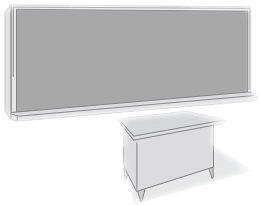
基般

解説 対話で教える コンピュータアーキテクチャ



都倉信樹　大阪電気通信大学

アーキテクチャ教育の困難を感じませんか？

コンピュータサイエンス（CS）の中心的科目である，「**アーキテクチャ**☆1」や「**OS**」の講義の現状はどうだろうか．教えにくくなっているのではないか．その理由はいくつかある．コンピュータの出現からすでに60年余が経過し，アイディアが出尽くし，標準的アーキテクチャが固まってしまった．本来この種の分野では種々な方法を考えて，そのときの技術で最も合理的なものを比較検討して選択することが重要である．しかし，講義学習の時間は十分取れない制約がある．また，欧米の大学教科書は一般に非常に分厚く内容豊富で説明が丁寧で分かりやすいものが多い．しかし，日本では，薄い教科書が求められ，その記述は標準的な1つの方式を紹介して終わることが多く，説明も少なくあまり分かりやすくはない．なぜその方式が考えられ，主流になったのかなど何も分からない．この教材の「空間的制約」のもと，一通りの標準的な方法だけが講義され，学生はわくわく感など感じることはない．いろいろな代案の中からそのときどきの目的や技術水準に応じて最適な選択をしてきたことを知り，今後新しい課題に自由に発想し，それを冷静に評価するという方法，発想を身に付けてほしいし，学生が興味を持っていろいろ考え始めるにはどうしたらいいかという問題意識で本稿を書いた．

☆1 アーキテクチャ．最近はソフトウェアやシステムのアーキテクチャという言葉遣いもされるようになったが，ここでは，コンピュータあるいはコンピュータシステムのアーキテクチャの意味である．

❑ 新しいアーキテクチャ教育教材

すでに飽和した感のあるコンピュータシステムの教育に，学生が興味を持っていろいろ考え始めるような材料を提供したいと考えた．そこで，メモリに機能を埋め込む機能メモリという発想でなにが可能かを考えていき，それを次々展開していろいろな課題を提出する．だんだん考えは違う対象にも及んで，新しいアーキテクチャへ考えが及ぶという構成を考えた．その提示方式として対話形式を採用する．教材をどういうスタイルで提示するかに悩むところであるが，積極的な卒研生，あるいは修士の学生（●）と筆者（◆）の対話形式として記述することを選択している．

教材の作成というと，教科書（本文，演習問題，補足材料など），演習問題・回答集，プレゼン素材，試験問題，採点基準等々多くの文書を作成することと考える方もおられよう．ここではそれらを全部揃えてみせることはできない．むしろ，教材の作成の際にどういうストーリーを立てるか，いわば設計段階をお見せしたい．

❑ ストーリーについて

筆者はこれまでいくつかのテーマで，教材としてストーリーを立てて使ってきた．たとえば，自動改札機の秘密，バーコードの秘密（郵便局の使うバーコードも含む）．これらは，普通何気なく見ているものから出発し，なんのために，どのように，というような疑問を発することから始める．1つの謎が解けると，次の謎が浮かんでくる．できあがったも

のの仕組みを単に説明するのではなく，どうしてそういう発想が生まれ，そのためにどういう技術を使ったかなどを結果的には伝えるのだが，できるだけ謎ときの連続になるように構成する．こういう謎は筆者だけでなく，多くの人がやはり興味を持ってくれる．こうやって，ある対象について，つぎつぎと疑問を持って調べて，仮説を立てる．どうしてもその仮説が自分の集めた証拠では確認できないこともある．実は，これらのストーリーは放送大学の情報工学という TV 講義で使って，その現場ロケを挿入することもあった．そのとき，ロケ先で担当者に聞いて謎を解決することもできた．こうしてだんだん補強し膨らませていく．聞き手に疑問を次々投げかける素材は，学生に考えさせる講義になる．聞き手も退屈せずに聞いてくれるし，インタラクティブな講義にもなる．

