解　説

命令セットアーキテクチャ

5. 命令セットアーキテクチャの評価†

宇都宮公訓††

1. はじめに

コンピュータシステムの評価は，システムプログラムも含めた全体として，以下のすべての項目について行うべきと考えている．

（1） 処理能力：単位時間当たりのジョブ処理量

（2） 応答時間：特に対話式処理，即時処理にあって，人間がコンピュータに待たされている時間

（3） 機能：データ処理，データベース管理，データ通信などコンピュータシステムができること

（4） RASIS：Reliability（信頼性），Availability（可用性），Serviceability（保守容易性），Integrity（保全性），Security（安全保護性）

（5） ソフトウェアの開発保守支援

（6） 操作：操作コスト，使い勝手など

これらの項目に，さらに，ユーザ側では価格，メーカ側ではコストが重要な項目として加わる．

一方，コンピュータシステムの実現技術にはハードウェア（H/W），ファームウェア（F/W），ソフトウェア（S/W）があり，これらは組み合わせて使われる．速度性能はこの順に悪くなり，可変性はこの順に大きくなる．一般に，コンピュータシステムは H/W の割合が多くなるほど速くなるが可変性は小さくなり，S/W の割合が多くなるほど可変性は大きくなるが速度は遅くなる．F/W は H/W と S/W の中間的性質をもち，両者の掛橋として重要な役割を果たす．LSI 技術の発達で H/W コストは急激に低下した．LSI では少品種大量生産できるほどコストが低下する．したがって，分散処理，並列処理の導入が強く促されている．以前は大きなマシンほど性能価格比が良いとされたが，現在は小さなマシンほど良い．

コンピュータアーキテクトの課題は，上記の（1）～（6）で良い点数がとれるコンピュータシステムを H/W, F/W, S/W を組み合わせてできるだけ安く実現するようにその（骨格）構造を決めることである．このうち，H/W, F/W で実現する部分をマシンといい，S/W からみたマシンとのインタフェースを（マシン）の命令セットアーキテクチャという．コンピュータアーキテクチャは，たとえば TRON のように，ユーザインタフェースから内側，すなわち素子に向かって設計していくのが原則である．その過程で命令セットアーキテクチャも決定される．

大雑把にはマシンは S/W を動かすモータもしくはエンジンといえる．コンピュータシステムを一つのエンジンで動かすことも，複数個のエンジンで動かすこともできる．動かす S/W に合ったエンジンのほうが効率が良いに違いない．LSI の少品種大量生産の原理もあり，「どんな S/W でも動かす一つのエンジン」より「自分に合った S/W だけを動かせばよい複数個のエンジン」からなるシステムのほうが性能価格比が良いことが多そうである．自分に合った限定された S/W だけを動かす一つのマシンからなるシステムもあろう．この当然の帰結としても種々の命令セットアーキテクチャをもつマシンが作られるようになった．

現在の多種多様の命令セットアーキテクチャを統一的に比較評価することは難しい．目的が違う命令セットアーキテクチャを一つの立場で評価することはほとんど無意味である．ここでは，**2.～3.** で一般に行われている評価を紹介した後，**4.～6.** でマシンの計算速度という視点だけから命令セットアーキテクチャ（以下，単にアーキテクチャと書くことが多い）がどうあるべきかを議論する．

2. 評価基準

飯塚[1]は，良い命令セットアーキテクチャは

（1） コンパイラが効率的オブジェクトを作りやすい

（2） オペレーティングシステムがシステムの管理

† Evaluation of Instruction Set Processor Architecture by Kiminori UTSUNOMIYA (Institute of Information Sciences and Electronics, University of Tsukuba).

