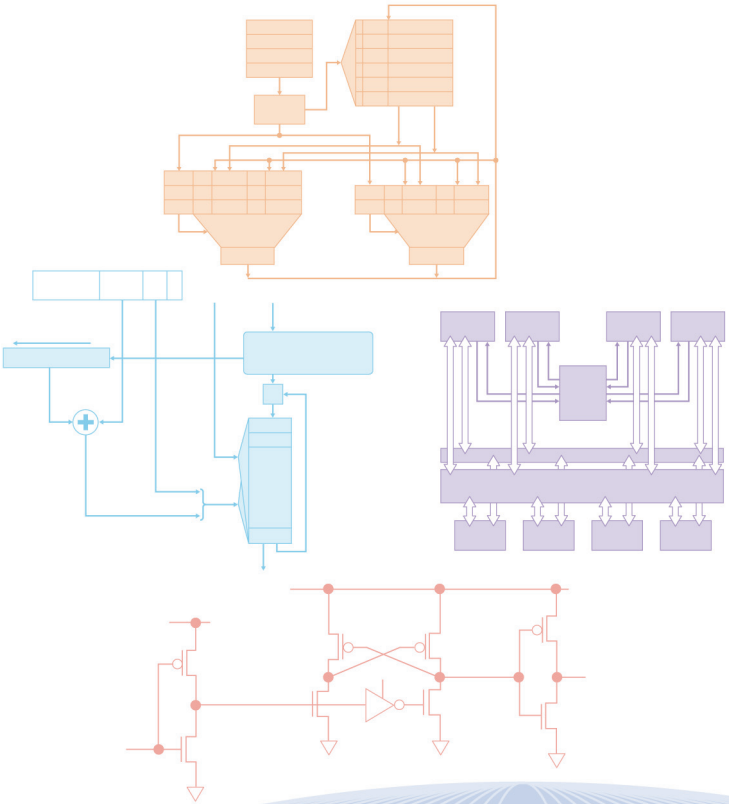
高性能コンピュータ技術の基礎

Hisa Ando ● [著]



コンピュータの歴史は「高性能化の歴史」

スーパスカラ／アウトオブオーダ実行／投機実行／
マルチプロセサキャッシュ／コヒーレンシ制御／ マルチコア化／
マルチスレッド化／ 仮想化／省電力化を解説。

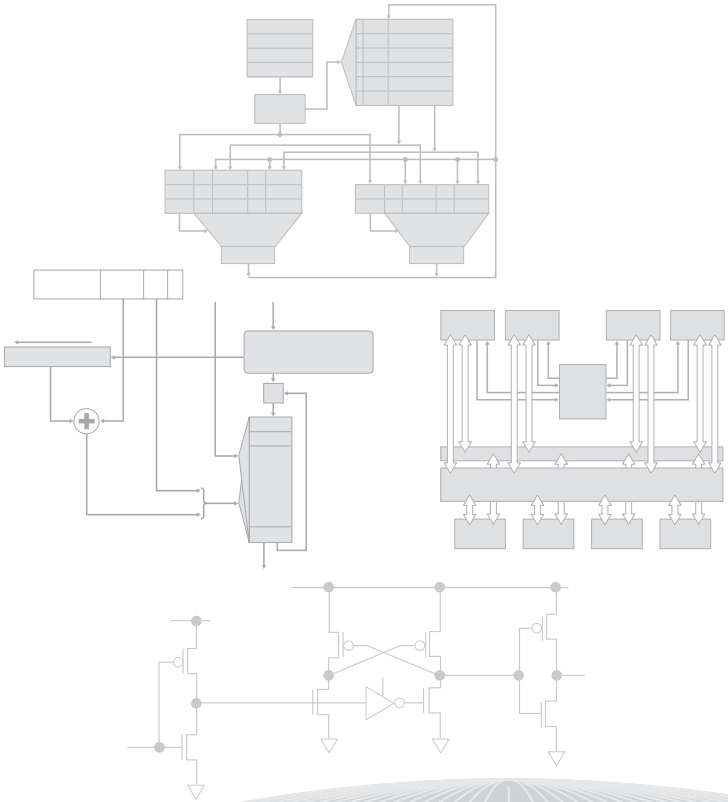
マイナビ

**コンピュータの高性能化の技法と
低消費電力な設計について解説。
プロセサだけでなく,
一般的な高性能ディジタルLSIを
設計する際にも役に立ちます。**

CONTENTS

高性能コンピュータ技術の基礎

Hisa Ando ● [著]