ある数学者と話をしたとき，ストーリーを感じさせる数学の名著が少なくないと言われ，いくつか例もご教示いただいた．少し専門的なので，ここでは最近見つけた本，『清水健一：大学入試問題で語る数論の世界』[1] を例として挙げる．入試問題から出発し，数論の中でのどういう問題かを述べ，少し条件を変えると未解決な問題となるという例を多数挙げてあり，実に興味深く最後まで読ませる力のある数学の本である．

筆者のストーリーは，放送大学の TV 講義，面接授業で放送大学学生に，出前講義で高校生に，鳥取環境大学の学部講義，あるいは，大阪大学の大学院の講義でというようにいろいろな機会に使った．もちろん，細部は講義の目的や聴き手のマチュリティ，時間に応じて，表現を変えたり数式の扱い，あるいは取り上げる疑問を変えたりはするが，基本的な謎解きはまったく同じものが使える．そして，ストーリーがしっかりできていれば，最後まで興味を持って付き合ってくれることも確信できた．ただ，これらのストーリーは私の頭の中にあり，文章で書くと長くなるし，あるレベルに固定した記述になる．これを外部に取り出して見せる方法はないかと模索してきて，今回試すのが対話形式の記述である．二者の間でのやりとりの中で，相手が考える部分は細かく記述する必要がなくなり，ストーリー部分を取り出しやすい．

昔語りから対話は始まった

◆ 70 年代，ミニコン pdp11（ピーディーピー イレブン）を知ってしばらくして思いついたアイディアが今回の話の発端なんだけど...

● 先生，待ってください．**ミニコン**，**pdp11** って聞いたことありません．

◆ え？（昭和も遠くなりにけり！［表紙がすれ切れかかかった pdp11 のハンドブック[2] を取り出し丁寧に説明を始める］）pdp11 はユニバスという共通バスを持ち，メモリも周辺機器も一様にアクセスできる仕組みになっていた．なお，メモリは 16 ビット 1 ワード（2 バイト 1 ワード）で，バイトアドレスがついている...

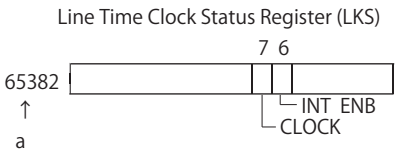
● それ，ハンドブックに書いてあるのですね．

◆ そうです．この小さいハンドブックはインターネットでも読めます[2]．アーキテクチャの勉強を始めるのにかっこうの参考書となります．英語で非常に分かりやすいのでぜひ読んでみてください．プログラマの立場から新しいアイディアを持ち込んだ非常に斬新なミニコンピュータで，現在の多くのプロセッサにその影響が見られます．また，プログラミング言語 **C**，オペレーティングシステム **UNIX** 等の発想に大きく作用したと言えます．

❑ 電源周波数クロック

◆ このミニコンには **Line Frequency Clock**（電源周波数クロック）というのがあり，それは電源の 1 サイクルを単位として時間間隔を知りました．これは商用電源を信号源とする簡単な波形変換回路（正弦波から方形波に変換する）と制御回路からなります．そして，割り込みを許可すると，1/60sec ごとに割り込みを発生します（西日本の話）．

図 -1 を見てください．動作開始時はリセットされるが，（OS が）a=65382 番地（番地は 10 進表現）



bit6	INT ENB（Interrupt Enable） 1にセットしておくと，CLOCK（bit7）が1になるたびに割り込みをかける． プログラムまたはリセットあるいは起動シーケンスでクリアされる．
bit7	CLOCK 電源に同期して毎1/60秒（あるいは1/50秒）ごとに1にセットされる．LKSを読む，リセットする，あるいは，STARTスイッチを押すと0にクリアされる．