†† 筑波大学電子・情報工学系

に要する機能が完備しており，実行速度も高い

（3） オブジェクトコードの実行速度が高い

（4） 実現が容易で，高速処理が可能

（5） オブジェクトコードのサイズが小さい

などの要件を満たすものであるとしている．(1)については，オブジェクトコードの効率と効率化のためのコンパイラの負担との兼合いに注意しなければならない．RISC アーキテクチャではコンパイラによるオブジェクトコードの最適化が前提であるし，スーパコンピュータ用コンパイラもアーキテクチャに合わせて効率よくベクトル化されたコードを生成する任を負っている．RISC やスーパコンピュータでは，そのアーキテクチャをいかすため，他のアーキテクチャに比してオブジェクトコードの最適化がより強く要請されている．最適化の負担をコンパイラに負わせるのが妥当かどうかは問題である．Myers[2] は「良くないアーキテクチャがコンパイラを複雑にしている」という主旨のことを述べている．速度至上主義のスーパコンピュータにはこの主張は当たらない．コンパイラがコードの最適化のためになし得る最大限の努力をして当り前だからである．一方，RISC はそうではない．RISC ではコンパイラの助けを借りることが前提であり，絶対的な速度ではなく，むしろ速度価格比を追求しているからである．しかし，一般論としては，「最適化のためにコンパイラに負担をかけるようなアーキテクチャは悪い」とは言えない．コンパイラは負担を負うべきである．ただし，必要以上に最適化を難しくしたり，セマンティックギャップ（4. 参照）が大きすぎるためコンパイラはこうすれば最適化できると分かっているのにそれを実現させないようなアーキテクチャは明らかに悪い．

(2)は，アーキテクチャをとおして与えられてない機能を必要とするより大きな機能を S/W で実現することはできないことをいっている．たとえば，test and set, swap, exchange のようなマシン命令がなければ lock/unlock 機構は作れない．(2)ではオペレーティングシステムについてだけいっているが，他の S/W についても同様である．

(4)はアーキテクチャが H/W, F/W で具体的に実現しやすいことをいっている．現在では，超 LSI 技術がいかせるアーキテクチャかどうかが一つの大きな判断基準になっている．(3)と(5)についてはあらためて説明する必要もなかろう．

良い命令セットアーキテクチャが満たすべき要件として上記(1)～(5)に次の二つを追加しておきたい．

（6） ソフトウェアの信頼性を支援する

（7） 汎用性が高い

(6)は Myers[2] が強調している要件である．彼は 10 進数を2進化したり，固定長表現することによる誤差の問題，バイト（8ビット）単位に主記憶を割り当てるために生じる符号付き 10 進数（4ビットで1桁）の余分桁の問題をはじめ，アーキテクチャがソフトウェアの信頼性を低下させている多くの例をあげている．

(7)は一つのアーキテクチャでできるだけ広い応用分野の S/W を動かせるほうがよいということである．計算速度がコストを要求するのと同様，汎用性もコストを要求する．同じコスト，同じ速度で多くの応用分野に向くアーキテクチャのほうが限られた応用分野にしか向かないアーキテクチャより良いに決まっている．

また，Wulf[1),3)] はコンパイラからみた良い命令セットアーキテクチャの要件を次のようにまとめている．

（a） 規則性：命令構成の規則性が高く，命令ごとに個別的に決められていないこと

（b） 直交性：命令コード，データタイプ，アドレス方法などが相互依存的に決められていないこと

（c） 組合せ可能性：命令コード，データタイプ，アドレス方法のすべての組合せが使えることなど

（d） 1対全：あることをなす方法はただ1種類だけか，さもなければあらゆるケースすべてについて用意されていること

（e） そのものずばりの解より基本操作：ソースプログラムの概念をそのまま実現するマシン命令より，組み合わせることによって種々の概念を実現できる基本操作が用意されていること

（f） データアクセスのためのアドレス機能が一般性と一貫性をもち，効率が良いこと

（g） プログラム実行時の環境をよく支援すること

上記(1)～(7)は一般的評価基準であるが，立場が異なれば各要件に対する重み付けも異なる．立場を固定すれば要件をより詳細化して議論することが可能になる．本章の残りおよび 4.～6. では，S/W を動かすエンジンとしてのマシンは，究極的には（絶対）速度性能が最優先の評価項目になるとして，速度性能に絞って命令セットアーキテクチャの基本的あり方を議論する．

速度性能を上げるための技術は次の5つに分類することができる.

(i) 高速スイッチング素子／高密度実装：マシンサイクル時間，クロック周期の短縮

(ii) 並行処理：パイプラインなどのストリーム処理，多重プロセッサによる並列処理など

(iii) メモリバッファリング：各種バッファ（キャッシュ），仮想記憶，一元化記憶

(iv) ウェアの強化：S/W の F/W 化，F/W の H/W 化

(v) セマンティックギャップの縮小：命令セットアーキテクチャと S/W の親和性の強化

このうち(i)だけがアーキテクチャに関係しない速度向上技術である．S/W 技術の開発習熟コスト，S/W 製品の資産価値などを考慮すれば，速度は上げて欲しいが，S/W に根本的な影響を与えるアーキテクチャの変更はできるかぎり避けたい．したがって，370 アーキテクチャに代表されるノイマン型マシンでは，(i)と(ii)～(iv)のうち S/W への影響の少ない命令のパイプライン処理，キャッシュ，仮想記憶，ウェアの強化，限られた多重プロセッサなどによる速度向上を図ってきた．この中で，アーキテクチャに関係せず効果も高い(i)がもっとも大きな貢献をした．