マイナビ

本書のサポートサイト

本書に関する追加情報等について提供します。

http://book.mynavi.jp/support/pc/architect1/

はじめに

1942年に完成したABCから数えてコンピュータの歴史は70年ほどですが，1946年に完成した超大型計算機ENIACの加算性能が毎秒5000回であったのと比較すると，最近のハイエンドマイクロプロセサの加算性能は2億倍になっています。このように，コンピュータの歴史は**高性能化の歴史**といっても過言ではありません。

拙著『コンピュータ設計の基礎[*1]』では，パイプラインやキャッシュなどの基本的なコンピュータの構造と演算器を中心に説明を行いましたが，本書では，さらなる高性能化を実現する複数の命令を並列に実行する**スーパスカラ**，データ待ちなどの命令を後回しにして実行する**アウトオブオーダ実行**，条件分岐の方向を予測して実行する**投機実行**などの高性能化技術について2章，3章で詳しく説明します。また，複数のプロセサを使ってさらに性能を上げる**マルチプロセサ**とそれに伴って必要となる**キャッシュコヒーレンシ制御**などについて4章で説明しています。これらの技術は複雑ですが，簡単な例をベースに，これらの機構がどのように動くかを説明します。多少，厳密さには欠けるところはありますが，理解しやすい記述になるようにしました。

アウトオブオーダや分岐予測などの機構の実現には，多くの状態の記憶や一致検出などの回路が必要であり，ムーアの法則によるトランジスタ数の増加でこのような機構の実装が可能になってきたことを理解していただけると思います。

これらの高性能化機構は1990年代からマイクロプロセサへの実装が開始され現在でもその改良が続いていますが，2000年代に入り，新たな状況が出てきました。マイクロプロセサの消費電力が100Wを超え，さらにトランジスタを注ぎ込んで高性能化を目指す方向の発展が難しくなってきました。また，巨大なデータセンタなどが建設され，コンピュータをより効率よく使ってコストや電力を削減したいという要求が強くなりました。このような背景から，**マルチコア化**，**マルチスレッド化**や**仮想化**に対するニーズが強くなり，コンピュータのマイクロアーキテクチャに影響を与えてきます。5章では，これらの最近の技術を取り上げます。

データセンタでは，運用コストのなかでもっとも高い比率を占めているのが電気代で，**消費電力の低減**が強く求められています。また，近年，広く使われるようになってきた携帯機器は電池で動作し，電池寿命の点でプロセサの消費電力低減が強く求められています。これらのニーズから，消費電力を低減することは最近のマイクロプロセサの設計では一番重要な設計目標となってきています。6章では，これらの**省電力化**の技術について説明します。コンピュータのアーキテクチャや論理設計関係の読者には馴染みの薄い半導体技術や回路技術の話が出てきますので，わかりやすい説明を心がけました。

本書を読む上で，『コンピュータ設計の基礎』を読んでいることは必須ではありませんが，本書は，パイプライン制御やキャッシュなどの技術は理解しているという前提で書かれています。

2011年5月 Hisa Ando

＊1 『コンピュータ設計の基礎』（マイナビ 刊）ISBN978-4-8399-3753-9

Contents：目次

Chapter.3
予測に基づく投機実行

Chapter.4
複数のプロセサで処理を高速化するマルチプロセサ

Chapter.5
最近のマイクロアーキテクチャの発展

Chapter.6
省電力設計

Chapter.1

コンピュータの高性能化

コンピュータの歴史は，高性能化の歴史である。1946年に完成した電子式大型コンピュータであるENIACは10進10ケタの加算を毎秒5000回実行するという，当時としては驚異的なスピードであったが，今日のパソコンで用いられているプロセサチップは同程度の精度の加算を毎秒1兆回程度実行できる。単純に加算性能だけをみても，この65年で2億倍の性能向上となっている。

1.1 コンピュータの性能向上トレンド

プロセサ性能の測定法としてStandard Performance Evaluation Corporation（SPEC）が開発，運営するSPECmark[5]というベンチマークプログラムが広く用いられている。このSPECmarkで測定したプロセサの相対性能の年次推移を図1.1に示す。

