3-1 コンピュータの基本構造

重要度 ★

コンピュータのハードウェアは，制御装置，演算装置，記憶装置，入力装置，出力装置という五つの要素(**五大装置**)に大別できます。ハードウェアの仕組みを学ぶということは，これらの要素を一つひとつ学習するということです。それらの詳細を学ぶ前に，まず全体像を理解してください。

さまざまなコンピュータ

最も身近なコンピュータは**パーソナルコンピュータ**（本書では，**パソコン**とよぶことにします）でしょう。もともとは個人利用を目的とした小型コンピュータで，略して**PC**と書かれることも多く，家庭や企業，学校など社会のあらゆる場面で利用されています。

パソコンは，その形状により，デスクトップ型，ノート型などに分けることができます。また，携帯電話に様々な機能を統合した端末（スマートフォンなど）やタブレット端末も普及しています。

左からデスクトップ型，ノート型

パソコンのほかにも次のようなコンピュータがあります。

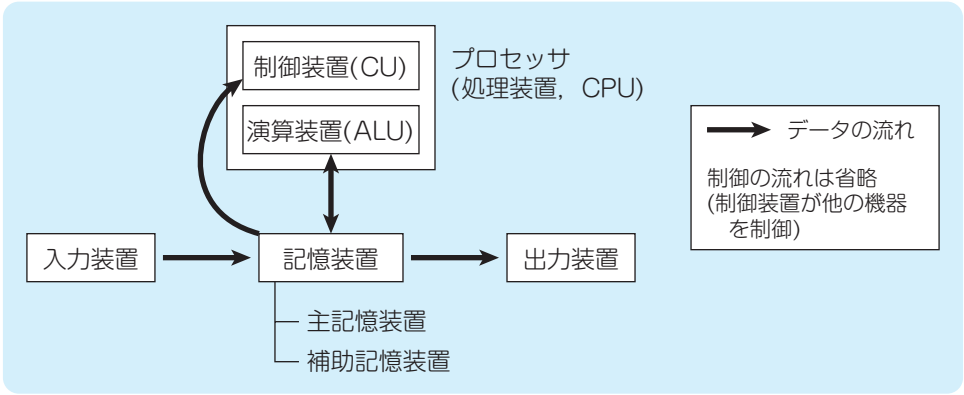
名 称	説 明
スーパーコンピュータ	最新の技術により，超高性能を実現したコンピュータ。気象予測など限られた時間内に大規模な科学技術計算を高速に行うさいに用いられる。
汎用(はんよう)コンピュータ	大型のコンピュータ。科学技術計算や事務処理計算など多目的(汎用的)に利用することができる。“**メインフレーム**”とよばれる。
ワークステーション	パソコンより多少性能が優れているコンピュータ。主に，研究用，建築物や機械部品の設計などに使われる。
マイクロコンピュータ	家電製品などに組み込まれている超小型のコンピュータ。

五大装置

ハードウェアとは，機器そのもののことです。コンピュータのハードウェアは，大きく分けて，

「制御装置」，「演算装置」，「記憶装置」，「入力装置」，「出力装置」

の五つの装置から構成されています。これらの装置を**五大装置**とよびます。



名　称	役　割	例
制御装置	各機器を制御する	プロセッサ
演算装置	各種の演算を行う	プロセッサ
記憶装置	プログラムやデータ，処理結果を記憶する	メモリ，ハードディスクなど
入力装置	プログラムやデータを入力する	キーボードなど
出力装置	処理結果を出力する	ディスプレイ，プリンタなど

上図中の矢印はデータの流れです。プログラムやデータ，処理結果などはいったん記憶装置に格納され，必要な装置に送られます。

制御装置（CU）と演算装置（ALU）で構成される部分は，一般に**プロセッサ**（**処理装置**）とよばれます。**CPU**(Central Processing Unit：中央処理装置）やMPU(Micro Processing Unit：超小型処理装置）などとよばれることもあります。

従来は，一つのプロセッサ内に，中核となる処理ユニット（コア）を一つ配置するのが一般的でした。最近では高性能化や省電力化のため，一つのプロセッサ内に複数のコアを配置して，並列に処理を進める手法がとられるようになってきています。このようなプロセッサの構成を**マルチコア**といいます。

また，記憶装置は**主記憶装置**と**補助記憶装置**に分けられ，主記憶装置のことを**メインメモリ**または単に**メモリ**とよぶこともあります。入力装置と出力装置のことをまとめて**入出力装置**とよぶこともあります。

