- 第 1 章　コンピュータと工学　（1 コマ）
- 第 2 章　コンピュータシステム　（1 コマ）
- 第 3 章　論理回路　（計 4 コマ）
  - ◇ 3.1　論理回路の数学　（1 コマ）
  - ◇ 3.2　論理関数の表現　（1 コマ）
  - ◇ 3.3　論理回路の設計　（2 コマ）
- 第 4 章　コンピュータアーキテクチャ　（計 6 コマ）
  - ◇ 4.1　基本アーキテクチャ　（1 コマ）
  - ◇ 4.2　内部装置のアーキテクチャ　（3 コマ）
  - ◇ 4.3　外部装置のアーキテクチャ　（2 コマ）
- 第 5 章　オペレーティングシステム　（計 3 コマ）
  - ◇ 5.1　OS の役割と機能　（1 コマ）
  - ◇ 5.2　プロセッサと メモリの管理　（1 コマ）
  - ◇ 5.3　外部装置の管理と制御　（1 コマ）

本書では，「コンピュータ工学」に関する重要な事項の説明は，本文の各所で，「枠囲み」で個条書きにしてある．また，本書の核となる術語は「定義」として，また，「定理」はできるだけ証明とともに，それぞれ枠囲みで明示してある．さらに，説明事項を実際的に補足するために，本文の各所に，「例題」を解答例付きで設けてある． 各章末には，章ごとに学習した事項についての理解度をチェックするために， その章全体に関する「演習問題」を置いてある． これらの演習問題の略解は， 本書の末尾にまとめてある．

また，補足的な説明および「コンピュータ工学」以外の術語や参考情報については，本文の論旨や展開を妨げないように，〔注意〕，《参考》あるいは「注釈」として本文の欄外に切り出してある．

加えて，私たちの身近にある「コンピュータ」や「工学」に関する 6 つの話題について，やわらかくまたかみ砕いて紹介する枠囲みのコラムを，各章および本書の末尾に，【鳥瞰（ちょうかん）】と名付けて置いてあるので，学習や講義の合間の息抜きに使ってほしい．

最後に，いつも陰ながら励まし応援してくれる 妻・真木子 に感謝する．

2014 年 盛秋　　京都・松ヶ崎あるいは岩倉にて

柴山　潔

た基本論理演算と基本論理素子との対応関係にしたがうと，多項（$n$ 項）の AND 演算および OR 演算のそれぞれの機能を実現できる多入力（$n$ 入力）の AND 素子および OR 素子が実在する．

**多入力 AND 素子**

$$\underbrace{A \cdot B \cdot \cdots}_{n\text{ 項}} = Z \tag{3.47}$$

を実現する．$n$ 入力で，$n$ 本の入力 $A$, $B$, $\cdots$ のすべてが “1” のときだけ，出力 $Z$ を “1” とし，それ以外では，出力 $Z$ を “0” とする論理素子（図 3.13 に記号表現を示す）である．

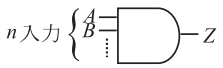


**図 3.13**　$n$ 入力 AND 素子

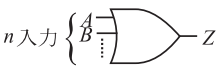


**図 3.14**　$n$ 入力 OR 素子

**多入力 OR 素子**

$$\underbrace{A + B + \cdots}_{n\text{ 項}} = Z \tag{3.48}$$

を実現する．$n$ 入力で，$n$ 本の入力 $A$, $B$, $\cdots$ のすべてが “0” のときだけ，出力 $Z$ を “0” とし，それ以外では，出力 $Z$ を “1” とする論理素子（図 3.14 に記号表現を示す）である．

多項 AND（OR）演算での結合則（定理 3.10，47 ページ）によると，1 個の $\underline{n}$ 項 AND（OR）演算は $(n-1)$ 個の $\underline{2}$ 項 AND（OR）演算の集まりと見なせて，それらの 2 項 AND（OR）演算をどんな順序で行ってもよい．また，1 個の 2 項 AND（OR）演算は 1 個の 2 入力 AND（OR）素子に対応する．結局，1 個の $\underline{\boldsymbol{n}}$ 入力 AND（OR）素子は $(n-1)$ 個の $\underline{\mathbf{2}}$ 入力 AND（OR）素子によって構成する組み合わせ回路と等価†となる．