図-1　電源周波数クロック状態レジスタ（右表にビットの説明）

の6ビット目（INT ENB 割り込み許可）ビットに1をセットすると以後7ビット目が1になるたびに割り込みがかかる．7ビット目は動作開始時OSがreset命令を発することで0にクリアされるが，電源から作った信号の変化したところで，ハードウェアが7ビット目を1とセットする．OSの時刻処理ルーチンが，a=65382番地を読む（read accessする）ことで7ビット目がクリアされる．そして，OSのシステムクロックを1/60秒分進めて割り込み処理から戻る．60回この操作を行えば，1秒経過したことになる．こうしてOSは実時間を保持したのです．

[課題案1]この電源周波数クロックの回路を設計する課題も面白い．電源の正弦波を入力として1/60秒ごとに，CLOCKビットを1にセットする働きをする回路である

● 時間も電力会社に依存していたのですね．いまならもっと精度の高いクォーツのクロックを使いますよね．

◆ そうです．そもそもコンピュータが実時間をどうやって知るのか，どの程度の精度が得られているかは本質的な問題だよね．もう1つ例を挙げます．

❑ ディスクインタフェース

◆ 電源周波数クロックより複雑な装置の例として**ディスク**があります．複雑な機器の場合は指示すべき事項が増えるので，レジスタの数が増えます．今から思えばごくごく小さい当時の磁気ディスクでも次のようなものは必要でした．サーフェイス番号，トラック番号，セクタ番号，メモリブロックスタート番地，メモリブロックサイズ等です．これらのデータを入れる複数のレジスタがあります．これらを設定しておいて，制御レジスタのビットr/wを1にし，start/doneビットを1にすると，ディスクからの読み出しが行われ，**DMA**（Direct Memory Access）方式[☆2]で，指定したセクタのデータがメモリに転送される．転送が終わったところで，割り込みをさせるなら，制御レジスタの割り込み許可ビットを1に設定しておく．制御／状況レジスタは1つのレジスタに集約することが多いのです．

❑ メモリ空間に置かれたIO

◆ これらの制御／状態レジスタ，データレジスタが，機器ごとに数ワードずつメモリの特定番地に設定されます．制御レジスタに書き込むことで動作の起動などを指示し制御し，結果やデータは状態レジスタ，データレジスタで読み取れます．この方式では，主記憶アクセスのバスと別に入出力バスを設けず，1つのバスで，プログラムで周辺機器を直接操作できました．**チャネル装置**を使う大型機に比べ，格段にコストが安く，かつ，標準品ではない機器でも，公開されているバスの規約に合わせ，未使用の番地にそれらの制御／状態レジスタ，データレジスタ類を配置し，プログラムを作成して，容易に接続することができたのです．このように，ある番地にアクセスして，1語のデータの読み書きをするメモリの1部に，特定の機能を持つレジスタ群

☆2 DMA：コンピュータに接続した種々の装置と主記憶間のデータ転送を中央処理装置CPUが取り仕切る方式をPIO（Programmed Input/Output）という．これに対し，CPUのお世話にならずに装置が自律的に動いてデータ転送を行うのがDMA方式である．PIOだとハードウェアは簡単というメリットはあるものの，CPUがIOの世話をするのでCPUの効率はいいとは言えない．DMAでは，装置ごとのDMA制御器に指示を与えるのはCPUだが，たとえば，ワンブロックのデータ転送を指示されれば，DMA制御器がすべての転送を終えて，割り込みをかけるという方式がとれるので，CPUの負担は減る．なお，別の見方では，分散制御方式とも言え，共通バスにアクセスする権利の調整のためバスアービタが必要になる．

を配置でき，メモリと同様にそれらのレジスタを読み書きできる仕組みで入出力機器を制御する方法を，**memory-mapped IO**（**図 -2**）と呼んでいました．主記憶と同じメモリ空間の一部，最後の 4k 語分をそれらのレジスタを置くために使いました．なお，割り込みベクトル等はメモリ空間の 0 番地に近い方の下位空間を使いました．