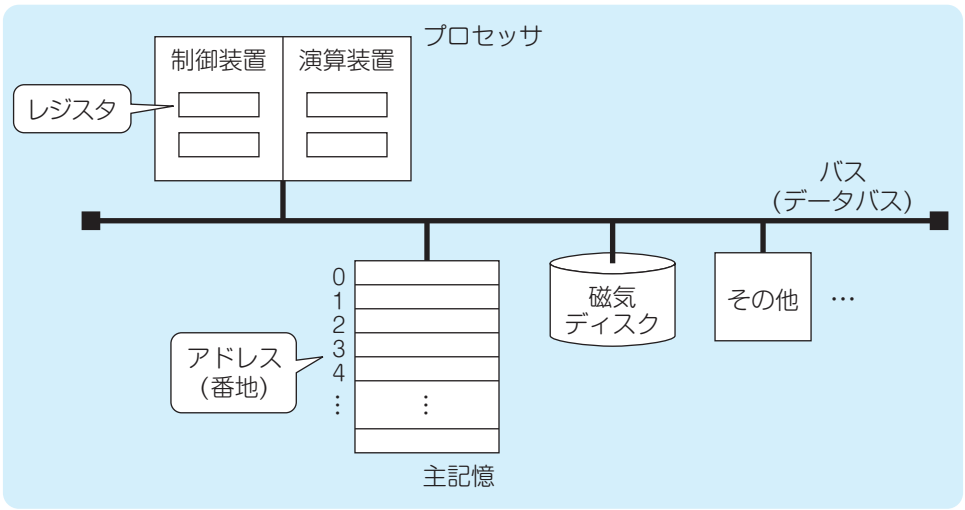
なお，炊飯器やDVDレコーダー，携帯電話などのデジタル家電製品の制御に用いられるマイクロコンピュータでは，これら五大装置の機能が一つの半導体チップ（半導体部品）上に搭載されています。このようなマイクロコンピュータやシステムLSIなどを利用したシステムを，エンベデッドシステム（組込みシステム）とよびます。

ノイマン型コンピュータ

現在のコンピュータの形式は，フォン・ノイマン(Von Neumann)によって提唱されました。ノイマンは，プログラムを記憶装置に格納すること(**プログラム内蔵方式**，**プログラム格納方式**)，それを1行ずつ実行すること(**逐次制御方式**)を考え，それを実現するための要素として五大装置を提案したのです。

コンピュータの基本構造

プロセッサと主記憶が結ばれていなければ，主記憶上の命令を実行することはできません。入出力装置も例外ではありません。コンピュータを構成する五大装置は，すべて何らかの形で結ばれていなければならないのです。



コンピュータ内部には，データ(プログラムも含む)を流すための回路や回線があります。これを**バス**といいます。バスは人間でいえば神経にあたり，あらゆるデータはバスを介して目的の機器に届けられます。実際には機器の速度や内部機器／外部機器の区別によって，接続するバスを分けています。データを送るためのバスをデータバス，接続機器のアドレスを指定するためのバスをアドレスバスとよぶこともあります。

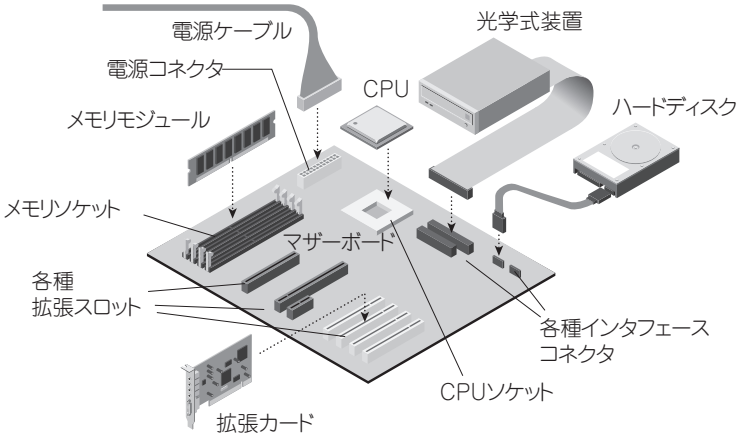
なお，CPUとチップセット，各装置を結ぶバスを**システムバス**とよびます。パソコンでは主記憶装置を結ぶバスをメモリバスとよぶこともあります。

主記憶には，領域を指定するための番号が振られています。この番号を**アドレス(address：番地)**とよびます。プロセッサはアドレスを指定することで，実行する命令や必要なデータを取り出します。

プロセッサは，主記憶から取り出した命令やデータを保持するために，非常に小さな(ただし高速な)メモリをもちます。これを**レジスタ**といいます。

参考：マザーボードとチップセット

パソコンでは，CPUや記憶装置などが一つの回路基板上に配置されます。このような回路基板をマザーボードとよびます。また，主記憶装置など各装置の動作を制御したり，データの受け渡しを管理する半導体素子(チップ)群が用意されており，これをチップセットとよびます。