しかし，(i)による速度向上には今後多くを望めなくなってきている．370 アーキテクチャの最大機種である IBM 3090 のマシンサイクル時間は 15 n 秒 (15×10^{-9} 秒) まできている．Anacker[4] は，マシンサイクル時間 1 n 秒，2 n 秒のアクセス時間で 256 KB (kilo byte) のキャッシュ，10 n 秒のアクセス時間で 64 MB (mega byte) の主記憶をもつマシンは 10 cm×8 cm×8 cm の空間の中におさめなければならず，IBM 3033 と同じ素子（バイポーラ ECL），同じ作り方 (300,000 ゲート) をした場合その発熱量は 20 kW にもなると計算している（主記憶 8 MB での発熱量）．その速度性能は約 250 MIPS（毎秒 250×10^6 個の命令を実行）である．しかし，現在の技術では 10 cm×8 cm×8 cm の空間から発生する 20 kW の熱を冷却することはできない．すなわち，3033 のアーキテクチャのままで，1 n 秒のマシンサイクル時間，250 MIPS のマシン（単一プロセッサ）をバイポーラ ECL で作ることはできないのである．新素材による素子の開発を期待するとしても，(i)による速度向上はここしばらくは多くを望めない．とすれば，速度向上の主役はアーキテクチャに移らざるをえない．なかでも(ii)の並行処理と(v)のセマンティックギャップの縮小が中心になる．セマンティックギャップは 4. でより詳細に議論する．

3. 定量的評価

MacDougall[5] は 370 アーキテクチャに対する速度性能評価モデルを作っている．彼は，MIPS（マシン命令の1秒当たりの平均実行数の百万単位表現）は

$$\mathrm{MIPS}^{-1}=I*C$$

$$I=E+D+S$$

$$E=\sum e(i)*f(i)$$

$$D=\sum d(i)*f(i)$$

$$S=u*M$$

ただし，C：マシンサイクル時間（10^{-6} 秒単位）

E：実行段階で必要な平均マシンサイクル数

$f(i)$：命令 i の相対実行頻度

$e(i)$：命令 i の実行段階で必要な平均マシンサイクル数，通常は 1

D：平均パイプライン遅れ（マシンサイクル数）

$d(i)$：命令 i の実行が原因で生じる平均パイプライン遅れ(マシンサイクル数)

S：平均メモリアクセス遅れ（マシンサイクル数）

u：主記憶キャッシュのミスヒット率

M：主記憶キャッシュミスヒット時の平均遅れ（マシンサイクル数）

で計算できるとし，事務計算環境での評価モデルを報告している．オペレーティングシステム (MVS/SP 1.3) が全実行の 47% を占めている．MVS/SP 1.3 の $f(i)$, $e(i)$, $d(i)$ を表-1, 2 に示す．

Fuller ら[1),6)] は 1977 年軍用システム選定時に次の3項目について定量的に比較評価している．

S 量：テストプログラムが占めるバイト数

M 量：テストプログラム実行中に主記憶と CPU の間で転送されたバイト数

R 量：テストプログラム実行中 CPU 中のレジスタ間で転送されたバイト数

S 量はプログラムサイズの評価尺度であり，M 量と R 量は速度性能を大きく左右する．IBMS/370, PDP-11, INTERDATA 8/32 の3機種に対して 12 本のテストプログラムで評価している．テストプログラム B による評価結果を表-3 に示す．このプログラムは入

表-1 MVS/SP1.3の命令ミックス f と命令の実行段階平均マシンサイクル数[5]

RANK	OP CODE	f	e	$e*f$
1	BC	0.18532	1	0.18532
2	L	0.14447	1	0.14447
3	TM	0.06289	1	0.06289
4	ST	0.05122	1	0.05122
5	LR	0.04864	1	0.04864
6	LA	0.04645	1	0.04645
7	BCR	0.03035	1	0.03035
8	LTR	0.02903	1	0.02903
9	MVC	0.02226	4.667	0.10389
10	IC	0.01790	1	0.01790
11	LH	0.01765	1	0.01765
12	BALR	0.01647	1	0.01647
13	STM	0.01563	5.543	0.08664
・	・	・	・	・
・	・	・	・	・
・	・	・	・	・
104	SCKC	0.00001	4	0.00004
				$E=\sum e*f$