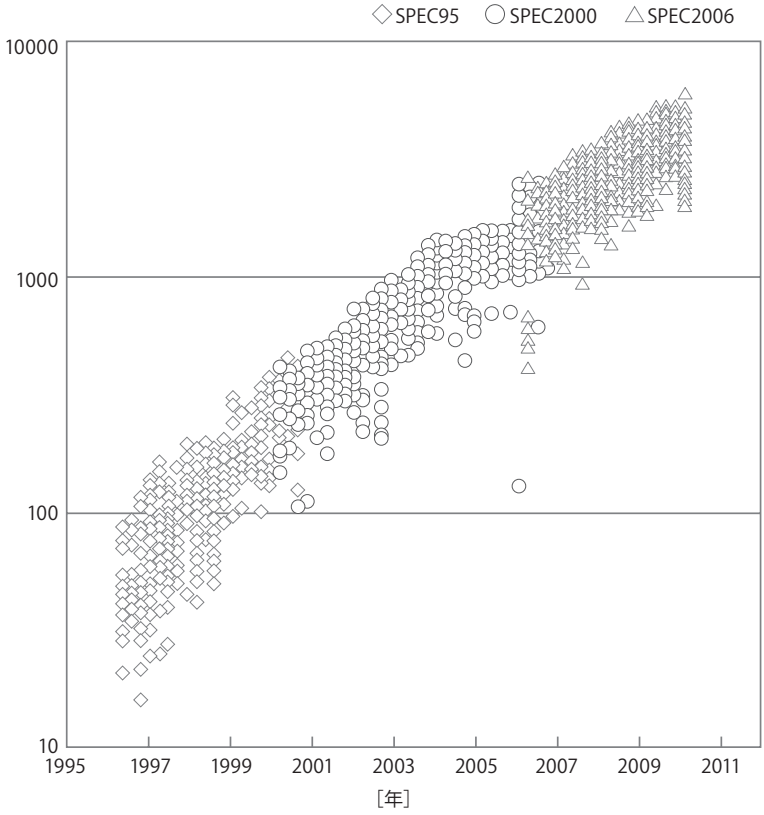


図1.1 SPECintBase性能の年次推移

図1.1を見ると，1996年の最高性能のコンピュータの性能は80程度であるが，2011年の最高性能のコンピュータは8000に近い値となっており，この15年で100倍の性能向上が達成されている。このペースは年率36%の性能向上であり，65年で2億倍とほぼ同じペースの性能向上が続いていることになる。

この性能測定に用いられているSPECの整数ベンチマーク（SPECintBase）は，コンパイラのgccやファイル圧縮のbzip，ビデオ圧縮のh264refなど10種あまりの実用プログラムが含まれており，これらの各プログラムの基準マシンに対する実行性能比の幾何平均で総合的なプロセサ性能を求めている。SPECベンチマークはコンピュータの進歩に伴って改訂されてきており，**図1.1**にはSPEC95，SPEC2000と現役のSPEC2006の結果が含まれている。これらの3つのベンチマークでは，含まれるプログラムが多少変わったり，同じプログラムでも問題規模が大きくなったりという違いがあり性能評価の基準が同一ではないが，これらのベンチマーク間で性能が連続するようスケールして図を描いている。

このようなプロセサの性能向上を実現してきた原動力は，半導体の微細化に伴うクロック周波数の向上と，クロックサイクルあたりに実行できる命令数を増加させるマイクロアーキテクチャの進歩である。また，コンパイラの進歩による性能向上も忘れてはならない。

しかし，クロック周波数の向上には，パイプライン1段あたりの遅延時間を短縮するマイクロアーキテクチャの改善による寄与があり，コンパイラは，新たなマイクロアーキテクチャや増大するキャッシュ容量を利用して効率の良い命令列を生成するというように，これらの要因の性能向上に対する寄与はオーバラップしており，改善効果を要因別に分解することは難しい。

1.2 高性能プロセサの設計

前著『コンピュータ設計の基礎』[1]で説明したパイプライン処理を行えば，理想的には毎サイクル新しい命令の処理を開始することができるはずであるが，現実的には，

(1)構造ハザード: 実行ユニットなどの資源要求が競合する
(2)データハザード: 前の命令の結果待ちで命令の実行が開始できない
(3)コントロールハザード: 条件分岐命令の分岐方向が決まるまで，次の命令がフェッチできない

というようなハザードがあり，1命令 / サイクル（Instruction Per Cycle：IPC）は達成できない。これに対して，ムーアの法則で豊富に使えるようになってくるトランジスタを利用して，実行資源を追加して構造ハザードを減らすだけでなく，命令の実行順序を入れ替えてデータハザードを減らすアウトオブオーダ実行や，分岐方向を予測してコントロールハザードを減らすなどの，IPCを向上するマイクロアーキテクチャの改善がなされてきた。また，1サイクルに1命令ずつ処理するのではなく，1サイクルに複数の命令を実行するスーパスカラ構造や，複数のプロセサを使うマルチプロセサなどによる高性能化が行われている。