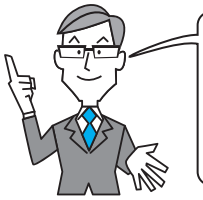


重要ポイント

- **五大装置：制御装置，演算装置，記憶装置，入力装置，出力装置**
- **プロセッサ (CPU) ＝制御装置＋演算装置**
- **プログラム内蔵方式：記憶装置に格納されたプログラムを読み出し，実行する方式**

3-2 プロセッサの構成要素と命令実行

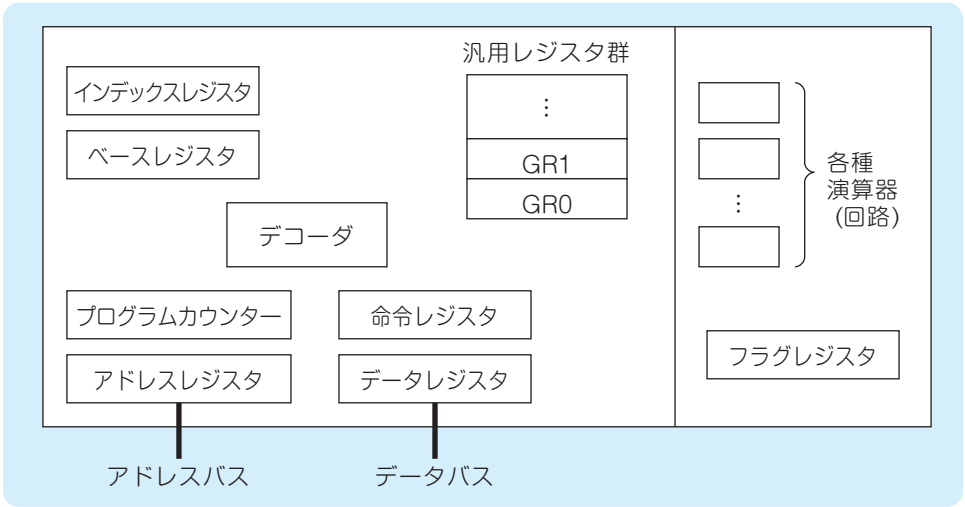
重要度 ★★



プロセッサは命令やデータを主記憶から取り出して実行します。と言葉でいえば簡単ですが，それを行うためにはさまざまな要素が協調しています。ここでは，プロセッサの構成要素がどのような役割を担っているかとともに，命令の実行過程を理解してください。

プロセッサの構成要素

まず，プロセッサの構成要素を示します。なお，ここで例示するものはあくまでも要素の一部であり，実際のプロセッサはより多くの要素から構成されています。また，アーキテクチャ（プロセッサの設計）によって要素が異なることもあります。



プロセッサは，主記憶装置（メモリ）から読み込んだ命令やデータなどを**レジスタ**（**プロセッサ内の高速な記憶装置**）に格納して処理します。レジスタは，役割によっていくつかに分類されます。各レジスタの役割を特化することで，効率のよい処理が行われます。

なお，ここでいう“命令”とは，コンピュータが直接理解して実行できる機械語の命令のことを指します。

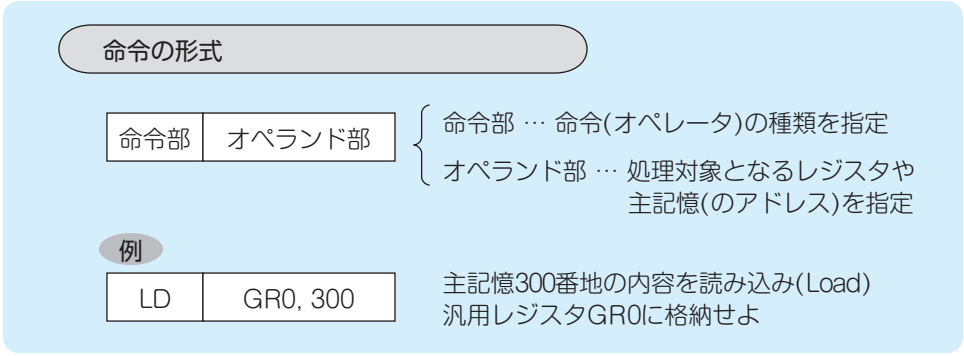
名　称	役　割
アドレスレジスタ	主記憶上のアドレスを保持するレジスタ
データレジスタ	主記憶から取り出した内容を保持するレジスタ
プログラムカウンター	次に実行する命令のアドレスを保持するレジスタ
命令レジスタ	主記憶から取り出した命令を保持するレジスタ
インデックスレジスタ	有効アドレスの計算に用いるレジスタ。指標レジスタともいう
ベースレジスタ	有効アドレスの計算に用いるレジスタ
汎用レジスタ	処理結果の格納やインデックスレジスタ，ベースレジスタとして用いることのできる，役割を限定しないレジスタ
フラグレジスタ	演算の正負や大小比較の結果を保持するレジスタ
デコーダ	命令を解読し，必要な機器に制御信号を送る機器
各種演算器	加算回路や補数回路など，各種の演算に用いる機器