● はい．

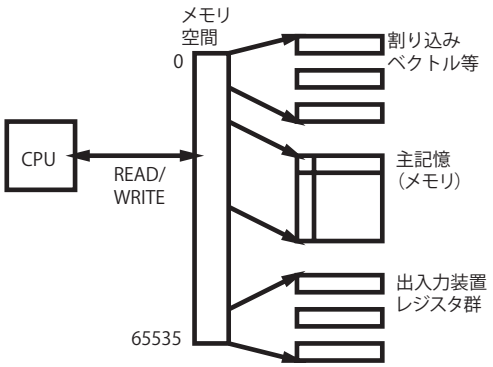


図 -2　メモリ空間に配置された IO

展開

❑ 2 進データの重み

◆ この仕組みで，当時いくつかのアイディアを持っていました．1 つ紹介します．番地 a に 2 進データ d を書き込むと，次の語（番地 a ＋ 2）のレジスタにそのデータの「重み」w(d) が得られるという演算機能を持たせたものです（**図 -3**）．

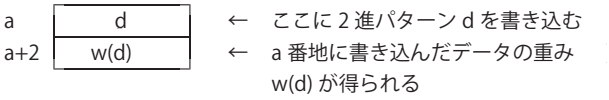


図 -3　重み算出器

● 重みってなんでしたっけ？

◆ **重み**（weight）は 2 進データ中の 1 の数です．2 進パターン d から，その重み（の 2 進表現）を求める組合せ回路は「論理設計」の演習でやらなかった？

●（きっぱり）一切記憶にありませんけど．

◆ ？... じゃ，復習にいいから最低 2 通りの組合せ回路を提案してください．なお，結構複雑になるので，小さい桁数から考えて桁数の増加にどう対応するかを考えるといいでしょう．

[課題案 2] このように論理設計の問題が作れる．これはちょっと難しい部類になります．回答は何通りもできますので，いろいろ自由に考えて利害得失を考えるといい

● はあ．それはそうとして... ほかにどういうことができるでしょうか．

◆ それは工夫次第．このように，メモリにデータを転送したら，計算結果がメモリのどこかに書かれてくるという機能を持ったメモリです．単にデータの入れ物というだけでなく，データの加工もするメモリという意味で，**機能メモリ** functional memory とか function in memory と呼んでもいいかもしれない．当時はサブルーチンコールのオーバヘッドも気になる時代ですからこの発想は魅力的でした．

● **サブルーチンコール**でどんなオーバヘッドが必要なのですか？

◆ サブルーチンコールでは，ただサブルーチンのところへ飛ぶだけでは済みません．パラメータを引き渡すとか，結果をもらうとか，サブルーチン内でレジスタを使うなら，レジスタのデータを退避しておくとか，いろいろ準備と後始末が必要です．これらがこの場合の**オーバヘッド**（Overhead）となります．まだそう高速でない時代のコンピュータです．少しでも早く動くことが求められました．いろいろ工夫を凝らして，こういう問題も 1 つ 1 つ改善されてきました．

● そうなんですか．

[課題案 3] このようなデータに対する演算機能を提供するものについて，そのインタフェースと，内部での処理の仕方を問う演習問題は多数作れる

◆ なにか 1 つのデータに対して，計算して結果を出してくれるような例を考えてみてください．たとえば，最初ちょっと設定して，あと次々データをほしいというような...

● ひらめきました．乱数生成器！　どうです？

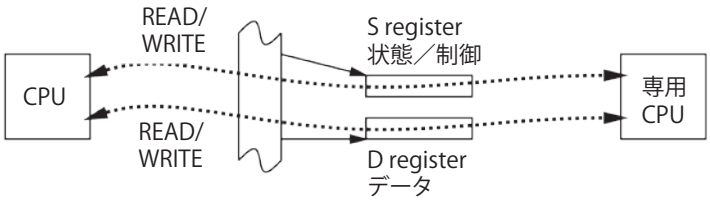


図 -4　窓口レジスタ

◆ なるほど．いい例を考えましたね．よく検討してください...（注：最大長系列，乱数，レジスタ転送表現，ハードウェアによる乱数発生などが話題になったが，紙面の都合で省略します）