表-2 MVS/SP1.3の命令ミックス f と命令の平均パイプライン遅れ d (マシンサイクル単位)[5]

RANK	OP CODE	f	d	$d*f$
1	BC	0.18532	1.029	0.1907
2	L	0.14447	0.428	0.0618
3	TM	0.06289	0.408	0.0256
4	ST	0.05122	0.580	0.0297
5	LR	0.04864	0.087	0.0042
6	LA	0.04645	0.313	0.0145
7	BCR	0.03035	1.437	0.0436
8	LTR	0.02903	0.773	0.0224
9	MVC	0.02226	0.982	0.0219
10	IC	0.01790	0.802	0.0144
11	LH	0.01765	0.625	0.0110
12	BALR	0.01647	1.601	0.0264
13	STM	0.01563	0.559	0.0087
・	・	・	・	・
・	・	・	・	・
・	・	・	・	・
				$D=\sum d*f$

出力割込み処理プログラムであり，2人のプログラマ (#2 と #13) が別々に書いている．すべてにおいて INTERDATA 8/32 がもっとも良い値（小さな値）を出している．

Lubeck と Moore[7] は富士通 VP-200, 日立 S 810/20, Cray X-MP/2 のスーパコンピュータ3機種を比較評価しているが，ここでは割愛する．

以上は実存するマシン間でのアーキテクチャの相対的評価であった．Flynn[1),8),9)] は理想的アーキテクチャを想定し，それと実在するマシンのアーキテクチャとの比較評価を試みている．従来型アーキテクチャのマシン命令を以下の三つの型に分類する．

F型（機能型）：データに対して演算する命令

M型（メモリ型）：load 命令，store 命令などのメモリ，レジスタ間でデータ転送しかしない命令

P型（手順型）：分岐など実行順序を制御する命令

これらのうちM型とP型は理想アーキテクチャであれば不要のオーバヘッド（命令）とし，オーバヘッドの割合を測定することによって評価する．いくつかの機種に対して測定した結果例を表-4 に示す．M 比 (M ratio), P比はそれぞれF型命令の使用頻度に対するM型命令の使用頻度，P型命令の使用頻度の割合であり，NF 比（非F型比）はM比とP比の和である．また，ソースプログラムを必要最小限のメモリと処理量で忠実に実行できるマシンを理想マシンとする．そのマシンで実行するときのメモリ量，実行命令数，データ参照回数などを規準解釈量 (Canonic Interpretive Measure) とし，それと実際のマシンでの実行による量との比較によって評価することも行っている．

Flynn が理想マシンとして心に描いているのは高級言語向きマシンである．Myers[2] は具体的な高級言語向きマシンである Burroughs B 1700 と IBM S/360, S/3 との比較結果を指摘している．所要メモリ量に関して，7本のサンプル FORTRAN プログラム全体で S/360 は 560 KB, B 1700 は 280 KB, 31 本のサンプル RPG II プログラムで S/3 は 310 KB, B 1700 は 150 KB であった．速度性能に関しては，S/3 モデ

表-3 Fuller らによる S 量，M 量，R 量を用いたアーキテクチャの比較評価[6]

	IBM S/370	INTERDATA PDP-11	INTERDATA 8/32
S MEASURE			
PROGRAMMER #2	372	133	144
PROGRAMMER #13	465	246	98
M MEASURE			
PROGRAMMER #2	424	208	192
PROGRAMMER #13	920	296	114
R MEASURE			
PROGRAMMER #2	2222	1096	698
PROGRAMMER #13	4583	1419	482

表-4 Flynn による M比，P比，NF 比を用いた理想アーキテクチャと実在アーキテクチャの比較評価[9]

PROCESSOR	"IDEAL"	IBM 7090	IBM/360	DEC-10	PDP-11
M RATIO	0.0	2.0	3.0	1.5	2.6
P RATIO	0.0	0.8	2.5	1.1	3.7
NF RATIO	0.0	2.8	5.5	2.6	6.3