参考：アキュムレーター

演算対象となるデータや，一連の演算における途中結果などを保持するための「演算用のデータレジスタ」のことを，アキュムレーターとよぶ場合もあります。

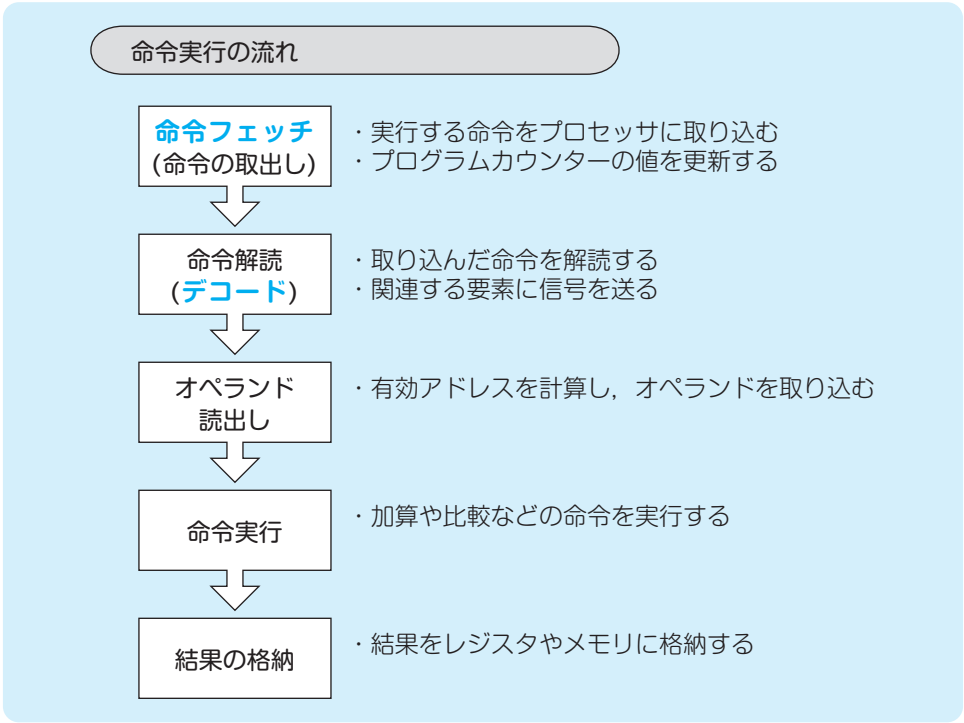
命令の形式と実行の流れ

命令（機械語命令）の形式の詳細は各コンピュータのアーキテクチャによって異なりますが，基本的には次のように「命令部＋オペランド部」という構成をとります。



命令部には命令の種類を，**オペランド部**には処理対象を指定します。たとえば，主記憶の300番地に値200が格納されていた場合，例に示した命令を実行することで，GR0に200が格納されます。

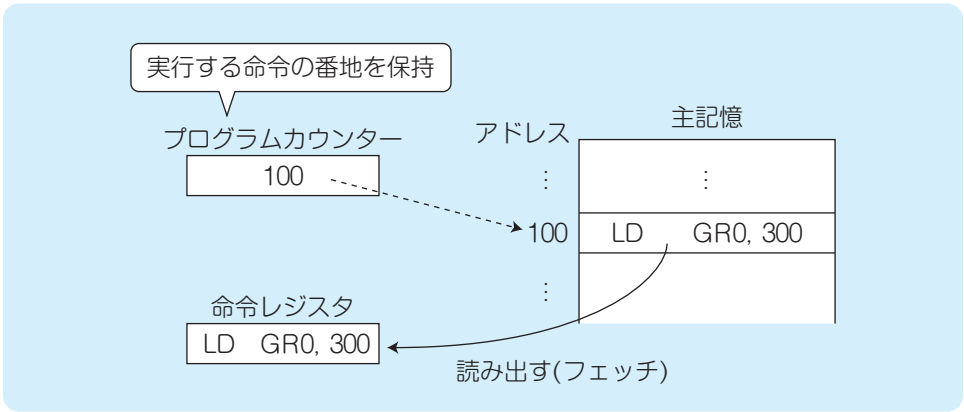
命令実行の基本的な流れは「読み込んだ後，実行する」というものです。もう少し詳しく流れを分割してみると，たとえば次のようになります（オペランドを用いる命令の場合）。



命令実行の過程を一つずつ詳しく見ていくことにしましょう。

①命令フェッチ(命令の取出し)