結構です．じゃ，他の例はどうです？

● はい，三角関数 sin x の計算などどうでしょう．（注：三角関数は浮動小数点表示，たとえば 3 語の入力に対し，3 語の出力が得られるということで議論する．sin x を作るなら，当然ほかの FORTRAN の種々の数値計算の関数なども用意したくなるという話になる．それらは同じインタフェースレジスタの未使用の上位 8 ビットで区別するとか，データ型の変換なども扱えることを議論．省略）

❑ 実現法

◆ ここまでの話を少し整理しましょう．アプリケーションプログラムや OS が動く通常の世界から，このレジスタという窓を通して，計算してほしいと求められるわけです．通常の世界の方が当然主で，依頼されて計算する方が従です．主従の関係を master-slave という言葉を使うこともあるが，すこし言葉がよくない．なにかいい言葉はない？

● やはりクライアントとサーバという見立てができます．ただ，クライアント―サーバというとほかと混同する心配もありますけど，この文脈だと了解していただけば，別にネットワークの世界などとは混同されないでしょう．

◆ 整理すると，通常の計算が展開される方がクライアント側で，そこからリクエストを受け取り計算して結果を返すサーバがあるということですね．サーバをどう実現するか？

● 専用回路を用意するハードウェア方式も考えられますが，柔軟なシステムを作りたいとすれば，当然プロセッサを置き，ソフトウェアを使ってという考えになりますね．

◆ そう．そういうサーバ側のアーキテクチャをどうすればいいかを考えるのが，また，非常に面白い．高速な応答を求められる機能ならハードウェアで作ってもいいが，サーバ側にもプロセッサとメモリがあって，クライアントからの要求を引き受けると考えてもいいですね．むしろ，この考えの方が柔軟性があって，いろいろな発想が湧いてくると思います．両者の間に存在するレジスタ群を**窓口レジスタ**（**図 -4**）と呼ぶことにしましょう．そのうち，データを置くレジスタを **D レジスタ**，制御／状態レジスタを **S レジスタ**と略称します．

● そうみると，結構問題がたくさん作れますね．クライアントは従前の **CISC** で作り，このサーバ側は **RISC** で作るというようなこともあっていいわけですね．いままで習ったことをいろいろ思い出しながら，どうすればいいかを考えるといいですね．

◆ そう，課題・問題群を提出するのがこの対話の目的なので，そう思ってくれるといい．

データ構造・オブジェクトの扱い

◆ ここまで 1 変数関数を扱ったが，多変数関数ならどうだろう．

● 多分そうなるだろうと，ちょっと考えましたが，1 変数のときと変わり映えしないのです．つまり D レジスタを必要数設置し，そこにデータを用意して，S レジスタを通して「計算始め」といえば，結果が帰ってくるというだけで，D レジスタが増えるだけです．

push	D ← data（data を書き込むと自動的に push される）
top	D のデータを読むだけ（変化なし）
pop	S.pop ← 1（S の pop ビットが 1 の間は pop 作業継続中．終わると自動的に 0 に戻る）
isEmpty	S.empty で表示（1 ならスタック空，0 なら空でない）
isFull	S.full で表示（スタックが一杯のとき 1 と表示）
InitStack	S.Init ← 1（スタックの初期化をする．このビットが 1 の間は作業継続中で，終了すると 0 に戻る）

表 -1　スタックのインタフェース

empty	スタックが空のとき 1，そうでないとき 0
full	スタックが一杯のとき 1，そうでないとき 0
pop	1 とすると，1 語分 POP する
Init	1 とすると，初期化する
busy	1 のとき新しい指令は受け付けない．0 のとき S，D レジスタに書き込める
INT ENB	1 で割り込み許可．誤操作のときに割り込み

