命令数はちょっと多めになっています．アドレス・モードに関しては，プログラム・カウンタの相対アドレスとオフセットつきのレジスタ間接アドレスの 2 形式しかありませんし，各命令形式も 8 ビットずつ，4 個のフィールドに分割した簡単な形式に固定されています．ハードウェアを簡単化した分だけソフトウェアで頑張ってもらおうという RISC の考えはこのチップでも同様でありまして，Prolog プログラムのコンパイル処理における最適化がこのチップの性能に大きく影響します．

ここで，1 サイクル実行，ロード／ストア・アーキテクチャ，少数の命令やアドレス・モード，簡単な命令形式といった RISC 方式の計算機は，どれくらいの論理規模で実現できるかについて考えてみます．例として UCB の RISC I を考えますと，チップ全体で 80 mm² なんですが，レジスタ・ファイルの部分が半分弱の 38 mm² を占め，残りの 42 mm² に 32 ビット ALU，シフタ，プログラム・カウンタ，制御回路などがすべて入っています．ここで注目していただきたいのは，この 42 mm² の部分には計算機として動作するための基本的な部分がほとんどすべて含まれており，これにアキュムレータでもつければ，これだけで RISC 方式の 32 ビット μ プロセッサになってしまうということなのです．RISC I の場合は，これで余った面積をオーバラップ機能をもった大容量レジスタ・ファイルに有効利用しているわけです．

RISC I は 4 μm の nMOS で実現されていたわけですが，現在は 1～2 μm の CMOS が全盛でありまして，仮に 2 μm として非常に単純化した計算を行いますと，42 mm² の計算機のコア部分は 10 mm² ほどになり，チップを 1 cm 角としますと全体の 1 割程度でこの部分が実現されることになります．1 μm プロセスではさらに小さくなって，全体の 1/40 になってしまいます．それでは，残りの 9 割，あるいは，39/40 をいったい何に使ったらよいのでしょうか．私どもは，チップのアプリケーションに依存した処理にこの余った面積を有効に利用できるのではないかと考えているわけです．

話をもう一度 Pegasus に戻しまして，私どもの目的は Prolog プログラムを高速に実行することなのですが，この目的のためにチップの面積をどのように利用したかについてお話させていただきます．Prolog のような AI 言語の場合，データにその型を示すタグを付加することがよく行われますが，Pegasus も 8 ビットのタグをもっておりますが，データの型に応じて動的に演算の内容を変更する機能はなく，単に 2 個のデータの型の組み合わせに応じて多方向分岐する"タグ・ディスパッチング"という機能，およびタグを対象とする演算，データ転送命令が用意されております．また，高級言語を対象とすると，どうしてもスタック指向になるわけですが，単にスタックの先頭だけを取り扱うのではなく，先頭からオフセットを加えた場所のデータを参照する機能や，この場所の値をレジスタに残しておく機能が Prolog の実現に有効になります．このために，Pegasus では 2 ポートのレジスタ・ファイルとアドレス加算器をもちまして，スタックポインタの値にオフセット値を加算した結果でメモリを参照でき，かつ，データの読み／書きと同時にスタックポインタの更新が可能なアーキテクチャとなっています．

もう一つの特徴は，Prolog の本質的な動作でありますバックトラックに備えて，現在の状態の退避／復帰を高速に行う機能を備えております．具体的には，

メイン側とシャドウ側と呼ぶ2セットのレジスタ群があり，1個の命令によって，どちらかのセットの全体を他方に一瞬にしてコピーする機能をもっています．ですから，バックトラックが起こる可能性のあるときに，メイン側のレジスタをシャドウ側にコピーしておき，以下の実行はメイン側のレジスタを対象に行われますが，それがフェイルした時点で，シャドウ側の内容をメイン側に戻せば以前の状態に高速に復帰できるわけです．もちろん，状態を連続して退避／復帰しなければならないことも起こりますので，シャドウ側のレジスタの内容をメモリに退避／復帰する操作を通常の命令の実行と並行して行う機能を備えています．

Prolog の高速実行のためにこれらの機能を備えているわけですが，これらに Pegasus チップのどれぐらいの面積が使われているかについてお話します．レジスタ・ファイルは全体で約 40,000 トランジスタですが，シャドウ側やコピー回路をあわせると，約 17,000 トランジスタ，全体の 43% が Prolog の支援に使われていると考えられます．また，標準セル方式で設計されている部分に関しては，アドレス加算器，シャドウ操作の制御回路，および，タグに関連する回路などを合計すると，7,300 ゲートのうちの 1,700 ゲート，約 24% が Prolog 支援と考えられます．この結果，全体の約 1/3 のチップ面積が Prolog の高速処理に使われているということになります．

これぐらいのことをやって，チップの性能がどうなったかということなんですが，Pegasus は約 250 KLIPS の推論性能をもっており，汎用 μ プロセッサ M 68020 に比べて 6 倍以上の性能が得られているわけです．

このチップの開発労力なんですが，Prolog プロセッサを作ろうと考え始めてから，このチップの上で簡単なテスト・プログラムが動くまでに 2 人の設計者が 1 年半で完了しています．このなかには，Prolog の解析，アーキテクチャの決定，シミュレーション評価，チップ設計のすべてを含んでいるわけで，本当のチップ設計は約 4 カ月程度で行っています．作り方としては雑な面もありますが，非常に短期間で開発を行っているわけです．