ル 10 で 208 秒要した RPGⅡ プログラムが B 1700 では 25 秒しかかからなかった（性能で 1 : 8，性能価格化で 1 : 5）．

Flynn[10] は RISC と CISC の定量的比較評価も行っている．その結論の一部を 5. で引用する．

ここではいくつかの定量的評価例を紹介したが，結果の数値もさることながら，どのような尺度を使っているか，なぜそれを使っているのかという点に参考になることが多い．

4. セマンティックギャップ

セマンティックギャップ（semantic gap, 意味の隔り）は「高級言語の概念とマシンアーキテクチャ（命令セットアーキテクチャ）の概念との差異の尺度」とされている．Myers[2] は，たとえば PL/I と S/370 のように，セマンティックギャップが大きければ，コンパイラによる橋渡しがうまくいかないため，オブジェクトが必要以上に大きくなったり，不要な速度性能の低下を招いたりするだけでなく，異なる意味に翻訳されてしまうことさえあると指摘している．S/370 アーキテクチャが PL/I のような高級言語用のエンジンとしてお粗末な理由として Myers は次のことをあげている．

（1） 高級言語の（より複雑な）概念を 370 アーキテクチャの単純な記憶構造に写像するために法外な量の S/W を必要とする．概念構造と記憶構造のギャップをプログラム論理で埋めなければならず，そのコード量が膨大になる

（2） 370 アーキテクチャはあまりにも一般的過ぎる．命令とデータの区別はないし，文字ストリングを命令に加えることだって（やろうと思えば S/W で）できる

（3） 370 アーキテクチャの記憶構造はごく単純であるから，命令セットも（機能として）単純なものに限定されている

そして，高級言語用のエンジンとしてのマシンには，タグ付きデータ，ディスクリプタ，演算用スタック，サブルーチン管理機構，字句レベルアドレス，資格アドレス (capability addressing)，可変長記憶セルなどの方式を採用すべきであると主張している．

Myers の主張は，「マシンアーキテクチャをよりユーザインタフェースに近づけろ」ということであり，高級言語マシンはその一つの解である．IBM System/38 もこのような考え方の上に立ったオンラインデータベース指向の事務処理向き商用機である．オブジェクト指向でほとんどの概念が F/W で実現されている（すなわち，H/W による速度性能の向上は図られていない）にもかかわらず，オペレーティングシステムのスループット，データベース管理システムの応答時間は十分満足すべきものになっている．

著者は「セマンティックギャップ」をもう少し広く解釈し，これこそアーキテクチャによる速度性能向上の本質と考えている．すなわち，マシンからみてS/Wを解釈実行するためのセマンティックスには

（1） 機能セマンティックス：S/W が実行を要求している機能についての意味

（2） 実行制御セマンティックス：S/W が実行を要求している機能間の実行順序についての意味（データ間の依存関係で決まる）

（3） データ構造セマンティックス：データ間の構造的関係についての意味

（4） 記憶空間セマンティックス：機能，データの空間的配置についての意味

の 4 つがあり，アーキテクチャ的速度性能はこれらのセマンティックスをどの程度オーバヘッドなくマシンアーキテクチャに写像し，実行できるかによって決まる．主として 2. の速度性能向上技術の（ⅱ）はこの（2）に，（ⅲ）は（4）に，（ⅳ）は（1）に，（ⅴ）は（3）と（1）にそれぞれ対応している．

S/W のセマンティックスは S/W ごとに異なるため，アーキテクチャを S/W に近づけてセマンティックギャップを詰めれば，マシンの汎用性は落ちる．汎用性と高速実行は両立しない．「より速く」を望むのであれば究極的には汎用性を犠牲にしなければならない．従来型アーキテクチャの主流である S/370 は 1964 年に発表された S/360 と基本的にまったく同じアーキテクチャである．S/360 の 360 は全方向を意味する 360° に因んでいる．S/360 は汎用性重視であった．基本的には，「多種類のことができる一つのマシン」の時代から「少種類のことしかできない多数のマシン」の時代へ向かわざるをえない．今後は多くのアーキテクチャが共存する時代になる．

5. RISC 対 CISC

RISC (Reduced Instruction Set Computer) と CISC (Complex Instruction Set Computer) の優劣に関する論争が盛んである．Tabak[11] は RISC の要件を次のようにまとめている．

表-5 Tabak の RISC 要件 (1)～(8) とマシンごとのその要件の満足 (S)/不満足 (V)[11]

