第８回　ハーバードアーキテクチャ、制御アーキテクチャ　ワイヤード vs マイクロプログラム

［１］　ハーバードアーキテクチャ

［補足説明］ 教科書 4.1 のハーバードアーキテクチャは、教科書の説明にあるように、CPU とメモリとの間のやりとりを、命令読出しと、データの読書きとで、別の2つの経路を設ける考え方である。今まで見てきたノイマン型アーキテクチャでは、これら2つは同じ経路を経由していた。（つまり、CPUから見てメインメモリは1つだけであり、1つだけの主記憶に命令もデータも載せる、というイメージである。）

現状での実態は、図 4.2 にあるように「キャッシュ」を2種類別々に設けるが、メインメモリ自体は1個だけ、という状況にある。この違いについては、後でキャッシュの性能を議論するときに関係してくる。 これが現在ハーバード的な構成を取る最大の理由である。

a.　（教科書 4.1） ハーバードアーキテクチャについて、正しい記述には○を、間違っている記述には×を付けよ

① ハーバードアーキテクチャは、メモリから命令を取り出す経路とデータを取り出す経路を分離しているので、CPUとメモリの間の転送が高速化され、フォンノイマンのボトルネックが回避される傾向にある

② ハーバードアーキテクチャは、命令をメモリに置かないので、ノイマン型コンピュータとは呼べない。

③ ハーバードアーキテクチャは、命令もデータもメモリに置くことには変わりないので、広義のノイマン型コンピュータに含める

④ ハーバードアーキテクチャは、現在ではメモリは単一とし、その手前のバスとキャッシュが分離されている形で使われている場合が多い。

［２］　ＲＩＳＣとＣＩＳＣ

RISC、CISC という略称のフルネーム（フルスペル）と、それの意味は、教科書 4.2 で読むこと。

a.　次の RISC と CISC についての記述で、正しいものには○を、間違っている記述には×を付けよ

① RISC は「縮小命令」と呼ぶとおり、個々の命令が短くなっており、CISC はそれに比べて個々の命令が長い。

② RISC は個々の命令が単純で少ない機能を持っているのに対し、CISC は個々の命令が複雑な機能を持っている。

③ 同じことをするのに、RISC だとたくさんの命令を使って処理しなければならないが CISC だと少ない命令でできることがある

④ RISC は命令が簡単なので実現するためのハードウェアが簡単になるのに対し、CISCI はハードウェアが複雑になる傾向がある。

b.　RISC の命令セット（1つのCPUが持つ命令の集合）と、CISC の命令セットを、具体的に比較してみる。RISC プロセッサの代表である MIPS の R シリーズCPUやルネサステクノロジーの SH-4A（組み込み CPU としてよく使われる）では、全ての演算（加減乗除など）命令がレジスタ間のみであり、加減算などで片方のオペランドをメモリから読み込むことはできない。

（SH-4A のマニュアルは、http://documentation.renesas.com/jpn/products/mpumcu/rjj09b0090_sh4a.pdf　参照）

それに対して、CISC の代表である Pentium や、教科書で扱った COMET II では、レジスタ間の演算のほか、1つのオペランドをメモリから読み込むレジスターメモリ変数間演算ができる。 では、これらの CPU によって z = x + y + z を計算すると、いくつ命令が必要になるだろうか？　COMET II でのプログラム例を参考にして、考えてみよ。 途中結果はなるべくレジスタに残して使うこと。最終結果はメモリ上の w へ格納すること。

① RISC では8命令なのに対し、CISC では6命令で済む

② RISC では6命令なのに対し、CISC では4命令で済む

③ RISC では4命令なのに対し、CISC では2命令で済む

④ RISC では3命令なのに対し、CISC では2命令で済む

c.　RISC と CISC の性能について、正しい記述には○を、間違っている記述には×を付けよ

① RISC は、命令が簡単で同じことをするのにたくさんの命令が要るので、命令ステップ数が増える。それに対して

CISC では、命令ステップ数が少なくて済む。

② RISC は個々の命令が単純なので、ハードウェアが単純になるため、1段あたりのハードウェア遅延が小さくなってクロックが速くできる。それに対して CISC は個々の命令が複雑なので1段あたりのハードウェア遅延が長くなりクロックが速くし難い。