最後にまとめますと，私どもは Prolog の高速実行用 μ プロセッサを開発したわけですが，なぜ RISC 方式を採ったかというと，何と言っても開発が早くできるということです．前に言いましたように RISC 方式計算機のコア部分は非常に小さい面積で容易に作れてしまうわけで，余ったチップ面積と開発労力をアプリケーション専用の機能の実現に集中できるということであります．

そこで，先ほどの富田先生のお話とは逆の考えになるのですが，“RISC で ASIC！” というのはどうでしょうか．RISC は確かに限定された機能にしか有効ではないかもしれませんが，CISC の計算機を最初から設計するのはかなり大変なことだと思いますので，もう少し容易に専用チップを開発しようとするときに，RISC のコア部分をもってきて，これにアプリケーションに依存する機能を付加して ASIC を実現していくことができるのではないだろうか，というのが私の提案です．以上でございます．

司会 ありがとうございました．今のお話は RISC を使いました Prolog チップを製作されたご経験から，要するに RISC の一番大きな利点は設計が簡単であること，したがって，面積も非常に少なくてすむこと，そしてそのために空いたところを使って，いろんな形でサポートすることもできるし，簡単だということを利用して，専用のものに転換していくことも可能ではないかというようなことのお話だったかと思います．

それでは最後になりましたが，日本電気 C & C 情報研究所の所長代理をなさっております山本昌弘さんからお話を伺います．山本さんはもうずっと前から高級言語マシンを中心に，いろいろおやりになっておられまして，アーキテクチャならびにコンパイラの権威であられます．いろいろそのへんの面白いお話が伺えるかと思います．よろしくお願いします．

RISC と CISC の使い分け

山本 日本電気の山本でございます．私は今ご紹介いただきましたように，ずっと CISC ふうアーキテクチャの研究を主としてやってきておりますので，たぶん CISC 派の立場で選ばれたんじゃないかなというふうに思っているわけですけれども，今までの 4 人のパネラの方からかなり議論が出ているかと思いますので，かいつまんでお話いたします．

必ずしも私も CISC がすべてだということをずっと思っているわけではないんですが，私は「RISC は CISC に勝るか」というのは，結局はどういうコンピュータを作るかということで，使い分けるべきであ

るという立場だろうと思うんですね.

それで,たとえば簡単なアーキテクチャを,なにもCISC チップでわざわざ複雑に作る必要はないわけで,たとえばCマシンあるいは UNIX マシンなどは高級なハードウェアの構造にしてもいわゆる CISC としてのメリットはあまり出てこない.

たとえば COBOL の専用コンピュータだとか,あるいは Prolog を直接実行するような計算機などにみられるように,複雑な応用に対して,いかに効率よくハードウェアでサポートして作るかというところに,CISC のポイントがあるだろうということで,結局は対象によって使い分けるべきであろうということだろうと思います.

RISC の利点として領域を限ってみれば,実力がはっきりする.ハードウェアあるいは LSI を作るという立場からみれば,RISC は非常にいい.しかし,ソフトウェアまで含めて考えないと,コンピュータのアーキテクチャを議論するのは,私は早計じゃないかな,そういう立場です.

CISC の利点と RISC の弱点

私は CISC の良い点,についてまずお話します.RISC の悪い点は,RISC のポイントはいろいろありますが,命令セットを簡単にして,単純な命令にするだけにして,命令の実行速度を速くして,LSI ハードウェアを作りやすくする点にあり,ソフトウェアをどう考えるかというポイントについてほとんど言及されていない.たとえばバークレーの SPUR チップ,あるいはスタンフォードの MIPS チップ,あるいは IBM の 801 といったものの出所をみると,LSI 屋さん,ディバイス屋さんが中心になって生み出してきたアーキテクチャであるということが一つ言えるんじゃないかなという気がいたします.

CISC の立場でみると,命令セットを OS とか言語だとか,そういった分野をうまく表現できるアーキテクチャということで,当然高速化を狙うと同時に,ソフトウェア開発を効率よくやろうというところにあるだろうというふうに考えられます.

それで CISC は,今までの私の経験でいきますと,一つはトータルにみたメモリ・アクセスというのをいかに減らすかということ.もう一つは,複雑な高級な命令を用意して,いわゆる低レベルの並列処理と,やれる範囲で並列処理をどんどんやって,効率よく実行してやろうというところにポイントがあるだろう.だからそういうことができるような対象の応用分野のものを選ばなければ意味がないわけです.逆にいえば,Cや UNIX レベルでの機能ですと,非常に低いというような気がします.

また,RISC アーキテクチャの問題点として,高級なコンパイラ,最適化されたコンパイラが要求されることです.今のたとえば SPUR, あるいは MIPS チップでは,大学関係を中心とした多数の研究者がコンパイラの研究をずいぶんやって,やっとできあがったものです.そういう実績の下で出てきているだろうと思うんです.それがいろいろな分野のアーキテクチャに対して,あのレベルの高品質のコンパイラを開発していけるかどうかは大きな課題です.