表 -2　スタックの状態語のビット

◆ そうだね．じゃ，今日は違う問題群を考えていきましょう．「データ構造とアルゴリズム」という講義を聞きましたね．

● はい．これはコンピュータサイエンスの最重要のコア科目ですから，ちゃんと勉強しましたよ．任しといてください．

◆ おっ！ 心強い言葉だね．いろいろなデータ構造が出てきましたね．

スタックから考えましょうか．スタックもここで考えている機能メモリに埋め込めます．

❑ スタック

◆ たとえば，1 語のデータを push したり，pop したりでき，top のデータはいつも直接見られるという**スタック**（stack）は簡単にできますね．

● はい．データレジスタ D と，状態レジスタ S とを用意します．そして，**表 -1** のようにすればいいと思います．

◆ 有限サイズのスタックという考え方でインタフェースを考えましたね．

● はい．抽象データ型のところでそういうことを聞きました．

◆ これはスタックの機能について整理したものですが，具体的な S レジスタのビット割り当ての案を作ってみてください．

● 図は省いて，ビット割り当てと役割を**表 -2** にまとめてみます．

◆ 状態レジスタのビットの位置は，システム内で一定の規約を設けて，同じ意味のビットは同じ位置にくるようにすると，プログラムミスを減らせるし，ハードウェアで制作する場合，回路を標準化できます．

● このシンプルなスタックでいいのかがちょっと心配です（以下，式の計算をスタックで実現する方法，アルゴル流言語のスコープルールを採用するような場合なども議論のテーマとなる．略）．

● 少し思ったのですが，こういう話をすると，なんでもサーバ側に任せないといけないと思う人がいますが，クライアント側でもちろん，これらのデータ構造は扱えるので，どっちがいいかのメリットデメリットを考えてやればいいことですね．

◆ そう，そういう考え方が大事です．いろいろな案を比較検討して，適切なのを選ぶのです．

[課題案 4]キュー，プライオリティキュー，表などの課題も面白い．まずインタフェースを考えるが，余裕があれば，内部での実現も検討してみるといい

◆ 抽象データ型の考えもこの方法となじむことも分かったと思います．ここまで来れば次はオブジェクト指向ですね．さあ，どうなるでしょう．

❑ オブジェクト

● オブジェクト指向プログラミングで扱うオブジェクトをサーバ側に預けるというのは適当かどうかはちょっと分からないのです．たとえば，基本データ型に相当する整数などのオブジェクトをわざわざこんなインタフェースを通して渡す必要はない気がします．ただ，システムの複雑な機能やセキュリティが問題になるようなクラスに属するものをサーバ側に任せて，クライアントからはメッセージを窓口レジスタを通してオブジェクトに送るという形が考えられますね．大体制御レジスタは空きビッ

トがあるので，そこで，あるオブジェクトに対するメソッドの区別をすれば，1つのオブジェクトに多数のレジスタを用意する必要はなくなりますね．

◆ そうだね．そういう切り分けが有用となる可能性があります．それから，new(p) でインスタンスを次々作るようなときにどうしますか．

● これはサーバ側にメソッドの処理を任せるが，データはクライアント側に置いて，そのアドレスを窓口レジスタに書いて，サーバからそのアドレスでアクセスしてもらうという方法で，クライアント側にデータを持つことはできます．

◆ そう，そういう考え方もできるし，データもサーバ側に押しつけたければどうする？

● うーん．そうですね．窓口は1つだけど，生成されるインスタンスを区別するために**タグ**（一連番号）を用意して，それを使ってサーバ内にあるオブジェクトの実体データを区別すればよいと思います．

◆ そういう考え方もできるね．こうすれば，データはサーバ側に隔離することもできる．結局，オブジェクトだからといって特にこれまでと変わるところはない．その実現法はいろいろ考えられるということで，課題がいくつも作れるでしょう．特に，クラスライブラリで提供されているようなオブジェクトをこういう仕組みで利用するのも面白いと思われます．その際，継承などのクラスにかかわる概念をどうするかもありますし，そのためにどういう仕掛けが必要で効果があるかの検討も必要ですね．