③ ①と②から、RISC は CISC に比べてプログラムの実行が速い

④ ①と②から、RISC は CISC に比べてプログラムの実行が遅い

⑤ RISC は汎用レジスタが多く、中間結果を汎用レジスタに多数保持できるので、メモリに書き戻す必要が少なくなる。CISC では汎用レジスタが少なく、メモリ上に書き戻す必要がある。

⑥ RISC はハードウェアの構造が簡単であるので、製造が容易になり、また IC チップ面積が小さいので歩留まりもよくなり、製造コストは CISC に比較して安価になる。

⑦ これらから見て、一般に RISC の方が処理速度が速くなり、かつ構造が簡単で製造も容易なので、現在使われているプロセッサはほとんど RISC である。

⑧ これらから見て、一般に CISC の方が最終的な処理速度は速くなるので、現在使われているプロセッサはほとんど CISC である。

［補足］ 教科書第4章の最後に書かれているように、RISC と CISC の絶対的な比較は難しい。だから⑦と⑧はいずれともつけがたい。 また、最近では元々CISC として設計された命令セット（1つのCPUが持つ命令の集合）を、RISC の構造を使って実現する（CPU 内部で RISC 命令に置き換える）ことも行われている（Intel Pentium 系のプロセッサが一例）。

［補足］ RISC が登場したのは 1980 年ごろであるが、その時代はメインフレームやワークステーションといった大きな図体をもつプロセッサから、LSI による 1 チップのプロセッサに移行する途中の時期であった。最初に広く普及した1チップ LSI による 8 ビットプロセッサの Intel 8080 と Motorola 6800 が発表されたのが 1974 年であり、16 ビットプロセッサの Intel 8086 が 1978 年に発表、Motorola の 68000 が 1979 年にサンプル出荷されている。当時は LSI に積載できるトランジスタ数がどんどん増加していたが、まだまだ厳しい制限があった。特に内部レジスタやバスのビット数を8⇒16⇒32と増やすと、それだけでトランジスタ数が比例して増えるし、一方でより複雑な機構（当初はたとえば浮動小数点演算回路をチップ内に取り込みたかったり、後述のパイプライン制御を組み込んだり、また 80386 の時代には仮想記憶のための制御回路も必要になった）を組み込みたい欲求もあった。その点で回路を簡単化する欲求があったとも思われる。 それ以降、難しいと言われた高集積がどんどん実用になり （たとえば最近の Core2Quad の Yorkfield で 4 億 1 千万トランジスタ）、結局は回路を簡単化してプログラムの命令数を多くするという RISC の方針は、直接は使われなくなった。もちろん、そこで生まれたさまざまな考え方・技術は、大いに生かされている。

［３］ 制御アーキテクチャ、ワイヤードロジック方式とマイクロプログラム方式

a. 教科書 6 章の制御アーキテクチャを参考にして、次の中から間違っているものを選べ

① コンピュータの制御の方式としてワイヤードロジック方式とマイクロプログラム方式があり、ワイヤードロジック方式は命令の解釈・実行を素子間の結線によって制御するのに対し、マイクロプログラム方式は命令の解釈・実行を（マイクロ）プログラムによって制御する

② ワイヤードロジック方式はLSI回路の集積度の上限から余り複雑な命令が作れないのに対し、マイクロプログラム方式では原理的には命令の複雑さに上限が無い

③ ワイヤードロジック方式は結線によって制御するので高速にし難いが、マイクロプログラム方式は 1 つの命令を実行するのにその中を更にプログラムで実現するので比較的に高速に動作する

④ ワイヤードロジック方式は結線によって制御するので簡単には変更できないが、マイクロプログラム方式はマイクロプログラムを書きかえることによって同じハードウェアを使ってどのような計算機でも作ることができる万能の設計技術である

b. CISC と比較したときの RISC の特徴として，適切なものはどれか。（基本情報 15 春 問 18）

	命令長	ハードウェアの制御	演算の対象
①	固定	主にマイクロコード制御	メモリ, レジスタ
②	固定	ワイヤードロジック制御	レジスタ
③	可変	主にマイクロコード制御	レジスタ
④	可変	ワイヤードロジック制御	メモリ, レジスタ