それから,簡単な命令セットでは簡単なデータ処理に対しては,非常に速いわけですけれども,複雑なデータ操作に対しては十分な性能が出ません.これは,たとえば十進データとかビット処理だとかリスト処理だとかタグ操作などを高速化しようとするとCISC ふうになっていき,だんだん境目が怪しくなってくるということじゃないかと思います.

事務処理のベンチマークで,Sun 3 と Sun 4 を比較した例が報告されていますが,Sun 3 は事務処理をやると,Sun 4 よりも 40% ぐらい速いと結果がでています.生の性能でいきますと,Sun 3 は 4 MIPS で,Sun 4 は 10 MIPS です.

私も COBOL の高級言語マシンの資料のなかにも報告していますが,十進処理のようなメモリ・ツゥ・メモリ・オペレーションでやったほうがいいような処理は RISC はあまり得意じゃないと思います.

次に,リスト処理について考えてみます.バークレーの Prolog マシン PLM について RISC チップ SPUR 上に PLM を実現して評価した例では,PLM が SPUR と比べて 50% ぐらい良いという結果がでています.若干 CISC ふうのアーキテクチャが良いという値が出ています.

次はオブジェクトメモリ量の点です.短い命令,単純な命令のみで構成しますと,いわゆるオブジェクトのメモリ量が非常に増えて,そのメモリのアクセス頻度がまたすごく増えるわけです.この点についてもいろいろ報告されております.さきほどと同じ PLM の評価のなかで,さきほどのようなベンチマークに対して,SPUR は PLM に対して,静的および動的なオブジェクトデータで 14 倍,17 倍,のデータ量になると報告されています.

もう一つの点は，チップのなかを単純にして，データあるいは命令を最大限に供給して，なかの動作を遊ばせないで，非常に速く実行させるわけですから，命令供給というのが非常にキイになるだろう．そこで外部のメモリアーキテクチャを高級にしないと，最大限の性能は出ないということになります．

それからCISCふうアーキテクチャとして汎用コンピュータがあげられます．これを単純なRISCで実現すると互換性や性能面で厳しい．この互換性の問題を解決するにはRISCになんらかの工夫が必要と思います．AIマシンについてはいろいろ議論がまだ現状でもなされています．いわゆる，Prologマシンでも Warrenのアブストラクト・マシン，さらにシンプルなアーキテクチャにして，RISCふうにするとかなり高い性能が出るという議論もありますが，私はCISCふうアーキテクチャが合っているんじゃないか，むしろ今のCISCで性能が出ないというのはCISCの研究がまだまだ不十分じゃないかなと思います．

最近このような高機能化に対してRISCチップのなかでもフローティング・ポイントだとかキャッシュだとか，あるいはMMUだとかなどを取り込もうとしている傾向があります．今後，さらに多くの機能をチップのなかへ取り込んでいくと機能が高度化していくことになり，CISCへの接近が出てくる可能性が非常に強いんじゃないかと予想します．

それからもう一つ，今のRISCはいわゆるUNIXマシンであり，Cマシンであります．このレベルでみるとかなりいいだろうと思うんですが，これが他の応用に利用できるかがポイントです．

以上はアーキテクチャという狭い領域で述べたわけですが，私はアーキテクチャのハードウェア，LSI設計のレベルだけ議論するんじゃなくて，ソフトウェアまで含めて議論しないと，議論がたりないだろうと思います．むしろソフト重視の判断というのが非常に重要であろうということで，ただRISCとCISCの性能面でみますと，単純な応用はRISCがいいだろうし，複雑な応用はCISC的なものがいいということになります，それからもう一つ重要な項目に開発コストがあります．それでハードにつきましては，さきほど報告がありましたように，シンプルな構造で，速く作ればいいという意味でRISCがいいでしょうけれども，ソフトウェア側からみればCISCのほうがいいと考えられます．

もう一つ，これは元々の高級言語マシンの狙いになりますが，信頼性に関する評価があります．RISCふうなアーキテクチャになってしまいますと，なかなか高レベルなエラーの検出ができなくなります．ソフトウェアからみると，CISCのほうが高いエラー検出が可能です．ただハードの信頼性からみれば小さい規模で作ってしまうわけでしょうから，非常に信頼性がいい．しかし，トータルにみれば，CISCあるいはCISCふうが長期的には重要と考えられます．

司会 ありがとうございました．RISCもCISCもおのおの特徴がある．したがって，それぞれがいい分野があるのではないか，それから単にハードウェアの作り方ということでなく，ソフトウェアも含めた総合的な判断が必要であろう．また，さきほどの日比野さんのお話と同様に，UNIXとかCとかというレベルでなくなったときには，また状況が変わってくる可能性がある．まあそういったお話をいただいたかと思います．

以上で一応パネリストのみなさん方からお話を伺いまして，総体的に絶対RISCだよとか，絶対CISCだよという話にはなってなかったかと思いますけれども，それぞれの良いところ，それから悪いところ，おのおのの主張というのがいろいろ出てきたかと思います．

皆さまのなかには，世の中では最近非常にRISCのマシンが増えていますので，RISCのほうがよいという考えの方が多くなったのかなと，思っておられた方もいらっしゃるかと思います．私もちょっとそんな気がしてまして，パネリストをお願いするときに，必ずしもそうではないのだという私の偏見が少し入ってたかもしれません．これまでのお話からすると，もしかすると，今日のパネリストは世の中よりはRISCという人が少ないという印象を受けられたかもしれません．今お話に出なかった面とかあるいはもっとRISCが，これだけいいんだよというご意見もいろいろあるかと思いますので，その点についてはこれからのディスカッションでいろいろご意見をいただければと思います．