▶〔注意〕
「多入力 OR 素子」の定義の下の「多項 AND（OR）演算…」で始まる段落は AND 演算について説明しているが，同様の説明は OR 演算にも適用できる．この段落を「OR 演算についての説明」とするには，文中のすべての “AND” をその直後のカッコ内の “OR” で読み替えればよい．
本書の以降でも，このような段落全体での “AND” と “OR” の総読み替えを「AND（OR）」と書く．

▶† 等価
機能が同一の論理回路，すなわち入力（信号）の論理値のすべての組み合わせに対する出力（信号）の論理値のそれぞれがすべて等しい組み合わせ回路を「等価」（論理関数でいう「同値」と同じ意味である）あるいは「等価回路」という．

**例　題**

3.13　$A + B + C + D + E = Z$ という 5 項 OR 演算に 1 対 1 対応する 5 入力 OR 素子を 2 入力 OR 素子だけで構成しなさい．

（解）　この 5 項 OR 演算は，たとえば，

$$\Big((A + B) + \big((C + D) + E\big)\Big) = Z$$

のようなカッコによる演算順序の指定にしたがう 4 個の OR 演算と見なせる．したがって，問題式に対応する 1 個の 5 入力 OR 素子は

として使用し，数学でも厳密に定義できる．次の 2 種類が代表的である．

(a) **整数** (integer)：数直線上に精度 “1” で存在する離散値である．数直線上で範囲を限ると，その中に存在する整数は有限個である．

- 演算：**算術演算**（演算機構については，4.2.3 項で詳述）を適用する．
- 格納形式：整数は範囲を限ると有限個であることを利用して，限られた範囲や個数の整数値を表現する**整数表現**（4.2.3 項［1］で述べる「固定小数点数表現」の代表である）が一般的である．

(b) **実数** (real)：数直線上のあらゆる所に存在する連続値である．数直線上で範囲を限っても，その中に存在する実数は無数にある．

- 演算：整数と同様に，**算術演算**（演算機構については，4.2.3 項で詳述）を適用する．
- 格納形式：領域が無限であるので，これらを有限のハードウェア機構で取り扱うための工夫が必要となる．したがって，主として，**浮動小数点数表現**（4.2.3 項［7］で詳述）を用いる．

大半のコンピュータは，整数と実数の 2 種類の**演算器**（4.2.3 項で詳述）と，それによって演算や操作を指定するマシン命令セット（4.1.3 項［2］を参照）を装備している．

(B) **論理値** (Boolean)：2.3.1 項で述べたように，“1”（真）あるいは “0”（偽）の 2 種類すなわち 2 値である．

- 演算：論理代数にしたがう**論理演算**（3.1.1 項で詳述）を適用する．
- 格納形式：1 個の論理値当たり 1 ビットで済むが，ハードウェアの有効利用のために 1〜数バイトに複数個の論理値を詰め込んで（「パック (pack) する」という），ビット列†として格納あるいは処理するのが普通である．

(C) **2 進コード** (binary code)：有限個の均質要素から成る数値や文字の集合を，別の記号列の集合（「コード」あるいは「符号」である）に変換したり（「エンコード」(encode),「符号化」,「コード化」という），逆に，元のデータ型に戻したり（「デコード」(decode),「復号」という）する．コードを，原則として，長さの等しい 2 進数やビット列で表現するとき，このコードを **2 進コード**と，また，2 進コードにエンコードすることを「2 進コード化」と，それぞれいう．2 進

▶ †ビット列
独立した論理値の集まり（列）をいう．
ビット列全体に対して、あるいは各ビットごとに論理演算を適用する．
1.1.3項で述べたように，図形，画像，音声などのアナログ情報は，コンピュータではデジタル化したデジタル情報として取り扱う．デジタル情報は量子化によってビット列や 2 進数値になっている．