● なるほど．まだまだ深く考える問題はあるということですね．

OS とセキュリティ

◆ 実は今までの話を振り返ると，サーバ側はクライアントのいろいろな要求を聞いて，データ処理をしたり，入出力を代行したりします．ということは，広義での OS になっているとも見えるわけです．そこで，逆に，「サーバ側にいるものを OS である」と視点を変えてみると，どういう景色が見えるかということです．

● 私も薄々そういう予感がありました．プログラムから OS に要求を発する**スーパーバイザコール**（SVC，supervisor call）も，この窓口レジスタのやり方で自然に実現できます．スーパーバイザコールは，プログラム割り出し，OS へのリクエスト等種々の用語があるが，実行中のプログラムから OS にいくつかのパラメータを伴って，依頼の種類を番号で伝えるものです．したがってパラメータを D レジスタに設定し，依頼の種類を S レジスタに設定し，サーバ側に伝えればいままでと同じ形で実現できます．クライアント側のメモリ空間にサーバ側が DMA のようにアクセスすることでクライアントのメモリの読み書きをする．こうして，通常の OS の機能はここでいうサーバ側にすべて持っていけるのですね．クライアント側はユーザタスクの実行のみをやり，サーバ側に OS がいて，サービスするという役割分担ができます．

◆ そうですね．さて，OS だと思えばどうなるか．宿題を出しておいたでしょう．

● 次のようなことを考えました．いわゆる**マルチタスク**が簡単にできるようになります．各タスクはタスク番号で管理します．たとえば，仮に0から31とすれば，32タスクを扱えます．

タスクごとに使う高速レジスタ群を，**タスクレジスタ**と仮に呼びます．これには，そのタスク用のプログラムカウンタ PC，プログラム状態語 PSW（むしろ，タスク状態語と呼ぶ方がいい），汎用レジスタなどが含まれます．タスク番号が32個あれば，タスクレジスタを32個用意し，タスク番号で切り替えて使います．普通のメモリはキャッシュを使うとして，仮想ブロックと実ブロックの対応は OS 側が管理しています．対応表はタスク番号をアドレスの上位ビットにつけて，変換表（連想記憶）を検索し，実ブロックアドレスを見つけてアクセスするわけです．OS は，クライアント側の実行タスク番号レジスタを制御して，そこに実行すべきタスクの番号を設定し，CPU に実行を命じれば，自動的にそのタスクに必要な高速レジスタ群が選択され，通常のメモリアクセスもタスク番号が論理アドレスの上位に

使われます．このような方式で，タスクスイッチは，単にタスク番号の書き換えだけで済み，初期の OS で問題になったタスクスイッチのオーバヘッドは非常に小さくなります．

◆ そう，**タスクスイッチ**のオーバヘッドを小さくできる．もちろん，スケジューリングは OS の方でやって，次に起動するタスクを決めておくという作業は必要です．

● でも，2 つのプロセッサとメモリを使うというのは少しもったいないという気がしますが．

◆ 最近は quad core のプロセッサもあります．集積度が向上し，むしろシリコンの面積は余っていて，そこに何を埋めるか，いろいろ工夫しているわけです．

● そういえば，ワンチップマイクロプロセッサは多くの機能をてんこ盛りにしてますね．

◆ どういうアーキテクチャが合理的かは，そのときの技術で決まります．pdp11 を知って，この機能メモリを考えたけれど，とても当時の技術では合理的ではなかった．だから，スタックくらいしかそのときは検討しませんでした．

● 今なら使えるかもしれないということですか．

◆ そうですね．今やシリコンチップには相当の機能が埋め込めます．並列処理の機会が多ければ同一のプロセッサを多く持つアーキテクチャが有利だけれど，この機能メモリの考えは，むしろ機能分割的な発想です．CPU は同じでなくてもいい．

こうなると，サーバ側は OS 専用の CPU にしてもよい．従来は 1 つの CPU で OS もユーザプログラムも実行するので，OS 専用という発想はなかったでしょう．