スーパスカラで複数の命令を並列に処理するためには，データハザードを減らす必要があり，そのために命令の順序を入れ替えてデータハザードのない命令を先に実行するアウトオブオーダ処理が行われるが，どのようにして命令の実行順序を入れ替えるのか，実行順序を入れ替えても処理結果が変わらないようにするにはどうするのかなどについてを2章で説明する。

また，スーパスカラで1サイクルに複数の命令を処理するようになると，コントロールハザードによる性能低下の影響が大きくなる。このコントロールハザードによる性能低下を減らすのが分岐予測である。条件分岐の方向や跳び先をどのように予測するのか，高い予測成功率を実現するにはどのような構造が使われるかなどを3章で説明する。

たくさんの仕事がある場合は，1台のプロセサでなく，複数台のプロセサを使え

ば速く処理ができる。しかし，多くのプロセサが協調して効率よく動けるようにするためにはいろいろな仕掛けが必要である。特に，キャッシュを持つプロセサを複数使うマルチプロセサシステムでは，それぞれのプロセサの持つキャッシュの内容を矛盾の無い状態に保つ必要がある。このキャッシュコヒーレンシ処理を中心として，マルチプロセサシステムの課題とそれを解決する技術について4章で説明する。

5章では，複数のプロセサコアを集積するマルチコアプロセサの出現の背景から，マルチコア化による性能向上について述べる。また，複数の命令列を並行して実行し，1つのプロセサを複数に分割して使用するマルチスレッドや時分割で1つのプロセサを多数の仮想プロセサに分割して使用する仮想化の技術について説明する。仮想化は運用に柔軟性を与え，データセンターに必要なサーバ台数や消費電力を削減する重要な技術になっている。

6章では，省電力設計技術を取り上げる。携帯電話などのモバイルデバイスに使用されるプロセサでは，電池で駆動されることから，低電力であることが非常に重要である。また，多数のサーバを使う巨大データセンターでは，運用コストの中で最大の比率を占めるのが「電気代」という状況になってきており，省電力が強く求められるようになってきている。このような状況から，省電力設計は現在のプロセサ設計の最も重要な設計技術となってきている。

Chapter.2

複数命令の並列実行

1命令ずつのパイプライン実行より性能を上げようとすると，考えられるのは複数の命令を並列に実行するというやり方である。歴史的な経緯も振り返りながら，どのような技術によって並列実行ができるようになってきたのかを見ていこう。

2.1 スカラ方式とベクトル方式

通常のプロセサの命令では，浮動小数点加算命令は，

```
FADD   F4←F1,F5;        F1の内容とF5の内容を加算し，F4に格納
```

のように，F1レジスタとF5レジスタに格納された1つのデータを加算して結果をF4レジスタに格納するという処理を行うが，CRAY-1スーパーコンピュータ[6]では，それぞれのレジスタが64個の浮動小数点データを格納し，1つのFADD命令で最大64個のデータを加算するという命令が作られた。そして，このように複数のデータ列（＝ベクトル）をひとまとめに処理する形式の命令をベクトル命令と呼び，このような実行方式をベクトル実行と呼ぶ。

なお，当時のテクノロジでは大量のトランジスタを必要とする演算回路は高価であり，64個のデータの加算は，パイプライン構成の1つの演算器で順に処理していた。それでも，1命令ずつ実行するプロセサでは1つの浮動小数点演算の実行に少なくとも数サイクルを必要とするのに対して，パイプラインによる実行の場合は毎サイクル新しい演算を開始することができるので高性能を実現できるというメリットがあった。また，これらの64個の加算は相互にデータハザードはなく，演算器の数を増やしてやれば，1サイクルに複数の演算を実行するように拡張することは容易である。このため，1990年代のスパコンではベクトル実行方式を用いるものが多く開発された。

また，現代のマイクロプロセサでは，x86プロセサのSSEやAVXといった命令は128ビット，あるいは256ビットのデータを扱っており，要素データが64ビット長の場合は，毎サイクル2要素，あるいは4要素の演算を並列に実行することができるようになっている。

このベクトル実行に対して，各レジスタは1つのデータを格納し，1つの命令で1つのデータを処理する普通の処理方式はスカラ実行と呼ばれる。そして，2つ以上のスカラ命令を並行して実行する方式は1サイクルに複数の演算が行われるが，ベクトルデータの処理ではなく，それぞれは異なるスカラデータの演算であるので，