実行する命令を，プロセッサ内部の命令レジスタに取り込みます。

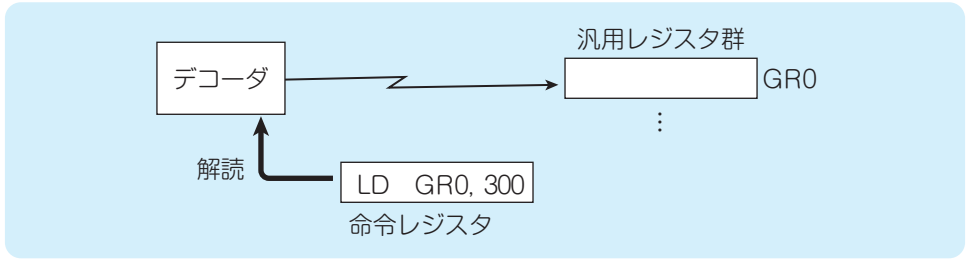


実行すべき命令の番地(メモリ上の格納位置)は，プログラムカウンターに保持されています。プロセッサはプログラムカウンターの内容が指す番地にアクセスし，そこに格納されている命令を読み出して，命令レジスタに格納します。

なお，命令の実行後は，次の命令を実行する準備として，プログラムカウンターの値を増やしておきます。

②命令解読(デコード)

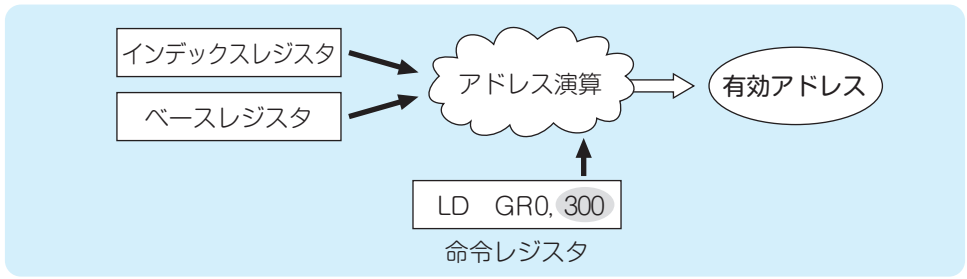
命令を解読(デコード)します。



命令レジスタにフェッチされた命令をデコーダが解読します。デコーダは「GR0を用いること」を解読し，GR0に制御信号を送ります。

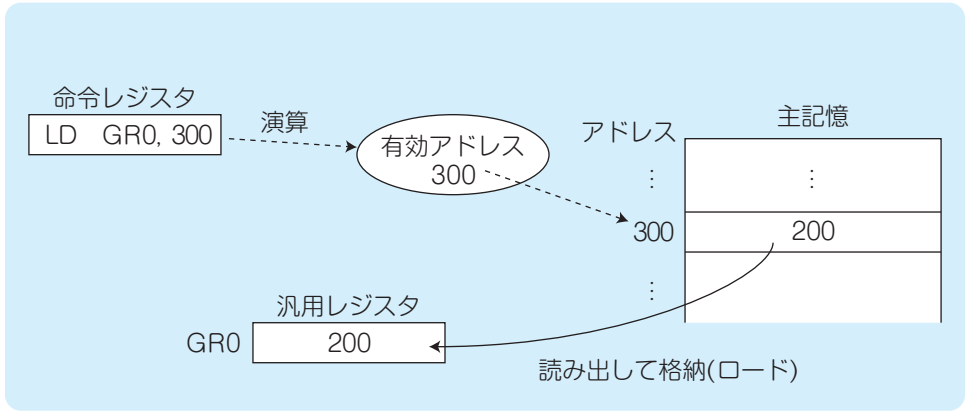
③オペランド読出し

オペランドが主記憶上のデータを表す場合は，主記憶へアクセスし，その内容をプロセッサ側に取り込む必要があります。まず，**有効アドレス**(実効アドレス)の計算を行います。有効アドレスとは，**処理対象となるデータが格納されているアドレス**です。



オペンランド部で指定したアドレス(**アドレス定数**といいます)がそのまま有効アドレスになる場合もあれば，アドレス定数と各種レジスタの内容をもとに有効アドレスを計算する場合もあります。

ここでは，有効アドレスがアドレス定数と同じ300であると仮定します。すると有効アドレスである300番地に格納されているデータが，主記憶からプロセッサに取り込まれます。



④命令実行

読み込んだオペランドなどの内容を用いて，命令部で指定された内容の演算処理を実行します。

⑤結果の格納

もし必要であれば，結果をメモリやレジスタに格納します。

この例では，主記憶から読み込んだ値200を，GR0に格納します。

参考：命令フェッチの詳細

厳密にいえば，主記憶から命令を取り出すときの動作手順は，

①プログラムカウンターの内容をアドレスレジスタに設定する
②アドレスレジスタが指す番地の内容をデータレジスタに転送する
③データレジスタの内容を命令レジスタに送る