● はあ．これで OS 専用のハードウェア・ソフトウェア協調設計の契機になりますね．

◆ そう，OS に特化したアーキテクチャを検討することは面白いと思いますよ．それとメモリ空間とプロセッサを分割することで，セキュリティを高める方向のアーキテクチャが考えられます．このように，どんどん自由に考えていいのですよ．

● じゃ... **TCP/IP** 方式の通信処理は必須のものとなっていますけど，この処理の部分をさらにサーバの奥のサーバに分離し，パケット処理を行うのはどうでしょう．

◆ 奥か，傍かはまあいいとして，処理を分離して安全性を高めようということですね．それも面白いし，重要ですよ．

● この調子だと，特別の入出力処理部を切り離すとか，データベース処理を別に切り出すとか，いろいろ窓口レジスタというインタフェースを介して連絡を取りつつ自律的に動く複合システムの形に全体を再構成するという案もありですね．

◆ そう，そのとおり！ わくわくしてきたでしょう．

● はい．アーキテクチャを自由に考え直せばいいのですね．（対話はここまで）

対話型教材とその利用

通常，講義案を大体立てて，それに応じたテキストや演示教材を作成し，自分の講義の準備をするであろう．そのテキストを公開して他の教員の参考にするという方法があるが，目的や対象の異なる場合，テキストを読み取った上で，また一からテキストを構成すると膨大な手間がかかる．その際，著作権を侵害しないことや自分独自の説明をしたいと考えると大きな負担がかかる．ここで教材案の提示法として，対話型を選んだのは，これを自分の講義の目的にあった教材作成の一種の設計書として利用できるのではないかという考えからである．ここで試みたのは，対話形式にすることで，不要な細部の記述をさけ，大きな流れを示したり，一種の抽象化を図っており，他の講義にも適用する幅を広げて，より広く使っていける方法としたいという意図がある．実際の対話型講義を一段抽象化したもの，あるいは，粗いシナリオと言ってもいい．謎を相手に投げかけ，やりとりを進めていくことでストーリーを記述している．ここで単に一方的に知識を伝える講義でなく，考えさせる講義をという意図も見えてくるであろう．

この例ではあまり感じられないだろうが，シナリオ，あるいは，対話型の場合，その教育の目的や歴

史的説明，その他のメタ情報を◆の言葉として埋め込むことができる．教科書は，著者の意図は前書き等に書かれることはあるが，できてしまった教科書の記述から，その背後の意図を推察することは難しいことが多い．しかし，ここで採用した対話型はそういう記述を埋め込む自由度があることも指摘しておきたい．つまり，教材の見せ方として対話型の可能性をお見せしたかったのである．

情報処理教育をさらに良くするために，もっと教材研究を進め，それを公開し検討する機会を学会としても強化するとよいと考えている．この本稿はそのために，対話型という提示方法の可能性をアーキテクチャ教育の事例で示したものである．なお，この事例はオープンエンドにしてあるが，これに刺激されてその先を考える人が現におられることは筆者のよろこびとするところである．そして，この例に限らず，多くの科目で新しいストーリーで，興味深い，学生を巻き込む講義の可能性が残っていると信じて，教材開発をしていただきたいと考えている．

また，学生に考えさせる講義の具体的な提案を，本号教育コーナー「ぺた語義」に収録していただいているので参照していただきたい．図 -2，図 -4 は久野靖先生によるものであり，ここに感謝申し上げる．

参考文献

1) 清水健一 : 大学入試問題で語る数論の世界，素数，完全数からゼータ関数まで，ブルーバックス，講談社 (2011)，ISBN978-4-06-257743-4.
2) pdp11 handbook, Digital Equipment Corporation (1969), http://research.microsoft.com/en-us/um/people/gbell/Digital/pdp%2011%20Handbook%201969.pdf

（2011 年 7 月 31 日受付）

都倉信樹（正会員） ntokura@nike.eonet.ne.jp
CS'90，CS'97，IS'97，J07-IS 等の策定に参画した．大阪大学，鳥取環境大学名誉教授，大阪電気通信大学学長．本会フェロー．