［補足］ マイクロプログラム方式が広く使われた例として、IBM System/360とDEC VAXコンピュータファミリーが挙げられる。これらはいずれも、「ファミリー」と呼ばれる性能・価格が異なる幅広い一連の製品から成り、見かけ上同じ構造・同じ命令体系を実現し、1つのプログラムをどの機種でも実行できる。高位の機種ではハードウェアをふんだんに使ったハードワイヤードロジック方式を用いて高性能を得るのに対し、低位機種では比較的簡単なハードウェアとマイクロプログラムによって、低価格ながら見かけ上同じハードウェアを持っているように見える。ファミリーの考え方は、ソフト開発の軽減や業務拡大に伴う機種アップグレードの容易さから、広く受け入れられた。

［補足］ 教科書 6.2.2 の命令実行時の動作の（1）LD 命令の部分を読んで、書かれていることが理解できるようにするとよい。

c.　教科書 6.3.2 「マイクロ命令の形式」 において説明されている 「水平型」「垂直型」 マイクロプログラムについて、次の中から間違っているものを選べ

① 水平型マイクロプログラムは、ハードウェアのゲートに対する制御入力信号のビットパターンをそのままマイクロ命令のビットパターンとしたものであり、マイクロ命令の各ビットを直接制御入力に接続して使う。他方、垂直型マイクロプログラムは、上記のビットパターンを出現パターンに基いて符号化してマイクロ命令のビットパターンとしたもので、マイクロ命令をデコードしてゲート制御信号を作り出す必要がある。

② 水平型マイクロプログラムは、一般にマイクロ命令の長さが長くなるのに対し、垂直型マイクロプログラムは短くなる

③ 垂直型マイクロプログラムは、命令の長さの制限から複雑なハードウェア論理が作れないのに対し、水平型マイクロプログラムはそれぞれの制御ゲートに対応したビットを命令に含むので複雑なハードウェア論理を作ることができる。

④ 垂直型マイクロプログラムは、デコードが必要なので信号の伝播遅延が発生しクロックが遅くなるのに対し、水平型マイクロプログラムは速くできる

d.　マイクロプログラム制御方式に関する記述のうち、適切なものはどれか。（応用（第1種）14 年春　問 17）

① RISC プロセッサで多く用いられている。

② 機能の追加や変更がマイクロプログラムの変更だけで行えるので、命令の追加及び変更が容易である。

③ マイクロプログラムは一般のプログラムと同様に主記憶装置に格納され、プロセッサが読み出して利用する。

④ ワイヤードロジック方式と比較して、機械命令の処理が高速である。

［補充問題］ 時間と体力に余裕のある学生は、次のような挑戦をしてみるとよい。
たとえば COMET II を取上げよう。 このプロセッサの動きを、Java のプログラムで真似する（シミュレートする）ことが課題である。具体的には、次のように考えるとよい。

ア） CPU内のレジスタ （GRn、FR、命令レジスタ、プログラムカウンタなど）、メモリ （0 番地から適当な最大容量番地までの配列） を、Java の変数として宣言する。

（注： レジスタ・メモリとも本当は 16 ビットだが、面倒なので 32 ビットの整数で取ってしまってよい。上半分は使わない）

イ） 命令をメモリから読み出して命令レジスタへ入れ、命令レジスタの内容を解釈し、それに従って実行し、プログラムカ

ウンタを1進める、というサイクルを、ループで書く。キーボードでたとえばSと押すとステップ実行することにすればよい。

ウ） 1サイクル実行するごとに、レジスタの内容、メモリの内容を表示する

エ） アセンブラ（アセンブリプログラムから機械命令列への変換プログラム）を書くのはそれなりに面倒なので、初めは他のシミュレータに付いているアセンブラで変換した 0/1 の機械命令列をセットすることにする。

オ） 命令の解釈（デコード）は、COMET II の資料 http://www.jitec.jp/1_00topic/topic_20081027_hani_yougo.pdf の１１ページ以降を参照する。具体的には、先頭の4ビット（主 OP）を抜き出して、その値が 0, 1, 2, 3, … のどれか、更に次の4ビット（副 OP）が何であるか、というように分岐して（Java の Switch 文が使える）、決めてゆけばよい。