それではフロアの方からご意見をいただきたいと思いますが，お話になるときには一応お名前と所属をおっしゃって議論していただきたいと思います．それから特にパネリストのなかのどなたかの意見をお聞きになりたいということがございましたら，ご指名をいただいても結構でございます．

それでは，何かございますでしょうか．どうぞご遠

慮なくお願いします．パネル討論会はフロアの方からいろいろ言っていただかないと，なかなか面白くなりませんので，積極的にご参加をお願いしたいと思いますが．いかがですか．

RISC は命令セットを最適化しているから CISC より速い？

質問 富士通研究所の久門と申します．ただいまのパネラの方のご意見をお伺いしまして，もうちょっと独断と偏見に満ちて，絶対に私はいやだというような意見を期待していたんですけれども，みなさん，両方を融合させたようなものがいいんじゃないかという意見で，ちょっと私はものたりなかったんですが，私，ちょっと意見を述べさせていただきます．

みなさん RISC と言いますと縮小命令セット・コンピュータと日本語で訳されて，要するに命令セットが単純な計算機があり，CISC というと，それに対して，どちらかというと高級言語指向の計算機というふうに対比されていない概念というふうに，僕は考えています．それで，ちょっとここで RISC というのを縮小命令セットであるということを忘れて，具体的に外延的にこれが RISC だということを列挙して，それがどういう特徴をもっているかということを，あっさり，何も考えずに決めてしまおうと，遊びでやってみました．まず，SPARC というのは RISC である．それから Mips 社の R 2000, 3000 も RISC である，それから IBM の RT/PC の要素プロセッサ，あれも RISC なんだと，それからモトローラの 88000，たぶん世の中にまだ出ていないでしょうが，一応性能などが公表されているので，これと，それからあとは AMD の 29000．それ以外のものは，RISC じゃないか，CISC であるというふうに考えた場合，クロック当たりの命令実行数というのを計算してみるんです．そうしますと，当然 RISC というのは，みなさんのお話のなかにもありましたが，クロックが速いから速度が速いというような意見が一般にありますけれども，これをクロックの周波数で割ってしまって，クロックのスピードは命令実行と関係なくするということをやりますと，まあ何を MIPS として計算するのかというのが問題になりますが，ここでは最近の流行りでもありますので，一応 UNIX の，それもかなり小さなベンチマークで評判も悪いんですけれども，Dhrystone・ベンチマークあたりから換算した MIPS 値といったのを計算しますと，1クロックで何インストラクション実行したか，RISC の命令じゃないですよ．標準インストラクションを何命令で実行したかを計算すると，明らかに RISC と CISC というのが，二つに分かれるんですね．

それで，どっちが大きいかというと，RISC のほうが全然大きいんです．だいたい数字でいいますと，僕もちょっとうろ覚えなんですが，0.5 より上が RISC，それより下が CISC というふうになっています．CISC の一番大きいところでも，まあすべての計算機のクロック数が分かっていませんので，分かっている範囲ですと，VAX の，たとえば 8650 ぐらいですが，これが 0.3 の上のほうだと思ったんですが，それぐらいの数値です．それに対しましてモトローラの 88000, 29000 クラス，あるいは MIPS 社の R シリーズですと，0.8〜0.9 ぐらいですか．

ですから，1クロック当たり何標準命令実行したかということを考えてみますと，RISC のほうが全然速い．つまり，RISC というのは，クロックを上げなくても速いということなんですね．

それで，どうしてかということを考えてみますと，さきほど日比野さん，山本さんのお話のなかにもありましたが，ある意味で RISC，特に今出ている RISC というのは UNIX の専用化されたプロセッサであるということです．だから命令を縮小しているかどうかにかかわらず，RISC というのは専用命令計算機なんではないかと．もちろん十進演算というのが Dhrystone のなかに入っていませんから，十進命令をやれば遅い．でもそれは十進命令のプリミティブというのを RISC のなかに突っ込むということをやってやればいいわけで，それと高級言語指向マシンであるということは矛盾しないような気がしているんですね．

それで，今言ったようなことから考えまして，アプリケーション・プログラム，最終的にはデバイス技術というのがクロックで決まってきてしまうということを考えたとき，1クロックで何命令実行するかというのが，高ければ高いほど良いアーキテクチャに決まっていると考えると，RISC というのは，じつは命令セットを縮小したのではなくて，やはり最適化したほうの計算機であろう，というふうに考えていますけれども，それについてはどのようにお考えでしょうか．

司会 どなたか，特にご指名はございますか．よろしいですか．では稲吉さんお願いします．

稲吉 今の1サイクル当たり何命令で実行したかと

いう単位，その逆がよく使われるんですよね．1命令当たり何クロックかかっているかというサイクル・パー・インストラクションという単位なんですが，まあ同じことなんですが，結局，出すオブジェクトが違っているんです．