System	Property							
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
RISC II	S	S	S	S	S	S	S	S
MIPS	S	S	S	S	S	V	S	S
801	S	S	S	S	S	S	S	S
Miris	S	S	S	S	S	S	V	S
Pyramid	S	V	S	V	S	S	V	S
Ridge	S	S	S	V	S	V	V	S
Acorn	S	S	V	S	S	V	S	S
INMOS	S	S	S	S	S	V	V	S
RT PC	S	S	S	V	S	V	V	S
HP	S	S	S	S	S	S	S	S
MF 1600	S	S	S	V	S	S	S	S
Clipper	S	V	V	S	S	V	S	S

(1) マシン語命令の種類が少ない

(2) アドレッシングモードが少ない

(3) 命令形式が少ない

(4) 多くの命令の実行が1マシンサイクルで終わる

(5) load/store 命令だけがメモリをアクセス

(6) 汎用レジスタの個数が多い

(7) 配線論理制御

(8) サブルーチン呼出し／戻りなどの高級言語支援

いくつかのマシンに対してこの基準の満足／不満足を調べた結果を表-5 に示す．これに呼応して Furht[12] は次のように CISC 要件を決め表-6 を作った．

(1) マシン語命令の種類が多い

(2) アドレッシングモードが多い

(3) 命令形式が多い

(4) 多くの命令の実行マシンサイクル数が2以上

表-6 Furht の CISC 要件 (1)～(8) とマシンごとのその要件の満足 (S)/不満足 (V)[12]

System	Criteria							
	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
Motorola 68020	S	S	S	S	S	S	S	S
Intel 80386	S	S	S	S	S	S	S	S
Fairchild Clipper	S	S	S	V	V	V	V	V
Zilog Z 80,000	S	S	S	S	S	V	S	S
AT&T WE 32100	S	S	S	S	S	V	S	S
Hewlett-Packard Focus	S	S	S	S	S	S	S	S
NS 32000 Series	V	S	S	S	S	V	S	S
DEC VLSI VAX	S	S	S	S	S	S	S	S

(5) メモリにアクセスする命令がいろいろある

(6) 特定目的のレジスタがある

(7) マイクロプログラム制御

(8) 比較的高級言語の概念レベルに近いマシン命令が多い

RISC の動機は超 LSI を生かしながら速度性能を上げたいということである．3. の MacDougall のモデルを用いると単一プロセッサの MIPS は

$$MIPS^{-1}=(E+D+S)*C$$

で表せる．MIPS を大きくするにはこれら C, E, D, S を小さくすればよい．Taback の RISC 要件(1)～(3)は C を小さくすることに貢献している．E の最小値は1であるが，RISC では $E=1$ であることを要件(4)（および(7)）が保証している．要件(5)と(6)によって S を小さくしている．分岐命令やデータ間インタロックなどのため D は0にはならないが，RISC では遅延分岐などコンパイラがパイプライン制御の特性をいかす最適化を行うことによって D をできるだけ0に近づける努力をしている．従来の

表-7 RISC I でのCプログラムのベンチマーク（左の列は RISC I によるミリ秒単位の実行時間，68000 の列より右は何倍遅いか）[11]

BENCHMARK	NUMBER OF TIMES SLOWER THAN RISC I					
	RISC I	68000	Z8002	VAX-11/780	11/70	C/70
E—STRING SEARCH	.46	2.8	1.6	1.3	0.9	2.2
F—BIT TEST	.06	4.8	7.2	4.8	6.2	9.2
H—LINKED LIST	.10	1.6	2.4	1.2	1.9	2.5
K—BIT MATRIX	.43	4.0	5.2	3.0	4.0	9.3
I—QUICKSORT	50.4	4.1	5.2	3.0	3.6	5.8
ACKERMAN (3,6)	3200	—	2.8	1.6	1.6	—
RECURSIVE QSORT	800	—	5.9	2.3	3.2	1.3
PUZZLE (SUBSCRIPT)	4700	—	4.2	2.0	1.6	3.4
PUZZLE (POINTER)	3200	4.2	2.3	1.3	2.0	2.1
SED (BATCH EDITOR)	5100	—	4.4	1.1	1.1	2.6
TOWERS HANOI (18)	6800	—	4.2	1.8	2.3	1.6
AVERAGE		3.5±1.8	4.1±1.6	2.1±1.1	2.6±1.5	4.0±2.8

CISC 用コンパイラはそこまではやっていない．RISC ではマシン命令が基本的なものだけに限られているのでサブルーチンの使用頻度が高くなるが，要件(8)と(6)（レジスタウィンドウ）がそのオーバヘッドを少なくしている．