のように，アドレスレジスタやデータレジスタを介したものになります。これはオペランド読出しの場合も同じです。ただし，ここまで詳細な手順が試験で問われる可能性は低いので，試験対策としては「プログラムカウンターの指す番地から読み出し，命令レジスタに入れる」のように概要だけ理解できていれば十分です。

参考：命令実行順序の解釈

文献によっては，命令実行の順序を

命令フェッチ → デコード → 実行及び有効アドレス計算 →
メモリアクセス(オペランド読出し) → 結果格納

のように説明しているものもあります。これは，その文献で想定しているコンピュータの設計が「メモリアクセスを伴う命令は，ロード(読出し)とストア(書込み)に限る」というものであり，オペランドを読み出してからさらに加算などに用いることは想定していないためです。

本書では，「必要なオペランドを取得してから，それを用いて演算を行う」という単純な解釈で実行の流れを説明しました。情報処理技術者試験においても，こちらの考えを用いて出題がなされることの方が多いようです。

重要ポイント

- **命令実行の流れ：**
  命令フェッチ→デコード→オペランド読出し→実行→結果格納
- **レジスタ：プロセッサ内で命令やデータの内容を保持**
- **デコーダ：命令を解読し，制御信号を送る**
- **プログラムカウンター：実行中または次に実行する命令のアドレスを保持**

3-3　プロセッサの設計と高速化

重要度　★★

命令実行の過程を把握したら，次に実行の高速化について見ていきましょう。あらゆるハードウェアは高速化を目指します。プロセッサもその例外ではありません。というよりも，プロセッサの高速化がコンピュータ全体の高速化を牽引したといえるかもしれません。
ここでは，高速化のための考え方・プロセッサの設計思想を紹介しつつ，具体的な高速化技法について説明します。

プロセッサの設計思想や高速化技法を理解するためには，関連するさまざまな知識が必要です。そこでまず，本テーマで説明する主な知識・技法を一覧しておきます。

命令パイプライン	CPU高速化の基礎となる技法 1プロセッサの中で複数の命令を並列実行する
ハザード	命令パイプラインを阻害する要因
スーパーパイプライン	命令パイプラインを高度化し，並列度を上げる技法
クロック周波数	プロセッサの動作テンポ
CPI	1命令の実行に必要な平均クロック数
スーパースカラ	1CPUの中に複数の実行ユニットを設けて並列動作させる技法
VLIW	複数の実行ユニットを設けて，長形式命令を実行させる技法

RISCとCISC

まず，プロセッサの設計思想について説明します。プロセッサ設計に関する大きな流れには，大きく**RISC**（リスク）と**CISC**（シスク）があります。

プロセッサアーキテクチャ
- **RISC** …縮小命令セットコンピュータ (Reduced Instruction Set Computer)
  - ・実行する命令を使用頻度の高い**単純な命令に限定**
  - ・命令の解読をワイヤードロジック方式で行うことが多い
- **CISC** …複合命令セットコンピュータ (Complex Instruction Set Computer)
  - ・単純な命令から**高機能命令まで用意**
  - ・命令の解読をマイクロプログラム制御方式で行うことが多い

ワイヤードロジック方式は，命令の解読をハードウェアで行う方式です。命令解読は高速ですが，複雑な命令の実現や拡張に対応するのが困難という特徴があります。

一方，マイクロプログラム制御方式は，プロセッサ内部に小さなプログラム（マイクロプログラム）を用意し，これを実行して命令を解読するものです。ワイヤードロジック方式と比べると速度では不利ですが，命令の追加や拡張に柔軟に対応できるという特徴があります。

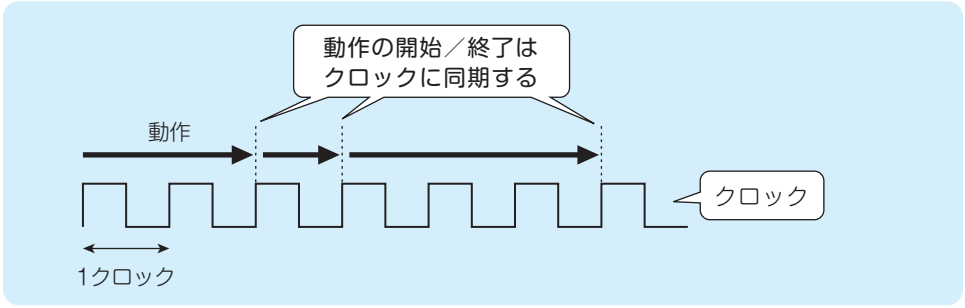
このような特徴から，単純な命令だけを用意するRISCではワイヤードロジック方式が，複雑な命令を多数用意するCISCではマイクロプログラム方式が採用されることが多くなっています。

