

# 計算機科学

keisanki.tex

服部哲弥

19970211-15;17-25;28;0303-16;0401-07;10-15;

19970418;0530(Sato970513 etc.);

19970531(FAND etc.);

---

## 章目次

- 0 . 概観
- 1 . ハードウェア
- 2 . ソフトウェア
- 3 . アルゴリズム

## 第 章 概観

### § コンピュータ

道具	機能（働き；仕様）	実体（もの；実現）
コンピュータ	情報処理システム (§0)	*

‘\*’ の中身 (§1) . 電気がなければただの箱 .

数学的描像	Turing machine	データ	記録（記憶）	入出力	演算・制御
2000年現在	コンピュータ	2進数	磁気材料	電気 - 光・機械・電気	半導体

道具のもう一つの側面：人が使う . Interface . 標準化されたとは到底言えない . 他の側面に比べて明確な理論があるように見えない . 現在の課題 .

入力	スイッチ → 紙テープ・パンチカード → キーボード → マウス, OCR <sup>1</sup>
出力	ランプ → プリンター → スクリーン → multimedia (?)

マウス . (人が入力するための) 入力装置のうちでキーボードによらない (文字列によらない) ものの代表 .

#### マウス

文字列によらない入力という概念が何故重要か？

「(命令を知らない) 初心者でも使える」 . Graphical user interface (GUI) .  
Interface の標準化は、内部構造を覆い隠す方向 (black box 化) に進んでいる .

- マウスは本当に初心者でも使えるか？うそ！マウスの「しっぽ」が「頭」についているため、初心者は逆向きに持とうとする . GUI は未完成 .
- Black box 化の利点 . 初心者が内部構造を知らずに使える . システムの道具化 (§0.1) という意味で工学的に自然な方向 . 例：自動車は、変速機に始まって、オートマチック、電子制御、と、事故が起きると修復できない方向に製品が変化した .
- Black box 化の欠点 . 内部構造を知らなければ事故が起きたときに修復できない . 事故は必ず起きるから (公理 1) , これは不適切 . 理学と工学の最大の対立点 . 理学は、原理を知ることによって人がシステムの原因に対処できる (修復できる) ことを目指してきた<sup>2</sup> .

現在 . 方法の工夫 . text → GUI (graphical user interface) → virtual reality (?) . 計算機の resource (資源, 処理能力) の大きな部分を interface としての入出力処理にあてている . 未成熟 (目が不自由だと GUI は text base interface よりさらに難しい) .

### § システムとしてのコンピュータ

#### §0.1. システム

定義 1 . システムとは、複数種類の多数の素子が相互に影響しあって、ある機能 (目標を達成する活動) を実現すること . そのような仕組み<sup>3</sup> . 素子とはシステムの構成要素であって、入力に対する出力が定義されて

<sup>0</sup>本講義ノートで主に参考にした教科書は [7, 8, 9] .

<sup>1</sup>Optical character reader. 光学的文字読みとり装置 .

<sup>2</sup>哲弥流 . 拝金主義社会における理学の実質的役割である . 同時に機械を含めた環境に支配されないヒトの存在価値を示す理学のブライドのよりどころでもある .

<sup>3</sup>哲弥流 . 以下「哲弥流」とある項目については特に意見・議論を歓迎する (「哲弥流」という命名は、木田先生の添削により 19970225 に「我流」から変更したものである .)

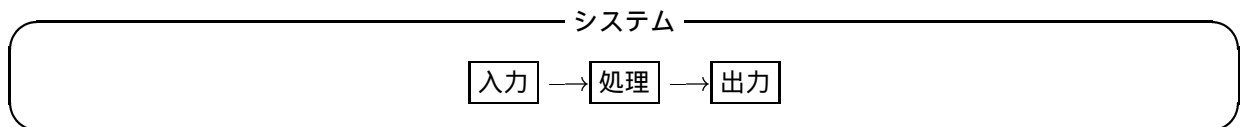
いる「もの」、その実現を知る必要がない対象（数学では関数にほぼ相当）。

- 注. (i) 機能（入出力の対応）と実体（素子が構成する内部構造）がなければシステムではない。特に、目標のない活動は機能ではないから、システムではない。
- (ii) 均質な働きをする素子が非常に多く含まれるときは無限個の極限で近似したほうが本質がつかめることがあるかもしれない（熱力学的極限）。数学的に難しいのは多数だが無限とは言えない、つまり、一つの素子が「見えてしまう」場合であろう<sup>4</sup>。
- (iii) 対象について述べる言葉というよりはむしろ、対象をどう見るかという視点に関する言葉。何を素子（最小単位）としてどこまでをシステム（系全体）と見るかは、一つの対象に一通りではない。

最小単位 素子の数が少ないときやシステムの内部構造を問題にしないときは、道具と呼ぶ。

系全体 自動車、ジャンボジェット機

- (a) 自分で運転する場合：外界と運転・操縦を入力として測度・方向を制御しながら推進力を出し、目的地へ移動・運搬を行うことを目標とする。この場合、運転しやすく事故を起こしにくい車を開発することがシステム設計の重要な課題。
- (b) 運転手を雇う場合：運転手に目的地を告げて、目的地へ移動・運搬を行わせる。運転手はシステムの制御機能と見ることになる。この場合、事故を防ぐためのコストと、事故時の予備の運転手や車の確保の費用の比較でシステムの安全性を設計する。<sup>5</sup>