RISC の原理は上手な単純化と高規則化であり，それによって，マシンサイクル時間が詰まりパイプラインの効率が良くなるとともに，超 LSI による実現コストも下がるので一石二鳥の効果が得られるというものである．もちろん最適化の負担をコンパイラにかけることになるが，Wulf のコンパイラの立場からの良い命令セットアーキテクチャの要件，とくに(e)と(d)を根拠に RISC 派はコンパイラにとっても RISC 優位を主張している．

Patterson ら[13]は彼らが開発した RISC I といくつかの CISC との速度性能を比較し表-7 のように報告している．RISC 派は，RISC は速く，安く，しかもコンパイラにとっても有利と主張する．CISC の立場は一見なさそうにみえる．しかし，4. で述べたように著者は速度性能は S/W のセマンティックスとアーキテクチャの一致度で決まると考えているので，速度性能を追求するかぎり CISC アプローチのほうが優位との説をとっている．もちろん Tabak の要件すべてではなく

(1) 命令の種類は多くてよい

(2) マイクロプログラム制御でもよい

(3) レジスタメモリ命令を採用する

だけの否定を行い，残りはできるかぎり RISC 要件にそった CISC とする．多くの命令の実行が1マシンサイクルで終わるという要件(4)もそれほど気にしなくてよい．海野[14]によればマイクロプログラム制御の CISC である IBM プロセッサ 3033 でも事務計算時に 87% は1マシンサイクル命令である．Radin[15]は CISC の垂直マイクロプログラムと命令用キャッシュをもつ RISC の S/W とは差がないと言っているが，ミスヒットがない，分岐で遅れが生じないという点で垂直マイクロプログラムのほうが優れている．Flynn ら[10]は load/store 命令でしかメモリアクセスしない RISC よりレジスタメモリ命令を採用したマシンのほうが命令キャッシュの効率が良いこと，一般にレジスタ数を増やせばメモリアクセスは減るが，それに合わせて命令も上手にコード化しなければ（すなわち，基本的な命令に限るのであれば）レジスタを増やす効果は小さいことを指摘している．後者は，S/W の記憶空間セマンティックスは結構大きく，そのセマンティックスに十分なレジスタを用意するのであれば，機能セマンティックスもより大きなローカリティでとらえることができ，より大きな機能粒度の命令でデータを処理したほうが効率的であることを示している．RISC に投げかけられた疑問に対する Patterson[13]の反論は根拠不統一な点が多いように思う．Colwell ら[16]も同じような見解である．RISC I はC言語環境を想定しており，多言語環境を想定している CISC との速度比較（表-7）をそのまま鵜呑にはできない．4. で述べたように汎用性と速度性能は互いに排他的であるからである．

現在 CISC といわれているマシンは RISC ほどの確固たる哲学をもって設計されてきたわけではない．RISC はアーキテクチャの設計をもう一度根本的に見直すという点で非常に良い影響を与えている．CISC も RISC の厳しさを見習うべきである．そして，RISC, CISC に固執せず，両者の良さを素直に吸収し合うべきである．Colwell らも主張しているように，コンピュータシステムの設計はシステムの機能をどの実現レベルに割り当てるかに焦点を置いて行うべきである．個々の S/W について 4. で述べた(1)～(4)のセマンティックスを精一杯引き出すことと，それらのバランスよい共通化が鍵である．超 LSI の集積度は今後も向上する．目的を明確にし，RISC から出発して慎重に CISC 化していく戦略がよい．特殊目的のマシンであれば限られた命令セットで十分なこともあろう．

6. 並列処理の問題

プログラムカウンタ制御のマシンでは，S/W が並列実行のセマンティックスを有していても，それを引き出して速度性能を上げることはできない．データフロー制御／リダクション制御をマシン，あるいは関連し合うマシン系(複数プロセッサ系)のどこかに取り入れなければならない．データフロー制御／リダクション制御はデータの依存関係にもとづいた実行制御セマンティックスを受け入れることができるのでプログラムカウンタ制御より優れている．しかし，並列実行の効率は並列に実行する機能の粒度と記憶空間セマンティックスのローカリティにも左右される．並列処理系においてはメモリの分割とプロセッサ／演算ユニットとの接続が本質的課題である．すべての命令やデータがどのプロセッサ／演算ユニットからでもただちにア

クセスできるようになっていることが理想であるが，それではメモリの速度が足りなくなる．それゆえ，メモリを分割することになる．しかし，最適の分割は個個の S/W のセマンティックスで決まる．セマンティックスに合わない分割はオーバヘッドでしかない多大なメモリ転送を引き起こす．3. の Flynn の評価のオーバヘッド命令とまったく同じ破目になる．並列処理系の性能はメモリアーキテクチャ，接続アーキテクチャにかかっている．