たとえば，これはさきほどご紹介いたしましたけれども，これだけの命令を RISC は出すんですね．コンパイルしたオブジェクトとして，CISC はこれ1個で済む．ところが処理時間は同じだとしますと，表現の上では，まあ端的な話，MIPS でいってもいいんですが，この処理の実行が 1 μsec であるとすれば，1 MIPS ということになりますね．その代わりこの5つの命令を実行するのに 1 μsec だとしますと，これは5 MIPS という表現になるわけです．しかし，処理時間は同じ 1 μsec ということでして，出すオブジェクト命令がどのくらいの機能をもっているかということで，MIPS の表現値が異なります．MIPS は1秒間に何命令実行するかという尺度ですが，1クロック当たり何命令を実行しているかというのも同じ尺度であり，じつは客観的な性能判断基準じゃないというふうに，私は考えております．

質問（久門） 今のご回答は，ちょっと誤解があるようなので，もう1回補足して質問させていただきますが，私のほうの質問は，たとえば Dhrystone・ベンチマークというのはだいたい直感的には，1,600 から 1,700 ぐらいで割ると MIPS の数値が出るというふうに考えております．ですから VAX で 1,500～1,600 Dhrystone ぐらいですから，だいたい 1 MIPS 前後というふうに考えて，要するに実際にそれが Tron の命令で何命令か，あるいは SPARC の命令で何命令かということを一切抜きにして，Dhrystone の数字をだいたい 1,600 から 1,700 ぐらいで割ると MIPS の数字が出ます．その MIPS というのは，さきほど標準インストラクションと申しましたが，それは実際の命令数とは関係なく，ノーマルオペレーションとして標準的に考えて MIPS を議論するときの命令をいっています．その命令は SPARC だったら 100 命令になるかもしれませんし，Tron ですと1命令で終わるかもしれない．ユーザからみたら同じことをやっているわけですから，どの場合も一つのものだと考えた場合に，やはり，さきほど申しました SPARC のほう，あるいは R 3000，2000 のほうがかなり値が高い．

つまり，機能的には命令の規模といいますか，1クロック当たりに実行できる実質的内容が多いというふうに考えております．ということなんですけれども，よろしいでしょうか．

司会 お話の MIPS は各計算機での命令実行数で測った MIPS とはちょっと違うんですね．これはノーマライズした命令で数えても RISC のほうがいろいろやってみると，本当に速いのではないですかというコメントだと思います．

稲吉 そうですね，たとえば今1クロック当たりの平均命令実行数というのが SPARC だと 1/1.5 ぐらいになっていると思います．しかし，それもたとえば Dhrystone を C でコンパイルして出したオブジェクトがどういうオブジェクトを出すかということとやっぱり切り離せないと思うんですよね．

それを VAX でノーマライズするというのはどういうふうにノーマライズするというのか，ちょっとよく分からないんですが．結局，Dhrystone の処理時間そのものと比較していることになります．たぶん命令数でいきますと，VAX の C コンパイラを使ってオブジェクトを出すとどのくらいになるんですかね．それよりも SPARC の C コンパイラで出すオブジェクトの数のほうが多いはずなんですね．だいたい 1.5 倍ぐらい，つまりコード長がそれだけ長くなっているんです．だから時間当たり，またはサイクル当たりの命令実行数で比較する場合には，その 1.5 倍のファクタで割らなければならないと思います．

私はとにかくアーキテクチャのなかには，インストラクションセットが含まれているというふうに考えていまして，できるだけコンパクトなオブジェクトを出す命令セットのほうがいろんな意味で良くなるんですね．性能が．というふうに考えておりますけれども．

司会 今質問された方，もうちょっとご意見があるのではないですか．

質問 そうですね．どういうふうに言い換えればいいかよく分からないんですけれども，たとえばGマイクロ 200 のほうは，クロックがどのくらいで，まあ Dhrystone 値というのは最近ですと，当然発表されていると思いますけれども，なんていいますか，もちろん RISC の場合には命令セットがでかくなる．あるいはオブジェクトコードが大きくなるのは事実ですし，実際大きくなっていると思います．

それで普通 CISC ですと1命令で実行できるような命令を複数の命令に転換したり，あるいはループで転換したりと，いろんな方法を使って大きなオブジェクトコードを吐き出していくわけです．ただユーザと

いうのは，完全なアーキテクチャのコンパイラを作るということを抜きにして，もう完全に言語レベルのユーザが考えるときには，速いマシンがいい．もちろんデバッグの容易さということもあって，そういった意味での高級言語マシンというのは，たしかに有益だと僕は思っておりますが，それをちょっと除いて，スピードの点だけ考えたときに，たしかに命令供給が追いつかないので遅いだろうとか，というふうに観念的にいうことはよく分かりますし，あるいはストリング・オペレーションというのがマイクロでこうやって書いたほうが速いだろうということ，そこのポイントだけ見ればそれはわかるんです．しかし，実際に出てきた数字だけ見てみると，やっぱり遅いんですね．

それで，さきほど高機能命令をいろいろ CISC はもっているから速いんだというふうにおっしゃられましたけれども，私はなんで RISC が速いのかということを，見てみたところ，まず第一に簡単な命令セット，簡単な命令セットだといいますけれども，3オペランド命令を使っている点があげられます．これがかなり今のコンパイラの最適化の手法に効いているみたいなんです．なぜかといいますと，A＋B＝C というのを A も B も壊さないで，そのまま演算したいというのは，最適化コンパイラの場合はよくあることで，A も B もあとで演算で使うとすると，C トロンの命令セットをよく知らないので申し訳ないんですけれども，たぶん3オペランド命令もあるでしょうけれども，普通世の中に出ている，従来のコンピュータですと，だいたい1回セーブしますね，それから後，ロード命令が入る．つまり2命令で実行せざるをえないんです．そこがその3オペランド命令の場合は，1命令で実行されます．