スーパスカラ（Super Scalar）実行と呼ばれる。

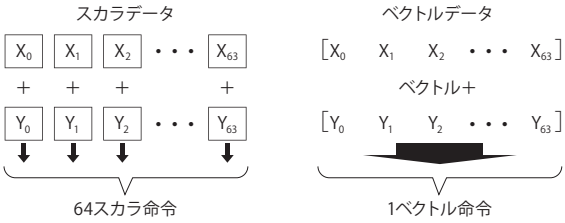


図2.1　スカラ実行とベクトル実行の概念図

ベクトル実行を行うことにより，1サイクルに複数の演算を実行することができ，大量の計算を行う科学技術計算には有効であるが，一般の処理ではベクトルでないデータに対しても並列に命令を実行して性能を上げたいという要求がある。このため，現在のマイクロプロセサでは，複数のスカラ命令を並列に実行するスーパスカラ実行が用いられるようになっている。

2.2 複数の命令を並列に処理するスーパスカラ方式

一般に，スカラプロセサは，独立した整数演算器（ALU：Arithmetic Logic Unit）と浮動小数点演算器（FPU：Floating Point Unit）を持っており，次の例のような命令列の場合には，浮動小数点加算（FADD）と整数加算（ADD）を並列に実行すれば，両方の演算器を遊ばせることなく有効利用ができる。なお，ここでFの付いたレジスタは浮動小数点データ用レジスタであり，Rの付いたレジスタは整数データ用レジスタを示す。

```
FADD   F4←F1,F5;        F1の内容とF5の内容を加算しF4に格納
ADD    R6←R3,R2;        R3の内容とR2の内容を加算しR6に格納
```

また，整数演算器はプロセサ全体の面積から見ると数％かそれ以下であり，面積が小さいので，これを2個搭載して2つの整数加算命令を並行して実行できるような構成としても，面積増加によるコストアップは僅かである。このようなプロセサでは，整数演算命令と浮動小数点演算命令のペアだけでなく2つの整数演算命令のペアも相互にデータ依存が無ければ同一サイクルに実行することができ，2命令を並列に実行できるケースが大幅に増加して性能改善効果が大きい。このような構造のスーパスカラプロセサのブロックダイヤグラムを**図2.2**に示す。

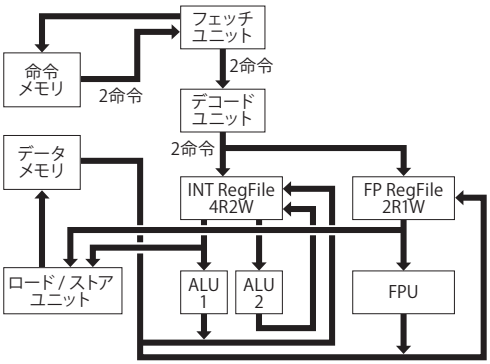


図2.2　2つのALUと1つのFPUを持つ2命令並列処理スーパスカラプロセサの例

図2.2に示すように，スーパスカラ実行を行うためには複数の命令を同時にフェッチし，命令デコードユニットも複数の命令を並列にデコードする。そして，デコードユニットは，それぞれの命令についてデータハザードや構造ハザードが無いことを確認するが，それに加えて，同一サイクルに実行を開始する先行する命令と後続の命令の間でデータの依存性や使用資源の競合が無いことを確認する必要がある。そして，このようなハザードが存在しないことが確認できると，整数（INT）レジスタファイル，あるいは浮動小数点（FP）レジスタファイルからオペランドを読み出し，それらの命令を演算の種別に応じてALU1，ALU2，FPU，ロードストアユニットといったそれぞれの処理の実行パイプラインへと送り込むということになる。

しかし，

```
ADD   R6←R3,R2;      R3の内容とR2の内容を加算しR6に格納
ADD   R7←R6,R4;      R6の内容とR4の内容を加算しR7に格納
```

のように，並列にデコードする命令間にデータ依存（この場合は，先行するADD命令の結果であるR6を後続のADD命令が使用する）がある場合は，最初のADD命令が終了してR6が求まってから次のADD命令を実行することになり，スーパスカラ実行はできない。

また，データ依存が無くても，浮動小数点演算器が1個しかないと，次の例のように浮動小数点加算命令が連続している場合は，両方の命令を並行して実行することはできない。

```
FADD  F4←F1,F5;      F1の内容とF5の内容を加算しF4に格納
FADD  F6←F3,F2;      F3の内容とF2の内容を加算しF6に格納
```

もちろん，浮動小数点演算器を追加して2個にすれば並列実行が可能であるが，浮動小数点演算器は整数演算器よりも必要なチップ面積が大きいので，筆者の知る限りでは，2命令並列デコードの汎用プロセサで2個の浮動小数点演算器を持つ例は無い。一方，4命令並列デコードのプロセサでは，2個の整数演算器と2個の浮動小数点演算器を持つプロセサが一般的である。ただし，全く同じ演算器を2個持つとは限らず，IntelのCore 2プロセサなどでは，一方の浮動小数点演算器は加減算に加えて乗除算や三角関数などの複雑な浮動小数点演算をサポートするが，もう