7. おわりに

マシンには第一義的に速さが求められる．アーキテクチャ的には S/W のセマンティックスとの差が少ないマシンほど高速に動作する．命令セットもセマンティックスに合わせて注意深く決定する必要がある．今後はコンパイラ，プログラミングとアーキテクチャの関係はますます密接にならざるをえない．アーキテクチャによるマシンの高速化はすべてセマンティックギャップに関わっている．セマンティックギャップを縮小すれば汎用性が落ちるが，それを救うにはユニバーサルホストプロセッサ[17]などダイナミックマイクロプログラミングが役立つ．

素子の速度に頼りにくくなりつつあり，アーキテクチャが頑張らなければならない．S/W がもつセマンティックス以上の速度性能は引き出せないが，現在のアーキテクチャは S/W のセマンティックスを受け入れにくくなっている．この不一致によるオーバヘッドが非常に大きい．370 アーキテクチャにしろ，非ノイマンアーキテクチャにしろ，速度向上のキーはメモリにある．また，1. で述べた評価項目(3)～(6)がアーキテクチャにより直接反映されるようになる．したがって，命令セットアーキテクチャは今後も変化せざるをえないし，多様化するに違いない．

参考文献

1) 飯塚 肇，田中英彦編：ソフトウェア指向アーキテクチャ，オーム社 (1985).
2) Myers, G. J.: Advances in Computer Architecture, John Wiley & Sons (1978), 渡辺勝正による邦訳（コンピュータ・アーキテクチャの設計，共立出版，1981）あり.
3) Wulf, W. A.: Compilers and Computer Architecture, Computer, Vol. 14, No. 7, pp. 41～47 (1981).
4) Anacker, W.: Josephson Computer Technology: An IBM Research Project, IBM J. Res. and Dev., Vol. 24, No. 2, pp. 107～112 (Mar. 1980).
5) MacDougall, M. H.: Instruction-Level Program and Processor Modeling, Computer, Vol. 17, No. 7, pp. 14～24 (July 1984).
6) Fuller, S. H. and Burr, W. E.: Measurement and Evaluation of Alternative Computer Architecture, Computer, Vol. 10, No. 10 (Oct. 1977).
7) Lubeck, O. and Moore, J.: A Benchmark Comparison of Three Supercomputers: Fujitsu VP-200, Hitachi S 810/20, and Cray X-MP/2, Computer, Vol. 18, No. 12, pp. 10～24 (Dec. 1985).
8) Flynn, M.: Directions and Issues in Architecture and Language, Computer, Vol. 13, No. 10, pp. 5～22 (Oct. 1980).
9) Flynn, M. J. et al.: Performance Evaluation of the Execution Aspects of Computer Architectures, Proc. of Int'l Workshop on HLL Computer Architecture, pp. 97～104 (Nov. 1982).
10) Flynn, M. J., Mitchell, C. L. and Mulder, J. M.: And Now a Case for More Complex Instruction Sets, Computer, Vol. 20, No. 9, pp. 71～83 (Sep. 1987).
11) Tabak, D.: Which System Is a RISC, Computer, Vol. 19, No. 10, pp. 85～86 (Oct. 1986).
12) Furht, B.: A Definition of Complex Instruction Set Computer (CISC) Architecture, Computer, Vol. 20, No. 5, pp. 108～109 (May 1987).
13) Patterson, D. A. and Sequin, C. H.: A VLSI RISC, Computer, Vol. 15, No. 2, pp. 8～21 (Sep. 1982).
14) 海野三郎：IBM 3033 プロセッサの内部設計とパフォーマンス，日経エレクトロニクス別冊コンピュータその技術動向とソフト／アプリケーション技術 (1978).
15) Radin, G.: The 801 Minicomputer, IBM J. of Res. and Dev., Vol. 27, No. 3, pp. 237～246 (May 1983).
16) Colwell, R. P. et al.: Computers, Complexity, and Controversy, Computer, Vol. 18, No. 9, pp. 8～19 (Sep. 1985).
17) 飯塚 肇：ユニバーサルホストプロセッサ，bit, Vol. 12, No. 10 (ダイナミックアーキテクチャ), pp. 109～130 (Aug. 1980).

(昭和63年10月14日受付)