カ） 演算のオーバーフロー等は、整数型変数 32 ビットのうち 16 ビットしか使わないので、17 ビット目がオーバーフローに当たる。

≪解答≫

［1］ a. ① ○ ② × ③ ○ ④ ○

④については、たとえば最近の Pentium 系プロセッサでは、レベル１キャッシュが命令・データを分離、レベル2キャッシュ・レベル3キャッシュおよびメインメモリは合併になっている。

［2］ a. ① × ② ○ ③ ○ ④ ○

①は成り立たないケースがままある。 たとえばオペランドを必要としない命令（たとえば RET 命令）は、CISC の場合は OP 部だけなので短くなり（たとえば1バイト）、RISC の場合は他の命令に合わせる為に長くする（たとえば4バイトにそろえる）ので、逆転することもある。

②は正しい。 ③は正しい。次の問題c）を参照。④は元々RISC が望んだ方向である。

b. ②

RISC ではレジスタ間の演算しか認めないとすると、x, y, z それぞれをメモリからレジスタにロードするために3命令、足し合わせるのに2命令、結果をメモリに書き戻すのに1命令を要するので、合計6命令になる。 他方 CISC の例として COMET で見ると、x をレジスタにロードし（1命令）、それにメモリ中の y を直接加え（1命令）、更にその結果にメモリ中の z を直接加え（1命令）、最後に結果をメモリに書き戻す（1命令）ので、合計で4命令になる。

c. ① ○ ② ○ ③ どちらとも言えない ④ どちらとも言えない ⑤ ○ ⑥ × ⑦ × ⑧ ×

①②はそれぞれ正しい。

③と④は、プログラムの実行速度は （実行する命令の数）×（1つの命令の実行時間） で決まるが、RISC だと命令数が大きくなって1つの命令の実行時間が小さくなるから、どちらとも言えない。

⑤は正しい。 ⑥は前半は正しいが、最後の製造コストは何とも言えない。現実には、出荷台数が CISC の Pentium 系がけた違いに多いので、安価になっていると思われる。

⑦は、結論の「現在使われているプロセッサがほとんど RISC」は誤り。

⑧は、「一般に CISC の方が最終的な処理速度が速くなる」が誤り。

なお、［補足］にも書いたように、現状は見かけ（命令体系）は CISC だが、内部（実行の仕組）は RISC の考え方と CISC の考え方が融合（折衷？）したような形態の Pentium 系プロセッサ（PC 用の Intel 7i や 5i、サーバ用の Intel Xeon など）が使われている。

［3］ a. ③が間違い。（ワイヤードに比べてマイクロプログラムは低速になる）

ワイヤードロジックの決定的な利点は高速性であり、欠点は設計の困難さと回路規模が大きくなってチップ内に収容しづらくなることである。但し、最後の回路規模の問題は LSI 技術の発展に伴い、かなり複雑な回路を作ることができるようになったので、あまり問題視されない。それに対して、マイクロプログラムの決定的な利点はその自由度にあり、要するにマイクロプログラムを作り変えれば容易にどのような命令体系でも実現できるし、また設計上のミスがあっても修正が容易である。しかし処理性能は一般に（プログラム化したことによって）遅くなる。

b.　②

RISC の命令は比較的動作が簡単であり、ワイヤードロジックでも容易に実現できる。また、演算対象はオペランド長が長くなることやメモリとのやり取りの時間が不定なことを嫌って、レジスタのみとしている場合が多い。

c.　③

マイクロプログラムにおける水平・垂直の違いは、授業では取上げないが（まあどうでもいい話題と思う）、一応知っておこう。

①の説明にあることが水平・垂直の違いである。要するに普通の機械命令のOPコードと同じような形に書いてあるのが垂直、そうではなくて、機械内のどの信号を出すかを1つ1つ丹念に書いたのが水平、と思えばよい。

②と④は正しい。

d.　②

なお、③は、マイクロプログラムは通常はマイクロ命令専用の（高速）メモリ （「コントロールストア」と呼ぶこともある） に書かれている。 RAM の場合は電源投入時に書換え可能であるし、ROM の場合は出荷時に固定される。また消去可能な ROM （EPROM、EEPROM、フラッシュメモリなど） に置かれていれば、一定の条件下（特定の端子に電圧を加えるなど）でアップデートが可能である。 殆どの場合、これらは主記憶の一部ではない。