一方の演算器はベーシックな加減算だけというようなそれぞれの演算の使用頻度を考えた非対称な実装が行われている。このように構成の異なる2個の演算器を持つ設計は，デコードや構造ハザードの検出という点では多少複雑になるが，演算器の必要面積と得られる性能向上のバランスの点で有利である。

このように，1サイクルに2命令を並列にデコードし実行可能なハードウェア構成としても，データハザードや構造ハザードにより1命令しか実行できない場合があり，常に1サイクルに2命令を実行できるわけではない。従って，性能向上効果は，プログラムの中でどのような命令が連続して並んでいるかに依存するのであるが，一般的に言って，2つの整数演算ユニットを持ち，2命令を並列にデコードするスーパスカラ方式のプロセサでは，単純な1命令ずつのパイプライン実行の場合と比べて1.2〜1.4倍くらいの性能向上が得られる。

2.2.1　複数命令の並列デコード

このように複数の命令を並行してデコードする場合，命令長が固定のRISCアーキテクチャの場合は，2番目以降の命令の開始位置も決まっている。従って，4バイト固定長の命令を2命令ずつ処理する場合は，**図2.3**に示すようにその次のサイクルでは8バイト先の命令からデコードすればよい。

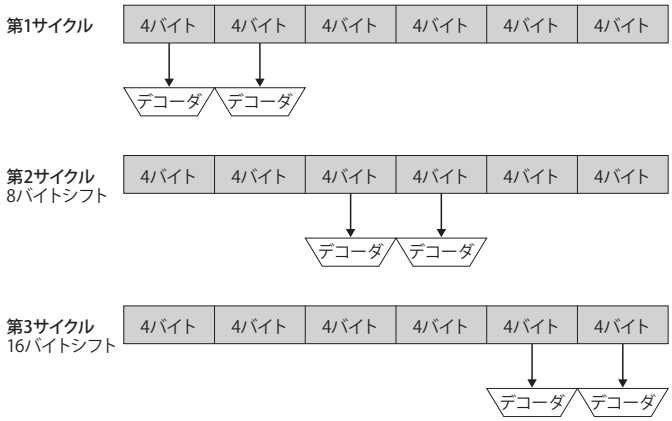


図2.3　固定長のRISC命令の2命令並列デコード

一方，命令長が可変のCISC命令の場合は，最初の命令をデコードしてその命令

の長さを知らないと次の命令の開始アドレスが分からず，次の命令のデコードができない。Intelのx86アーキテクチャ[7]では最短の命令は8ビット（1バイト）であるが，メモリアドレスの指定がレジスタの内容＋オフセットとなったり，更に，各種のプレフィックス（Prefix）やアドレスのディスプレースメント（Displacement）などが付いたりすると最大15バイトまで命令の長さが変わってしまうので，次の命令の開始アドレスを見つけるのは容易ではない。

図2.4に示すように，CISCの場合は，2番目の命令のデコードは最初の命令をデコードして長さを知り，次の命令の先頭バイトが正しい位置にくるようにシフトを行ってデコーダに入力する必要がある。そして3番目の命令は前の2命令をデコードしてそれらの命令の長さが分からないと開始番地が分からずデコードができない。

このようにCISCアーキテクチャのプロセサでは，前の命令から順に命令の長さを知ってシフトを行って命令の先頭バイトの位置を合わせる必要があり，1サイクルに並列にデコードする命令の数が増えると，命令数に比例してデコード時間が長くなり，クロックサイクルタイムが延びてしまうという問題がある。

この問題をある程度回避するため，スーパスカラ方式を採用したPentium Pro以降のIntelプロセサは，出現頻度が高く，かつ，デコードが容易な命令は複数命令を並列にデコードするが，複雑に長さが変化する命令に出会うと，並列デコード命令数の制限内であってもそこでデコードを打ち切り，複雑な命令は次のサイクルに廻すというような処理を行っている。