もある．

5. **結果格納：演算器**の出口に置いた演算結果データを，2. でデコードした**デスチネーションオペランド**の指定にしたがって，データバスや内部バスを経由して**レジスタ**あるいは**メインメモリ**に書き込む．
6. **次命令アドレスの決定**：次に実行すべき命令のアドレスを PC に設定する．今実行している命令が演算命令（次の 4.1.3 項で詳述）ならば，単に PC をカウントアップ（count up; 増加順での計数）するだけでよい．一方，順序制御命令（次の 4.1.3 項で詳述）なら，命令機能にしたがって，次命令アドレスを生成して PC に設定する必要がある．

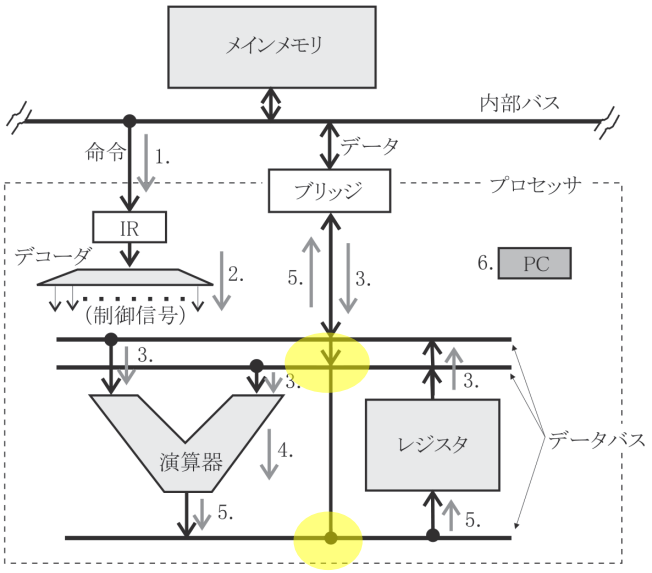


**図 4.13** 命令実行サイクルとハードウェア機構

この 1.〜5. のステージはこの順序でしかできない一連の操作であるが，6. のステージは，可能ならば，1.〜5. のステージのどれかと並行させてもよい．

1.〜5. の各ステージは，図 4.13 に示すように，バスを除くと，それぞれ相異なるハードウェア機構によって実現する．また，そうすることによって，各ステージのハードウェア機構は順に実行されるマシン命令列によって共用することができる．たとえば，先行して実行サイクルに入った命令 A が 3. のオペランド取り出しから 4. の実行ステージに到達すれば，A に後続する命令 B が 2. の命令デコードから 3. のオペランド取り出しのステージに入ることができる．ステージ処理ごとに用意されたハードウェア機構によって，複数のマシン命令の相異なるステージを実行するプロセッサの高速化（「命令パイプライン処理」という）については，4.2.2 項［8］で詳

整数と実数とを用いて算術演算を行うコンピュータの演算機構については，4.2.3 項で詳述する．

(B) **論理演算**：1 ビットの論理値どうしの演算であり，次の 3 種類がある．論理演算については，3.1.1 項［2］で詳述しているので，ここでは，演算名称の列挙だけに留める．
  (1) **否定**（ノット (NOT)）
  (2) **論理積**（アンド (AND)）
  (3) **論理和**（オア (OR)）

(C) **ビット列操作**：論理値を複数個まとめて（**ビット列**である，117 ページの注釈を参照），同時に操作する．
  (1) **論理演算**：ビット列の全部または一部に対して，まとめて (B) の論理演算を適用する．
  (2) **シフト** (shift) 演算：図 4.14 に示すように，ビット列をひとまとまりのデータとして，指定するビット数だけ左右に移動する（ずらす，「シフト」である）操作である．シフト操作のソースオペランドは，i) 方向（左か右か）；ii) シフトするビット数；である．シフト演算は，次のような操作に細分できる．
    (a) **論理シフト**：ビット列全体を独立した論理値（ビット）のまとまりとしてシフトする．
    (b) **算術シフト**：ビット列を算術データとしてシフト操作する．2 進数に対する算術シフトでは，$n$ ビットの左算術シフトが $\times 2^n$ の乗算に，$n$ ビットの右算術シフトが $\div 2^n$ の除算に，それぞれあたる．
    (c) **循環シフト**：シフトによって左（右）端からあふれたビット列を反対の右（左）端から詰め直す．