事故. 目標と出力の差が装置設計時の想定を越えた状態、または、越えることを回避できなくなった状態。（外からの助けが必要。助からないかも知れない。）

本 ([7, p.4]) には

目標と出力が一致するときに、このシステムは完全なシステムとなる。

とあるが、私は、次を提案する。

公理 1. システムは必ず事故を起こす<sup>6</sup>。

注. 小さなシステムは事故がないようにみえる。しかし、これは全体で一つの道具と見ることができる場合で、実は事故がなければ道具の構成要素をシステムとして議論する必要がない！このように考えると、事故を起こすことがシステム（としてみなければならぬということ）の定義だと気づく。これが 公理 1 の意味である。

公理 1 をふまえると次に注目すべき点は、システムが大きくなると（事故が多くなるとは限らないが）事故が起きたときの被害は巨大になることである。残念ながら、私はそのような定量的法則を知らない。

コンピュータは大きさに比べると巨大なシステムである<sup>7</sup>。

定理 2. データのバックアップはかならず取ること。バックアップしないでよいのは遠くない時期に完全に失っても構わないデータだけである。

証明. 明らか。

□

<sup>4</sup> 哲弥流。

<sup>5</sup> 毎年同じ様な自動車事故が多数おき、また、飛行機が墜落すると必ず操縦士の責任という結論になるのを見ると、のりものは一般に運転手を取り替え可能な素子の一つとして含むシステムとして設計されていると思われる。

<sup>6</sup> 哲弥流。しかし、これは当然強調すべきことだと思う。このことからバックアップや保険がシステム論の重要な柱になるはずである。

<sup>7</sup> 素子が小さいから (§1)

データのバックアップ． 普段利用する場所とは別の場所にデータの複製を保管しておくこと．単に論理的に別の場所（例：別の directory）ではなく、ものとしての別の場所（例：ハードディスクとフロッピーディスク）に保管する．

注．定理 2 は 公理 1 の帰結だから永遠の真理．

他方、我々の子孫の時代には通用しないが、我々が気をつけないといけないこともある．コンピュータは実体が現れてから 50 年しかたっていない、発展期のシステム．まだ標準化されていない．自動車は講習を受けて免許を取れば誰でもどんな車でも運転できる（標準化）．内部構造を知らなくて良い、道具と見ることができ（エキスパートやマニア以外の一般ユーザーはシステムと思う必要がない）．

コンピュータは未成熟なのでまだ容易に事故を起こす（ハングアップ、ディスククラッシュ等）．一方で、一人で仕組みの全貌を知るのには難しいくらい巨大なシステム．計算機の説明がシステムという視点から始まるのはこのためであろう．巨大になったために、計算機を使う人全員がエキスパートになることは現実的でない．プロまたはエキスパートとしての仕事につくための全知識を大学で提供するのとは不可能．他方、そういう仕事に就かない限りは、危険が分かる程度に基礎を勉強して、安全で便利のところだけつまみ食いする使い方が利口に見える．

講義の目標． 発展途中の現状で企業は即戦力を求めるであろうが、使えるようになる訓練は使う場所で行うのが社会的効率を考えれば当然．しかし、計算機の発展に対応できる人材の育成という社会的要請を全否定することもできない．

大学の講義は、普遍的な部分に重点を置くのが自然．計算機は発展途上だが、それでも歴史的理<sup>8</sup>あるいは論理的理由などから（もはや）変えることは不可能と思われる部分もたくさんある．例えば、2 値論理、デジタル信号、半導体集積回路、文字や数字の 2 進数表現の規約、von Neumann 型（プログラム内蔵方式）、operating system の概念を含むソフトウェアの階層化、など．今後の進歩はこれらを用いつつ、これらの不自由な部分を覆い隠すという方向に進むしかあり得ない．

例えば今後急速に変化するものは、部品の高性能化（新素材を含む）、言語とプログラミング（アセンブラから高級言語まで含む）、アプリケーションソフトウェア、システム管理、man-machine interface（入出力装置を含む）、ネットワークシステム、であろう．これらの部分はこの講義では最小限にとどめるが、実用上（情報処理技術者試験受験を含めて）は、これらを知らなければ計算機を使えないので、必要に応じて各自が自習することを仮定する．

ハードウェアとソフトウェア． von Neumann 型計算機というシステムを、便宜上ハードウェアとソフトウェアに分ける．

hardware	もの	電子部品や機械部品を組み立てたもの
software	プログラム	制御・処理を指定するデータ（プログラム） プログラムの設計仕様書、操作手引き書