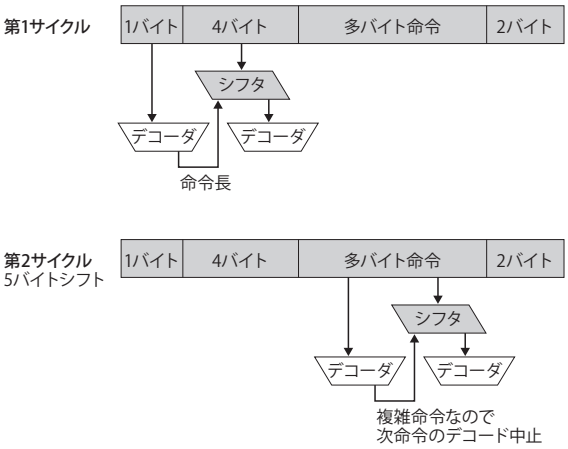


図2.4　可変長のCISC命令の2命令並列デコード

また，RISC，CISCに拘わらず，同一サイクルに発行する命令間のデータ依存性のチェックや使用資源の競合のチェックなどは，本質的にシリアルに処理する必要があるので，複数命令の並列デコードはクロックサイクルタイムに対するプレッシャーになる。

並列度を2命令から3，4命令と増やしていくとそれなりに性能は向上するのであるが，データ依存性により並列に実行できないケースも増えるので，ハードウェアが複雑になる割には性能の向上度合いが飽和する傾向にある。このような理由から，次に述べるアウトオブオーダ実行を行わない単純なスーパスカラプロセサでは，2命令並列というものが一般的である。

2.3 アウトオブオーダ実行

これまでに述べてきたプロセサは，プログラム（CやC++などの高級言語ではなく，それらをコンパイルした機械命令レベルの）に書かれた通りの順序で命令を実行するプロセサである。

しかし，

```
FDIV   F1←F2,F3;      F2の内容をF3の内容で割り，結果をF1に格納
FADD   F4←F1,F5;      F1の内容とF5の内容を加算しF4に格納
FSUB   F6←F7,F8;      F7の内容からF8の内容を減算しF6に格納
```

のような命令列があった場合，浮動小数点加算命令FADDは，実行に時間の掛かる浮動小数点除算命令FDIVの結果であるF1に依存しているので，FDIV命令が実行を終わって除算結果が出るまで実行を開始できないが，浮動小数点減算命令FSUBは先行するFDIV命令やFADD命令の結果には依存しないので，FDIV命令の終了以前に実行することも可能である。そして，**図2.5**（次ページ）の下側に示すようにFSUBや，FDIVやFADDの結果に依存しない後続のFADD/FSUB命令を先に実行してしまえば，命令の実行時間を短縮することが可能なはずである。なお，ここでは浮動小数点除算器と浮動小数点加算器は並列に動作できると想定している。

実社会でも，課長が不在でOKをもらえないとか，部品が足りないとかで着手できない仕事を後回しにして，できる仕事から先にやるのは当たり前のことである。

ということで，プログラムの中での命令の順序を変えて，できる命令から先にやるというマイクロアーキテクチャが考えられた。この方式は，命令として記述された順序（Order）を崩して実行するのでアウトオブオーダ（Out of Order）実行と呼ばれる。

試し読みはお楽しみ
いただけましたか？

ここからはManatee
おすすめの商品を
ご紹介します。

Tech Book Zone Manatee

3.15
2017

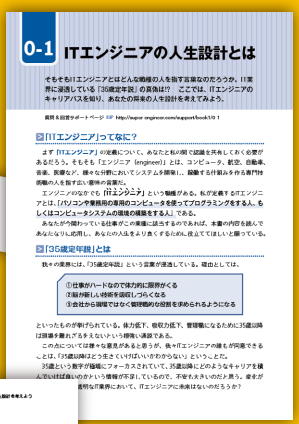
おすすめ電子書籍

Tech Book Zone Manatee

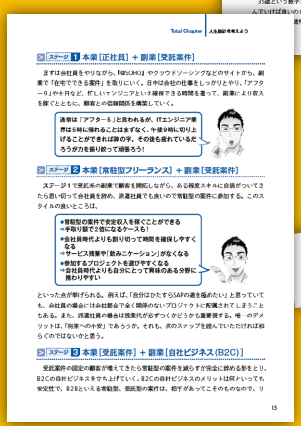
1

副業? 独立? それとも?
ITエンジニアの人生設計の決定版!

組織を束ねるマネジャーになるか、現場のスペシャリストであり続けるか。自分の技術を活かして独立するか、副業を考えるか……。意外に悩ましいITエンジニアの人生設計。会社に依存しない、転職や独立も射程に入れた「マインドセット」の持ち方から、お金、営業戦略、顧客対応術、ビジネスモデルの構築といった「ビジネスロジック」まで。求人情報ポータルサイト「@SOHO」の開発者が、自身の知見と経験から得たノウハウを教えます。



トピックスごとにオンラインの公式サイトで質問コーナーを用意。疑問に答えてくれる



ITエンジニアがどのように人生を描き切り開いていけるか、著者の実体験をもとに解説

「開発技法」

推奨職種を併記。アーティストやレベルデザイナなど専門ごとに読むべき章がわかる

プログラムやスクリプトの勉強をしたことがない人でも読み進められる

2

UE4におけるゲーム制作で
必須の知識と経験が身につく!