▶〔注意〕
「算術シフト」では，符号ビット（通常は最上位ビット）をシフト対象から外してそのまま保持する．すなわち，符号（ビット）以外のビット列がシフト対象である．
論理左（右）シフトでは，最右（左）のビットから新たに“0”を補充する．一方，算術左シフトでは，最右の最下位ビットから新たに“0”を，算術右シフトでは，最左（上位）の符号ビットの直右ビットから新たに符号ビットと同じ論理値（“0”か“1”）を，それぞれ補充する．

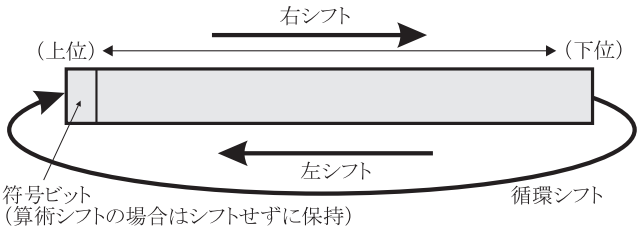


**図 4.14**　シフト演算

(D) **データ転送**：ソースオペランドで指定したデータを演算あるいは加工する操作である (A)～(C) に対して，データそのものはそのままで，その格納場所だけを移動する操作である．オペランドでは，移動元および移動先のそれぞれのレジスタ番号やメモリアドレスを指

(1) $(-7)_{10} = (\overline{0111})_2 = (1000)_2$　（例題 4.2 (1) の解）
$(+3)_{10} + (-7)_{10} = (0011)_2 + (1000)_2$
$= (1011)_2 = (\overline{0100})_2 = (-4)_{10}$

(2) $(-7)_{10} = (\overline{\overline{0111}})_2 = (1001)_2$　（例題 4.2 (2) の解）
$(+3)_{10} + (-7)_{10} = (0011)_2 + (1001)_2$
$= (1100)_2 = (\overline{\overline{0100}})_2 = (-4)_{10}$

---

### ［5］ 補数による加算

$(A - B)$ を行う減算器は，i) 補数器によって減数 $B$ の補数をとる，すなわち負数 $(-B)$ にする；ii) i) で得た $B$ の補数と被減数 $A$ との加算を行う；という 2 つの機構で構成できる．ただし，1 の補数を使う場合と 2 の補数を使う場合との加算手順は，それぞれ次に示すように，オーバフロー（132 ページの注釈を参照）の扱いや最上位ビットからの桁上げによる結果の補正の必要性の 2 点で違いがある．

(A) **1 の補数**による加算手順：

1. 符号ビットも含めて，すなわち，負数の場合は 1 の補数にしてから，被加数と加数を加算する．
2. 1. の結果，最上位ビットからの桁上げ（「エンドキャリ」(end carry) という）があれば，1. の加算結果にそのエンドキャリ “1” を加え（**補正**という），桁上げそのものは無視する．エンドキャリがなければ，加算結果の補正は不要である．この補正で用いる最上位ビットからのエンドキャリを**循環桁上げ**（「エンドアラウンドキャリ」(end-around carry)）という．
3. 残った結果が符号を含めた加算結果すなわち和である．和が負数（符号ビットが “1”）の場合は，1 の補数表現で示されている．

(B) **2 の補数**による加算手順（図 4.25 を併照）：

1. 符号ビットも含めて，すなわち，負数の場合は 2 の補数にしてから，被加数と加数を加算する．
2. 最上位ビットからの桁上げすなわちエンドキャリは**無視**する．
3. 残った結果が符号を含めた演算結果すなわち和である．和が負数（符号ビットが “1”）の場合は，2 の補数表現で示されている．

(B) の **2 の補数**による加算では，ハードウェア機構の規模の点で，「補正の有無のチェックおよび補正（+1 加算である）のための機構が不要」という長所が「（2 の）補数器には補正（+1 加算である）のための加算器が余分に必要」という短所を隠ぺいできる．したがって，現代のコンピュータでは，負数の加算すなわち減算は (B) の **2 の補数**表現で行うのが一般的で