それで，高級言語でストリングムーブであるとかいうのは，たしか聞きますけれども，そういったレジスタ・トランスファ・レベルの冗長性というのは，結構多いんですね．数としてみると．それで最終的に RISC のほうが速くなったんじゃないかというふうに考えていますけれども．

ですからユーザサイドからみたら，オブジェクトコードが小さいか大きいかというのは，もちろんメモリをいっぱいくうというのは切実な問題になるかもしれませんが，まず第一にスピードだけ考えたときには，高機能命令が必ずしも速い方向には伸びてくれていないのではないかと，もうちょっと具体的に OS のなかでこの機能を入れたことによって，全体が何パーセント向上するというような数値的なシミュレーションの結果として，命令セットを設計すべきではないかという，そういう意味でバランスのとれた計算機というのが一番速いのではないかというふうに考えています．

司会 高級言語マシンやソフトとの関わりが少し出たので，山本さんご意見ないでしょうか．

山本 今のご意見，私も分かるところ非常に大ですね．それで，たとえば今の RISC アーキテクチャは，そのたとえばCのプログラムにおける命令使用分布をとり，非常に頻度の高い機能を命令セットとして作ったアーキテクチャである．だから当然Cのアプリケーション，Cで書いたプログラムが非常に速い．それと同じことをもうちょっと高いレベルの問題をもってきて，それに絞って作った複雑な構造をもった計算機は作りうるだろうと思います．これは非常に特化された高級言語マシンになるかもしれません．そのときに，ハードウェアの中身を見たときに，いわゆるさきほど議論になった単純化されることによってクロックを短くできるから，RISC が速いというその現象と同じ状況を，高級言語の実行レベルの高いアーキテクチャでも実現できるんじゃないかと思うんですね．

ただ，私はさきほど議論したときには，もう少しグローバルな立場でとらえ，もっといろんなものを実行するとか，そういうことをとらえていかなければいけないだろうと申しあげたのです．そういうことでいくと，やっぱり複雑なことができるというメリットが出ると思います．また，並列処理的なところをどんどん取り込んでくる．それによるメリットが出てきて，トータルには速くなる．それを RISC でやると，パイプラインで実現することになり，これによってどのくらい稼ぎうるかということとの差で決まってくるだろうと思うんです．セマンティックなところまで含めて考慮し，並列処理的なところを取り込むことができるだろうから，単純な命令の物理的なパイプライン処理よりも，もっと速くなる可能性はあると，いろいろ実験した結果のなかでも報告されています．

司会 ありがとうございました．なかなか有効なコメントをいただきまして，いろいろ議論がありました．まだあるのかと思いますけれども，他にも議論があるのではないかと思います．何か他にご意見ございますでしょうか．

命令実行の並列化からみた RISC と CISC

質問（村上） 九州大学の村上です．まずは，RISC

の定義についてですが，これが非常に曖昧模糊としていてよく分からないんです．私なりの定義を述べさせていただきますと，RISC というのは命令の機能の高低うんぬんは別として，命令で指定されるオペレーションが非常に定型的な処理に限定されているものだと思います．

一方 CISC では，命令で指定されるオペレーションが非定型的で，実際にやってみないと分からない．たとえば，さきほどのストリングサーチ命令とか，十進命令もそうなんですけれども，可変長データの場合何回演算をやるかデコードしないと分からないわけです．こういうふうに，CISC では非定型的な演算要素をたぶんに含んでいると思うんです．

そこで，二つお聞きしたいと思います．先ほど山本さんのほうから低いレベルの並列処理を生かすには，CISC のほうがよろしいというお話があったんですけれども，私どもは多重命令パイプラインの研究をやっていまして，命令パイプライン処理にいかに低レベル並列処理，とりわけアウト・オブ・オーダ実行を取り入れるかということを考えています．そのときに CISC と RISC という二つのスペクトラムを考えた場合に，CISC というのは非常に非定型的であるがゆえに，1命令内ではともかく，命令間での低レベル並列処理というのが非常にやりにくいわけです．なぜかというと，1個の命令がメモリメモリ演算をやるとか，何回も演算を繰り返すとかで，演算装置を常にビジィに保っているわけで，他の命令の処理ができない．一方，RISC では，そもそもパターンはパイプライン化しやすいように，1サイクルで演算が終わるように命令セットを定義したわけです．しかし，最近の RISC ではモトローラの 88000 とかみると，乗算命令とか浮動小数点命令とかあって，必ずしも1サイクルで終わらせていない．5サイクルとか6サイクルかかっているわけです．

というのは，演算パイプラインを複数個用いることによって，外からは1サイクルで終わっていると見えるように，命令間の低レベル並列処理をさせているわけです．つまり，RISC では命令のパイプライン処理だけでなく，命令間での低いレベル並列処理というのも，可能になってきていると思うんです．