クロック周波数とCPI

プロセッサの高速化は，クロック周波数とCPIが重要な要素となります。

クロックとは，コンピュータ内で発している周期的な信号です。また，**クロック周波数**とは，単位時間当たりのクロックが発せられる回数であり，プロセッサの動作テンポを表します。プロセッサや各装置は，クロックに同期して（タイミングを合わせて）動作します。

このため，アーキテクチャが同じプロセッサであれば，クロック周波数が高いほど性能が高い（高速である）といえます。ただし，アーキテクチャが異なるプロセッサの場合は，1クロック当たりに行える処理の内容が異なるため，一概にクロック周波数だけでの性能比較はできません。



クロック周波数の単位はHz（ヘルツ）です。たとえば，「1GHz（ギガヘルツ）で動作する」ということは「プロセッサが1秒間に1,000,000,000回のテンポを刻む」ということです。このときの1クロック当たりの時間は1ナノ秒になります。

CPI(Cycles Per Instruction)は，**1命令の実行に要するクロック数**のことです。たとえば，1GHzのプロセッサでCPI＝10の命令は，実行に10ナノ秒を要する計算になります。CPIを減らせれば，それだけ少ないクロック数で命令実行を行えると考えられます。

命令パイプライン

プロセッサの高速化の基礎となる技法が，**命令パイプライン**です。

1命令の実行は，たとえば，

命令フェッチ(IF) → 命令解読(ID) → オペランド読出し(OF)
→ 命令実行(EX) → 結果の格納(WB)

といったように複数の段階(**ステージ**)から構成され，段階ごとに別々の実行ユニット(演算器，演算ユニットともいう)を使うのが一般的です。単純な制御では1命令のすべての段階を終了してから次の命令に取り掛かることになりますが，「IF用演算器は，1個目の命令のIFを実行したら，すぐに2個目の命令のIFにとりかかる」ようにすれば，段階を一つずつずらしながら，バケツリレーのようにして複数の命令を並列に実行させるということも可能です。この「**複数の命令を並列に実行する**」というのが，命令パイプラインの考え方です。

図のように，CPIが10(各ステージ2クロック×5)の命令を，5段階で並列化した命令パイプラインで実行すれば，平均的なCPIは10/5＝2に近づきます。

RISCと命令パイプライン

一般的に単純な命令ほど，命令パイプラインの各ステージの処理時間が均質化し，パイプラインの段数(並列度)を高くしやすいといわれています。つまり，RISCと命令パイプラインは非常に相性がよいのです。

ハザード

命令パイプラインの導入によって，必ずCPIが「1/段数」に減少するわけではありません。なぜなら，プログラムを実行する過程では，パイプラインを阻害するようなさまざまな現象が発生するからです。

命令パイプラインを阻害するさまざまな現象や要因を，**ハザード**とよびます。ハザードが発生した場合には，パイプラインをいったん停止(パイプライン・ストール)し，改めて実行を開始しなければなりません。

参考：ハザードの種類

ハザードには大きく次の三つがあります。

制御ハザード：分岐命令の実行などによって生じる
データハザード：先行する命令の実行結果を参照することによって生じる
構造ハザード：異なる命令がハードウェアを競合する(奪い合う)ことによって生じる

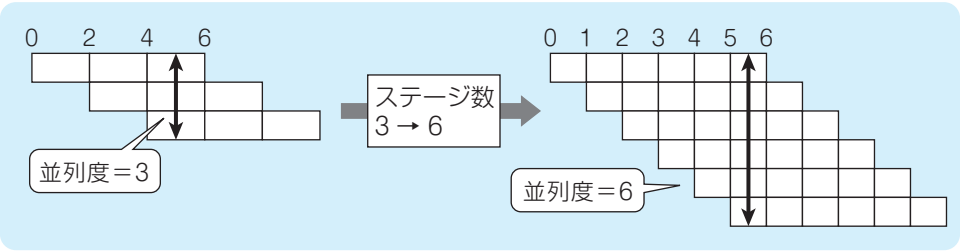
参考：投機実行

プログラムの実行においては，分岐命令があると次に実行される命令の候補が複数存在することになります。次に実行する命令の先読みを行うパイプライン制御において，分岐条件が確定する前に分岐先を予測し，その分岐先にある命令群をパイプラインに投入する技術を投機実行とよびます。