いう．これらは，プロセッサ内に実装する専用レジスタである．

図 4.27 に示すように，プロセッサ内にあって MAR および MDR を備えるメイン メモリ管理機構（MMU (Memory Management Unit) という）が内部バスを経由するメイン メモリアクセスを管理制御する．

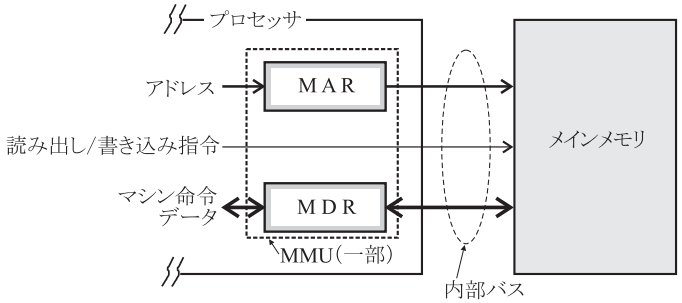


**図 4.27**　プロセッサによるメイン メモリへのアクセス

プロセッサの MMU（図 4.27 では一部）が MAR と MDR を使って行うメイン メモリへのアクセス操作は次のようになる．

(A) **読み出し**（リード (read)）

1. MAR に読み出し対象とする命令またはデータが格納してあるアドレスを設定する．
2. メイン メモリに読み出し指令を送る．
3. 読み出し対象の命令またはデータを，メイン メモリから読み出して，内部バスを経由して MDR に置く．

(B) **書き込み**（ライト (write)）

1. MAR に書き込み対象とするデータを格納するアドレスを設定する．
2. メイン メモリに書き込み指令を送る．
3. MDR にあらかじめ置いた書き込み対象のデータを，内部バスを経由して，メイン メモリへ書き込む．

▶〔注意〕
メインメモリに格納してあるマシン命令とデータのうち，マシン命令については，実行時は通常，プロセッサによる書き換えがない読み出し専用（「リードオンリ」(read only) という）である．したがって，プロセッサによる実行時のメインメモリへのアクセスでは，(A) の読み出しはマシン命令とデータの両方が対象となるが，(B) の書き込みの対象はデータのみである．

### ［3］ メモリの機能

現代のコンピュータは，メイン メモリ以外にも，いろいろなメモリ（次の［4］で詳述）をハードウェア装置として装備している．**メインメモリ**は，これらのメモリを代表する主要なメモリである．その意味で，メイン メモリが備える「命令とデータの (a) 格納；(b) プロセッサによる読み出しと書き込み」の 2 つの機能は，ハードウェア装置としての**メモリ**全般が備えている．

じである．すなわち，実際には，通信プログラムは次の手順で通信装置を制御する．（図 4.46 を参照）

1. OS：プロセッサが，**通信プログラム**（実際には，通信命令すなわち入出力命令を含むマシン命令列，5.3.2 項［1］［2］を参照）によって，通信コントローラに対して指令する．
2. **通信コントローラ：通信コマンド**（入出力コマンドである）列によって，通信装置に対して指令する．
3. **通信装置：コンピュータネットワーク**を介して通信する．

［7］ **通信機能におけるハードウェアとソフトウェアの機能分担**

データ通信機能におけるハードウェアとソフトウェアの機能分担方式，すなわち**通信アーキテクチャ**の代表例を示しておこう．

(a) **ハードウェア**：内部装置側にある**通信コントローラ**，**通信装置**および通信線（**外部バス**を含む信号線や**コンピュータネットワーク**そのもの）などのハードウェア機構である．

(b) OS：通信そのもの，通信管理，通信制御および通信プロトコル（［9］を参照）処理などの各機能を実現するプログラム（**通信プログラム**である）の実行を管理・制御するソフトウェア機能である．実際には，通信制御機能は**通信管理サービス**（5.3.2 項［1］を参照）という OS のシステムサービスによって実現する．通信管理サービスによる通信および通信制御の具体例としては，i) 通信プロトコルのチェックや実行；ii) データの送受信そのもの；iii) 通信線の多重使用の管理すなわち使用競合の解決；iv) 障害からの回復処理；などがある．通信機能における OS の分担については，5.3.2 項［2］に詳しくまとめてある．