それで質問ですが，日比野さんは RISC はこれから何もやることがないんじゃないかというふうにコメントされているんですけれども，RISC というのは逆に命令が定型処理であるがゆえに，さらなるパイプライン処理なり低レベル並列処理というのが可能ではないかと，私は考えていますが，いかがでしょうか．

マイクロプログラムの生産性対コンパイラの生産性

それが一つと，これと関連してもう一つは，さきほどのソフトウェア・クライシスということで，コンパイラの開発が非常に RISC では難しいという話だったんですけれども，逆にハードを作る人間からみると，マイクロプログラムを作るほうがもっと難しいんじゃないかなという気がします．というのは RISC の AMD 29000 とか SPARC では，将来の性能予測値を見ていると，年に2倍3倍というふうに向上していくわけです．しかも，2年おきぐらいにそういう性能アップした製品を出せるといっているんですけれども，CISC というのは，もうちょっと性能向上の度合というか製品サイクルというのは長いような気がします．

その原因はどこらへんにあるのかなと考えますと，やっぱりマイクロプログラムの生産性のほうが低いんじゃないかなという気がします．実際にものを作られている方にお聞きしたいんですけれども，そういうマイクロプログラムの生産性と，ソフトウェア・コンパイラの生産性とでは，どちらのほうが高いのか低いのか，そこらへんをちょっとお聞きしたいんですけれども．

司会 さきほど，日比野さんのお名前も出ましたし，議論もあったかと思いますが．

日比野 マイクロコードの生産性の話ですけれども，実際のところ，私どもも悩んでおります．さきほど超々 CISC ということで，すべてを直接実行するような形でやりますとマイクロコードの量というのは，非常に厖大な量になります．そういったものを，今度テクノロジが変わったということで，別のものにもっていこうとしましても，結局またそのレベルでアーキテクチャを変更しなければならないということが起こりまして，マイクロコードを捨てなければならなくなります．そうしますとデバイス・テクノロジをそのまま有利なところを生かすことができないということで，大量のマイクロコードというのは逆に大変な障害になってまいります．

ということで，ほどほどにしなければいけないというところがありまして，実際に製品作りをする立場から申しますと，たしかにご指摘のとおり大きな悩みが

ございます．さきほど3オペランド命令のことが出てまいりました．ああいうようなことを考えますと，適切なレベルのマイクロレベル・アーキテクチャを設定している場合には，マイクロコンパイラを適切に作ることができまして，それを利用して広い応用範囲に対して，性能をチューニングしていくことも可能になるのではないかと思います．

その場合には，RISC の命令セットに対する最適化コンパイラを作るのと，ある意味で同じようなところがございまして，その両者についてはあまり差がないのではないかなという気がいたします．ただマシンの利用者の立場からすると，だいぶ性格が違ってきまして，さきほどもちょっと舌たらずだったんですけれども，非常に普遍的全体的に効きめがあるという使い方と，個別的特定のものにしかうまくいかないというそういった差が出てくるのではないかなと思います．

司会 もう一つのほうはどうですか．さっき RISC には何もやることがないのではないかという話がありました．

日比野 アーキテクチャの話としてというような意味あいで，ちょっと申しあげたんで，インプリメントの立場からすると，ずいぶんやるところがあるというのは，たしかにおっしゃるとおりです．スケール則ということを出しましたのは，2乗のスケールでゲート数が増えていくということを，どうやって有効に使っていくかという観点からみると，RISC という提案自身が今後非常に長い間コンピュータそのものの中核部分を支配していく考え方になるということは疑問に感じますという意味です．

たしかに，プロセッサを簡単にしまして，まわりにいろんなものをつけまして，アプリケーション・オリエンテッドの ASIC を作ってシリコンの面積を利用していくという考え方は，もちろんあると思いますが，プロセッサの中核の部分にといったときに，どうなんでしょうかということでありまして，そこにつきましてはやはり別な考え方が出てきてもいいんではないかということです．

富田 私から一つ，CISC と RISC の違いの話がさっき村上君のほうからあったんですが，これはパイプラインでやるのかマイクロの並列処理でやるのか，そこの違いだとも思えるんですよね．RISC はパイプラインに非常に向いているような構造，CISC は並列演算に非常に向いているような構造のような気がするわけです．そうすると並列処理のなかにはかなり，やっぱり並列性がないとうまくいかないというふうに思うわけです．

そういう点で命令セットをいろいろみてみますと，あまり並列性があるような機械命令というのはないんじゃなかろうかと思います．あったとしても繰返し構造のものが非常に多いと思います．かなり複雑な判断をやりながら並列処理していくような CISC の命令というのは，現在の CISC のプロセッサにはないんじゃないかというふうなことが言えると思います．今後の CISC では，機械命令の機能レベルを高め並列演算を引き出す必要があります．そういう点で現状ではパイプラインに向いたような RISC のほうが速くなっているんじゃないかというふうな気がいたします．

それからもう一つ，マイクロの生産性というような話があったんですが，これはやっぱりマイクロ・アーキテクチャをどういう具合にクリーンな構造で作るかといった点にも，かなり絡んできて，汎用機のようなぐじゃぐじゃとしたわけの分からないようなマイクロを作ってもちょっとどうしようもないんじゃないかというふうに思います．なるべくきれいな構造のマイクロにすればかなり生産性は上がるんじゃないかと思います．われわれが昔京都のときに作った QA-1 とか QA-2 というのは，そういう点では非常に分かりやすい．分かりやすいんですが，非常に長くて 250 ビットぐらいの命令があったんですが，ある程度そういうマイクロの生産も考慮に入れたようなマイクロ・アーキテクチャを設計すべき時期になっているんじゃないかというふうな気が，昔からしています．マイクロは伝統の産物でどうにもならないかもしれませんが．