ハザードを完全に防止し，パイプラインを常に最適な状態に保つ手段は残念ながらありませんが，投機実行により制御ハザードの発生をある程度抑えることは可能です。

スーパーパイプライン

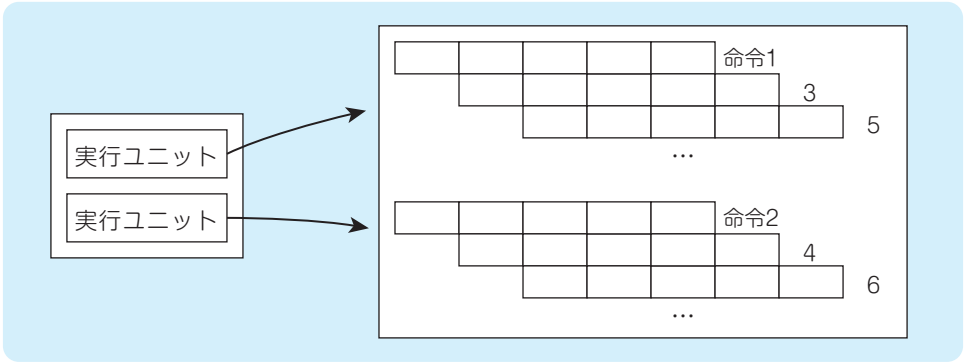
命令の実行過程をより細かな段階に分ける(段数を多くする)ことで，命令パイプラインの並列度が上がり，結果として平均的なCPIが減少します。この技術を**スーパーパイプライン**とよびます。



命令の段数を多くすれば，1段階当たりのクロック数は1に近づきます。そうなれば，平均的なCPIも1に近づきます。

スーパースカラ

どれだけパイプライン処理を効率化しても，CPIが1を下回ることはありません。この壁を乗り越えるために，**実行ユニットを複数用意して，パイプライン自体を多重化する技術**のことを**スーパースカラ**とよびます。

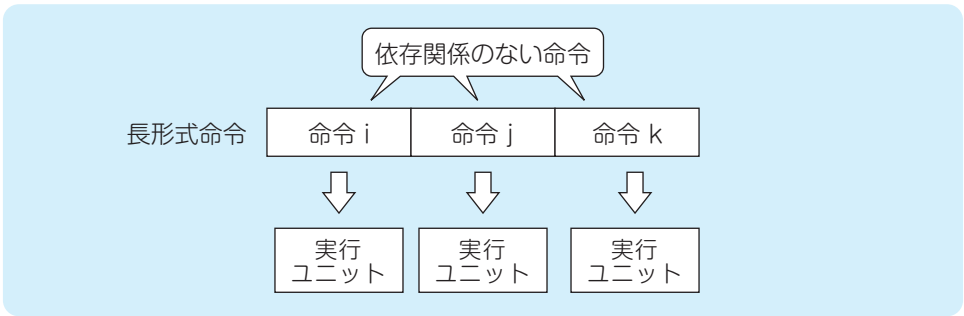


図の例ではプロセッサ内部に実行ユニットを二つ用意し，各ユニットで命令パイプラインを同時に稼働させています。

2本のパイプラインが同時に動けばCPIは0.5に近づくはずですが，そう単純にはいきません。なぜなら並列度が高くなればそれだけハザードの発生も増えるからです（スーパーパイプラインにも同じことがいえます）。

VLIW(Very Long Instruction Word)

ハザードがなぜ生じるかといえば，それは命令に依存関係があるからです。そこで，依存関係のない命令を一つの長形式命令に編集し，それを複数の実行ユニットで同時に実行させる技術が生まれました。このような技術を**VLIW**とよびます。



長形式命令の編集はコンパイラが行います。効果的なVLIWには，高性能なコンパイラが不可欠です。

参考：in order実行とout of order実行

プログラム中の命令を，実行の順序どおりにフェッチして実行を進める方式を，in order実行とよびます。命令パイプラインは，各ステージが重なり合いながらも基本的にはin orderで実行されます。

命令をin orderで実行する限りは高速化には限界があります。高速化のためには，どこかで命令の実行順序とは関係なく，準備の整った命令から実行を開始させなければなりません。これをout of order実行といいます。

関連用語

GPU(Graphics Processing Unit)

GPUは，3次元グラフィックスなどの画像処理に用いられる演算ユニットであり，多数の座標情報に関する並列処理を高速に行うために汎用的な行列演算ユニットを搭載しています。この行列演算は，ディープラーニングの学習におけるニューラルネットワークの処理においても用いられるので，GPUを機械学習などに利用することによって，高速に処理を行うことができます。

重要ポイント

【プロセッサ性能】

- **クロック周波数：1秒間当たりのクロック数(単位：Hz)**
- **CPI：1命令の実行に必要な平均クロック数**
- **1命令の実行に必要な時間＝1クロックの時間×CPI**

【高速化技法】

- **命令パイプライン：命令を複数のステージに分け，ステージをずらしながら，複数の命令を並列に実行させる**
- **パイプラインの発展系 … スーパーパイプライン(ステージを細かくする)**
  スーパースカラ(パイプラインを複数本用意する)
- **VLIW：独立に実行できる複数の命令は，一つの命令にまとめる**