(c) **ユーザプログラム**：通信機能あるいは通信処理機能の一部は共用できる形式のプログラム（「ライブラリ」という）としておき，必要に応じてユーザプログラムから呼び出して使用する．

［8］ **パケット交換**

現代のコンピュータの代表的なデータ通信方式は，(1) 送信側で送信データを**パケット**†(packet) に分解し，いったん蓄積して，空いている通信路を使用してパケットを順不同で転送する；(2) 受信側で受信したパケットをいったん蓄積し，受信データとして元のデータの形に組み立て直す；というパケット交換（図4.47を参照）である．また，パケット交換にしたがうコンピュータネットワークを**パケット交換ネットワーク**という．パケット交換は，「通信データを通信装置内にいったん蓄積（「バッファ」である）しておき，通信路の空きあるいは使用状況に適応するように，その伝送や

▶ †**パケット**
「フレーム」(frame) という一定長のデータブロックを複数個用いて構成する一定長以下の「データ通信時の単位データ」である．

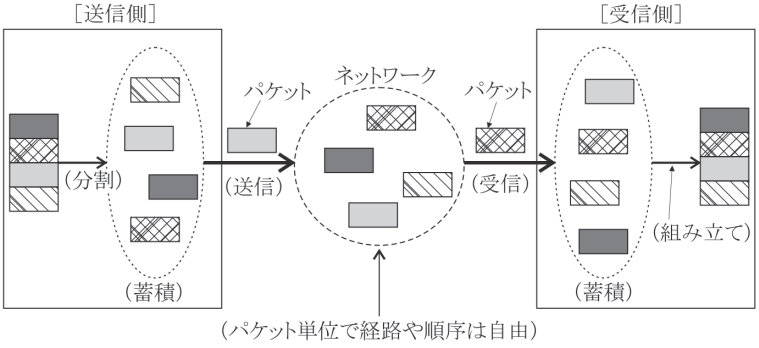


**図 4.47**　パケット交換

転送を制御しながら通信する」（**蓄積交換**という）方式の代表例である．

- 現代のコンピュータネットワークの主流は**パケット交換ネットワーク**である．

### ［9］ 標準通信プロトコルにおけるハードウェアとソフトウェアの機能分担

コンピュータネットワークの仕様をハードウェアとソフトウェアの機能分担方式すなわちネットワークアーキテクチャのある一定の機能レベルで統一あるいは標準化するのが**通信プロトコル**である．コンピュータネットワークの仕様を通信プロトコルによって統一あるいは標準化することで，「コンピュータやそのユーザが，物理的にも論理的にも，コンピュータネットワークの存在を意識せずにデータ通信できる」**ネットワーク透明性**が実現できる．

逆に言うと，ネットワーク透明性は「**通信プロトコル**として統一した共通あるいは標準ネットワークアーキテクチャを規定する」ことによって実現できる．したがって，通信アーキテクチャやネットワークアーキテクチャの設計では，そのコンピュータネットワークに接続する多種多様なコンピュータや他のコンピュータネットワークに対して示す**標準通信プロトコル**の確立が主となる．

ネットワーク透明性と通信プロトコルの関係については，特に OS の観点から，5.3.2 項［3］で詳述している．この 5.3.2 項［3］では，具体的な**標準通信プロトコル**として，(1) **データリンク層** (data-link layer)：入出力コントローラが発する通信コマンドによって実現する；(2) **ネットワーク層**，および (3) **トランスポート層** (transport layer)：内部装置で実行する OS プログラムで直接実現する；(4) **セッション層** (session layer)：応用プログラムで実現する；という 4 層（下層から上層へ (1)～(4) の順）の機能

- **ブロック置換**は，仮想メモリ機構のハードウェア機能部分であるブロック置換管理機構と分担する，OS による**割り込み処理**である．

### 5.2.3 ファイル管理 《種々雑多な大量ソフトウェアを簡潔に一元管理する》

**ファイル装置**は，コンピュータシステムのハードウェア装置としてのメモリをメインメモリとともに構成する．本項では，ファイル装置におけるメモリ管理機能である**ファイル管理**について詳述する．ファイル管理機能とは，具体的には，論理的情報の格納単位である**ファイル**をファイル装置という物理的なメモリ領域上で一元管理する OS 機能であり，**ファイルシステム**と呼ぶ OS のシステムサービスによって実現する．OS 機能としてのファイルシステムの原理は「ユーザプログラムからハードウェアであるファイル装置を隠ぺいする」，さらには，「ファイル装置へのファイルの格納形式およびユーザプログラムによるファイルの操作方式を一元化する」の 2 点である．