司会 ありがとうございました．だいぶ時間がたちましたけれども，もう一つぐらいご意見ご質問がありましたら，いかがでしょうか，ございませんか．

なかなか面白いご質問を二つほどいただきましたけれども，まだ他にもう一つぐらいあるとよろしいかと思いますが，どうぞ．

命令実行制御論理の設計と CAD

質問 大阪大学の斉藤です．CISC のほうが命令の充填度が高いという話がありましたが，一応マイクロ命令と RISC の1命令というのはだいたいよく似たようなことをしているわけですね．クロックを上げていくと CISC のほうが命令デコードの PLA とマイクロ・プログラムのアクセスタイムで先にボトルネッ

クがきそうな気がするんですが，さきほど富田先生のお話がありましたように，あそこの CISC の複雑な命令のデコードをランダム・ロジックでできるようなCAD とかの技術の進歩が本当にきそうなものかどうか見通しをちょっとお聞かせ願いたいんですけれども．

司会 これはどなたですかね，詳しいのは．どなたかコメントをお願いします．CAD の将来性，稲吉さんいかがですか．

稲吉 CAD は分からないけれども，たしかにトロンのデコーダを作るのが一番大変だったというのは事実なんです．でもいったんできちゃえば，あとはそこへ所定のタイミングで通過させればパイプラインは乱れません．現実に別のGマイクロチップでは，マイクロだけじゃなくて，ハードワイヤでやる部分というのがかなり入っているのもあります．

司会 平山さんはいかがでしょうか．あるいは，さきほどの部分についてでも結構ですけれども．

平山 CAD の話に関してはよく分からないのですが，データパス上でのデータの流れとそれのタイミングを設計者がきちんと設計していれば，制御回路は比較的容易に自動生成できるのではないでしょうか．実際，そのような論理合成をするような設計ツールを考えているかたもいらっしゃるようですし，論理の最適化の問題は残るとしても，自動化ができるようになりつつあるんじゃないかと思います．

それで前の話にもどりますが，アプリケーションとかターゲットが比較的限定されているのであれば，RISC ということを前提にしても，その専用機能をうまく組み込んでいけるのではないかという感じをもっています．

先ほど私はすでにできあがっているほうの Pegasus の話をしましたが，現在，改良中の Pegasus は，ポインタをさかのぼる機能をもった命令が追加されております．これはまったく非 RISC 的命令でパイプライン動作が乱れてしまうんですが，こういうことを行うのにも RISC のコア部分はあまり変えずに組み込むことができるわけです．ですから，対象の的が絞れれば，RISC プロセッサのなかに CISC 的なことをある程度入れていけると考えています．

一方，μプログラム方式には別の良いところがありまして，当社の PSI も Prolog 的な処理の高速化を狙ったプロセッサでありますが，こちらはμプログラム方式で作られています．この計算機では推論処理だけでなく，OS 全体を効率的に処理しなければならず，それに対するフレキシビリティを高めるためにμプログラム方式が有効に使われています．ただ，μプログラム方式では制御記憶の容量が大きくなりがちで，とても1チップ化できず，小型化とか高性能化に対して悪影響を及ぼすのではないかという気がします．

司会 ありがとうございました．ちょうどただいま時間になってしまいました．いろいろと面白いコメントを頂戴しました．RISC は CISC に勝るかということについては，そう簡単にどちがいいという結論にはなりませんでした．これは最初からある程度予測されていたことなんですけれども，どちらがいいかというのは RISC の定義にもよりますし，また他にもいろんな面もあります．単に速度だけだったら，やっぱり RISC のほうがいいのかもしれませんし，作りやすさからいっても RISC のほうがいいのかもしれませんが，実際計算機の命令セットは何がいいかというのは，ご承知のように，単にそれだけの問題ではありません．さきほどいろいろ議論がありましたように，ソフトウェアを作りやすいとかデバッグをしやすいとか，コンパイラとかいろんなことが係わってきて，本当に良い悪いが決まるわけです．そしてさらに，今日はあまり話が出なかったんですが，命令セットの議論では当然互換性という話がいつも出てくるわけでありまして，次々と新しい命令セットをどんどん作っていくわけにはいかないわけです．そのときに，技術のほうは CAD も進歩しますし，半導体の技術も進歩するし，またどんどん変わっていってしまう．やはり RISC をやるにしても CISC をやるにしても，単に速度や作りやすさだけでなくて，将来のことを考えなければなりません．命令セットのアーキテクチャというのは他の技術に比べて非常に寿命が長いものですから，決定に際しては十分考えなければならないと思います．

今日は結論は出ませんでしたけれども，おかげさまで RISC，CISC についていろいろ面白い議論ができたのではないかと思います．司会がはなはだ不慣れでございまして，うまくまとめられなかったことだけは残念でしたけれども，全体としてはよい議論ができたのではないかと思います．どうもありがとうございました．これで終わります． (以上)