『Unreal Engine 4で極めるゲーム開発』は、Unreal Engine 4（UE4）の機能を単に紹介するだけでなく、一本のサンプルゲーム開発に順序よく盛り付けていく構成になっており、機能の組み合わせ方や、実践的な使い方をラーニングできます。3Dゲーム開発の一般的なワークフローやプロセスも解説し、章の構成も実際の開発プロセスに近づけました。今使っている人にも、これから始める人にも、すべてのUE4ユーザーにオススメの一冊!

開発系読み物

IT エンジニアのための「人生戦略」の教科書
技術を武器に、充実した人生を送るための「ビジネス」と「マインドセット」

マイナビ出版
平城寿（著者） 256 ページ
価格：2,462 円（PDF・EPUB）

開発ツール

Unreal Engine 4 で極めるゲーム開発

ボーンデジタル
湊和久（著者）
592 ページ
価格：4,860 円（PDF）

Tech Book Zone Manatee

「開発技法」

3.15 2017

ワークフローを疑似体験！ GitHub が初歩からわかる

開発ツール

3

Web 制作者のための GitHub の教科書 チームの効率を最大化する 共同開発ツール

Web 制作における「GitHub」の使い方が、実際のワークフローをイメージしながら理解できます。「そもそもどんなサービスなの ?」「どういうときにどの機能を使えばいいの ?」といった初歩の疑問から解説します。

インプレス
塩谷啓・紫竹佑騎・原一成・平木聡（著者）
224 ページ　価格：2,052 円（PDF）

Docker が利用される 現場のノウハウが凝縮！

開発ツール

4

Docker 実践ガイド

Docker が利用される環境や背景をはじめ、導入前のシステム設計、基本的な利用方法、Dockerfile による自動化の手法、プロセッサ、ネットワーク、ストレージなどの資源管理、管理・監視ツールについて解説します。

インプレス
古賀政純（著者）
328 ページ　価格：3,240 円（PDF）

チーム改善に活かす ITIL 悩めるリーダーにオススメ

開発系読み物

5

新米主任 ITIL 使ってチーム改善します！

化粧品メーカーで主任に昇格した友原京子。異動先は問題だらけのハチャメチャ部署だった…。『新人ガール ITIL 使って業務プロセス改善します !』の第 2 弾。英国生まれの IT 運用ノウハウ「ITIL」をチーム改善に活かします。

シーアンドアール研究所
沢渡あまね（著者）
304 ページ　価格：1,750 円（PDF）

プロトタイピングによって 初期段階での可能性を探る

開発系読み物

6

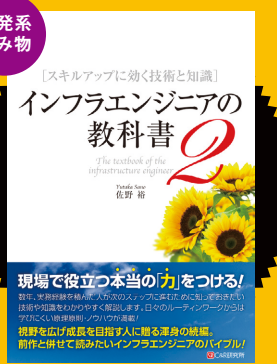
プロトタイピング実践ガイド スマホアプリの効率的なデザイン手法

本書で解説するプロトタイピングは、紙などを使った「低精度プロトタイピング」を中心とした手法です。設計フェーズの早期段階から作成し、検証と改善によって、機能要件や UI 設計、デザインを具現化していけます。

インプレス
深津貴之・荻野博章（著者）
240 ページ　価格：2,592 円（PDF）

インフラエンジニアの 必須知識をていねいに解説

開発系読み物



7

インフラエンジニアの教科書 2 スキルアップに効く技術と知識

数年間インフラエンジニアの経験を積んでいても「自分は詳しく知らないし、他の人に説明できない」といったことがあります。本書は実務経験を積んだインフラエンジニアを対象に、必須知識をわかりやすく解説します。

シーアンドアール研究所
佐野裕（著者）　価格：2,070 円（PDF・EPUB）

エミュレータ制作を通して コンピュータの中身を理解

開発系読み物

8

自作エミュレータで学ぶ x86 アーキテクチャ コンピュータが動く仕組みを徹底理解！

機械語やアセンブリ言語が CPU でどう実行されるか意識することはめったにありません。本書ではエミュレータの制作を通して x86 CPU の仕組み、メモリ・キーボード・ディスプレイといった部品と CPU の関わりを学びます。

マイナビ出版　内田公太・上川大介（著者）　196 ページ
価格：2,324 円（PDF）