#### ［1］ ファイルとファイル装置

ソフトウェアである**ファイル**とそれを格納するとハードウェア装置としての**ファイル装置**の相違点を明らかにしておこう．

**ファイル**

情報（プログラムやデータ）を格納あるいは保持するために名前付けた論理的単位を**ファイル** (file) という．

ファイルは「コンピュータシステムのユーザが作成するソフトウェア」である．実際には，「ファイルを作成するユーザ」とは，ユーザプログラムや OS というソフトウェア自身である．また，ファイルは各々が，型† および論理的構造（**ファイル構造**という，後の［3］で詳述）を持つ．

ファイルの代表例としては，(a) **プログラム**：ユーザが直接作成し編集する，あるいはコンパイラが生成する；(b) **データ**：ユーザが作成する，あるいはユーザプログラムが生成する；などがある．また，図 5.20 に示すように，「ファイルを構成する最小単位は固定すなわち一定長のブロック（**ファイルブロック** (file block) という）とする」のが一般的である．

一方，「ファイルを実際に格納する物理的なハードウェア装置」が**ファイル装置**である．ファイル装置は，4.3.3 項で述べたように，「アクセス速度よりも容量を重視するメモリ階層」であり，隣接するメモリ階層であるメイ

▶ †**型**
4.1.2 項［10］［11］で述べた「データ型」と同義である．「タイプ」(type) ともいう．

▶〔**注意**〕
本章では，これまでに，「実行中→待ち状態遷移」を意味する「ブロック」と，単なる「かたまり」の意味である「ブロック」とが出てきた．
ここでの「ファイルブロック」の「ブロック」は後者の意味である．以降では，紛れがない場合に，「ファイルブロック」を単に「ブロック」という．

### 5.3.1 入出力制御 《人間とコンピュータとの情報の送受を取り仕切る》

OS は，ユーザプログラム（「ユーザ」を含む）から多種多様な**入出力装置**，特にその仕様を隠ぺいし，また，ユーザプログラムに対して，入出力装置を対象とする多種多様な制御機能を統一して提供する．本項では，「入出力装置を管理・制御する」（**入出力制御**という）OS 機能について詳述する．

#### ［1］ 入出力制御の必要性

OS による**入出力制御**とは，「多種多様な入出力装置の多種多様な入出力機能を，統一的にあるいは一元化して，管理・制御する」ことである．

**入出力制御**の要件は次の 2 点である．

(a) コンピュータの内部装置（具体的には，プロセッサ–メイン メモリ対），特にプロセッサに比べると格段に低速動作である**入出力装置**に対する**制御**機能をプロセッサやメイン メモリの本来の機能から独立させる．

- 非同期動作するプロセッサ–メイン メモリ対である内部装置と，入出力装置を代表とする外部装置とが，共用する種々のハードウェアおよびソフトウェアを相互に効率良く利用できるようになり，コンピュータシステムとしての全体性能が向上する．

(b) 人間が扱う多種多様な情報メディアに合わせて用意する種々の**入出力装置**と，プロセッサ–メイン メモリ対との接続形態の多彩な組み合わせを実現する．

- コンピュータ本体と入出力装置とのインタフェースを一元化し，その統一したインタフェースで多種多様な入出力装置の制御を簡潔に行える．

したがって，これらの要件を満たす**入出力制御**は，入出力装置がプロセッサやメイン メモリと非同期かつ独立に並行して動作するように制御する機能と言える．

コンピュータを使う人間すなわちユーザから見ると超高速に動作する**プロセッサ–メインメモリ対**と，コンピュータから見ると超低速の人間の動作が動作速度そのものに影響する**入出力装置**とは，独立して動作させる，また独立して動作する方が互いに性能を発揮できる．すなわち，コンピュータシステムにおいては，独立して実行できる機能はできる限り並行動作させることによって，特に，共用するハードウェア機構の効率的な活用が可能となる．

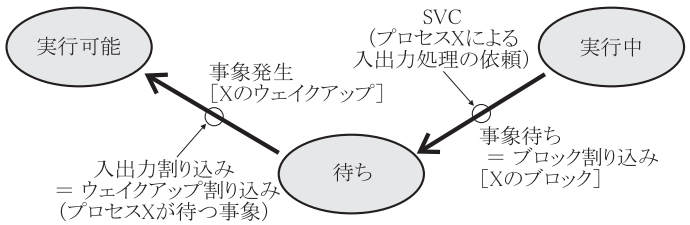


**図 5.23**　入出力処理におけるプロセス状態遷移

### [5] 割り込みによる入出力プロセスと入出力装置との通信

入出力処理を入出力装置に指令する入出力プロセス（“X” とする），および，その入出力プロセス X からの指令を受けて入出力処理を直接に行う入出力装置，のそれぞれが発生要因元となる割り込みとそれによるプロセス状態遷移について，時間順およびハードウェア装置別にまとめておこう．（図 5.23 を併照）

[OS]

1. 「入出力処理を OS に依頼する」入出力命令である SVC（「ブロック割り込み」である）を実行したプロセス X は自律的に**実行中→待ち**状態遷移（**ブロック**）する．
2. 1. の割り込みをきっかけとするプロセススイッチによって，別のプロセスを**実行可能→実行中**状態遷移させ，そのプロセスに切り替える．

[入出力装置]

3. OS（実際には，デバイスドライバ）からの指令によって入出力動作を開始する．
4. 入出力動作を完了すると，OS に対して，**入出力割り込み**という事象によって，「入出力動作の完了」を通知する．

[OS]

5. 入出力割り込みを受け付け，**割り込み処理**を開始する．「入出力動作の完了」という割り込み要因に対する割り込み処理とは，具体的には，(a) 入力の場合：入力された情報を内部装置内に取り込み，解読あるいは解析する入力処理プログラム（入力プロセス）を実行する；(b) 出力の場合：その出力動作を指令した出力命令に関係する出力処理プログラム（出力プロセス；例：新しいあるいは次の情報を作成し出力する）を実行する；などである．
6. 5. の割り込み処理をプロセス X として実行するために，「**入出力割り込み**（「ウェイクアップ割り込み」である）の発生」という事象を待っているプロセス X を**待ち→実行可能**状態遷移（**ウェイクアップ**）させ

▶〔注意〕
実際には，入力プロセスは，OSの始動とともに「入力」という事象を待つ待ち状態にあり，「入力」という事象の発生によってウェイクアップ(待ち→実行可能状態遷移）する．
一方，出力プロセスは，「出力」を指令する出力命令(SVC)のたびにブロック(実行中→待ち状態遷移)し，「当該出力動作の完了」という事象の発生によってウェイクアップ(待ち→実行可能状態遷移)する．

## 著者略歴

柴山　潔（しばやま　きよし）
1974年　京都大学工学部卒業
1979年　京都大学大学院工学研究科
博士後期課程単位修得退学
（京都大学工学博士）
2016年　京都工芸繊維大学退職
（京都工芸繊維大学名誉教授）

http://shibayam.sakura.ne.jp/

## 主要著書

並列記号処理（コロナ社，1991）
コンピュータアーキテクチャ（オーム社，1997）
ハードウェア入門（サイエンス社，1997）
コンピュータサイエンスで学ぶ 論理回路とその設計（近代科学社，1999）
改訂新版 コンピュータアーキテクチャの基礎（近代科学社，2003）
コンピュータサイエンスで学ぶ オペレーティングシステム ― OS学 ―（近代科学社，2007）

コンピュータ工学への招待



2015年2月28日　初版発行
2024年4月30日　初版第2刷発行

著者　柴山　潔
発行者　大塚浩昭
発行所　株式会社 近代科学社

〒101-0051
東京都千代田区神田神保町1丁目105番地
https://www.kindaikagaku.co.jp

京葉流通倉庫株式会社　ISBN978-4-7649-0475-0
定価はカバーに表示してあります．