この分類の持つ意義については §5 参照．

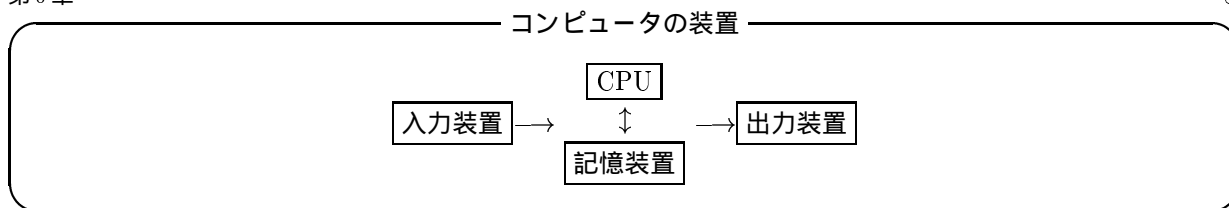
## §0.2. ハードウェア

以下、パソコン<sup>9</sup> を念頭において説明する．

<sup>8</sup>いまから異なる原理で計算機を作ろうとしても、既存の計算機と同程度に機能するものになるまでには投資が大きすぎて、しかも、既存の計算機を追い越すまでは利益を回収できないので、結局作れない．なお、この論法はアイザックアシモフの SF 「われはロボット」の中で、全てのロボットがロボット工学 3 原則に従って行動することを保証する論法として用いられている．

<sup>9</sup>Personal computer (PC). 和製英語だそうだが現在はアメリカでも personal computer で通用する．但し、略号 PC は politically correct という政治用語と誤解を招く．最近では日常会話では単に computer というパソコンのこと．一人 (single user) で使う計算機．本来、並行処理 (TSS-time sharing system) を考えず (single task), ネットワークによる利用やパスワードなどの個人情報保護も軽視 (stand alone) して、安くした計算機であったが、現在では interface の要求から multi task が復活し、web browser や電子メールの普及でネットワークも重くなりつつある．Apple 社の PC がこの方向の草分け．Windows95 のうたい文句もネットワーク．この変化に伴って個人情報保護も再び問題になるが、Windows95 ではネットワークもパスワード管理も未熟に思える．

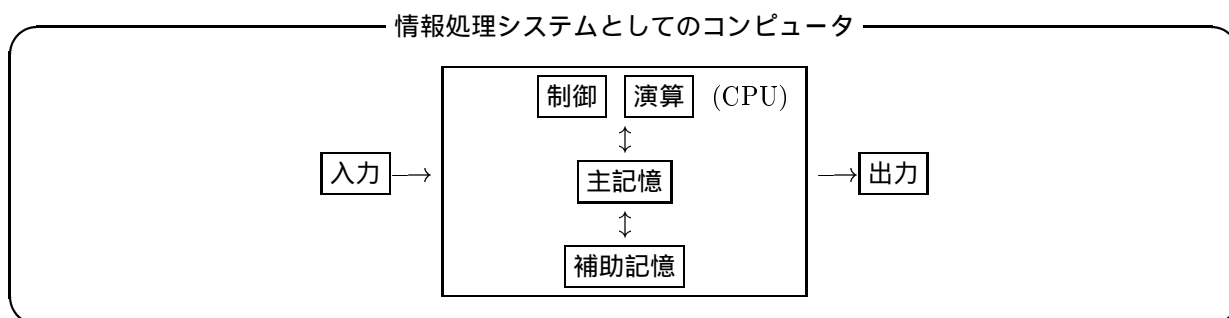
Program counter のことも PC と略記するので、混乱しないように注意．



CPU(central processing unit, 中央処理装置)． システムとしてのいくつかの機能(制御装置・演算装置)が一つの microchip (半導体性の黒い胴体と金属性の光沢のある足でできたげじげじ)に収まっている<sup>10</sup> (§3.3)．

記憶装置． 処理装置が直接内容を処理できる主記憶装置と、主記憶へまたは主記憶からのコピーしかできない補助記憶装置に大別される．前者はアクセス (access: ここでは読み書きすること) の速さを重視(半導体メモリ)．後者は大量に長時間データを保存する場所(主に磁気メモリ) (§3)．

主記憶装置は一定の記憶単位で番地(数字)がつけられていて、制御装置は数字が与えられると、その数字の番地の内容の制御・処理装置への読み出しや、処理装置の内容のその番地への書き込みを行う機能を持つので、データの内容を直接処理できる．補助記憶の内容はいったん主記憶にコピーしてから処理する．主記憶は容量が小さい上に通常計算機の電源を切ると内容が消えるので、補助記憶にコピーして長期保存する．



演算． 情報処理は数学的には計算とすることができる．

ALU: arithmetic and logic unit (算術論理演算装置)．データの算術演算, 比較, 一時的保持, 主記憶からの読み出しと主記憶への書き込み．

制御． 状況に応じて処理の内容を(システムが自発的に)変更すること．例えば feed back は出力と目標の差を検出して制御を行うこと．出力と目標を完全に一致させるのは不可能だから, 一般に制御が必要である(外界が複雑過ぎて計算しきれない. 外界の変動が予測できない.) 事故を詳しく定義し直すと, 内的な制御によっては出力を目標に近づけることが不可能になった状態．コンピュータは von Neumann 型と呼ばれる特徴的な制御方式を採る．

### §0.3. データとしてのプログラム

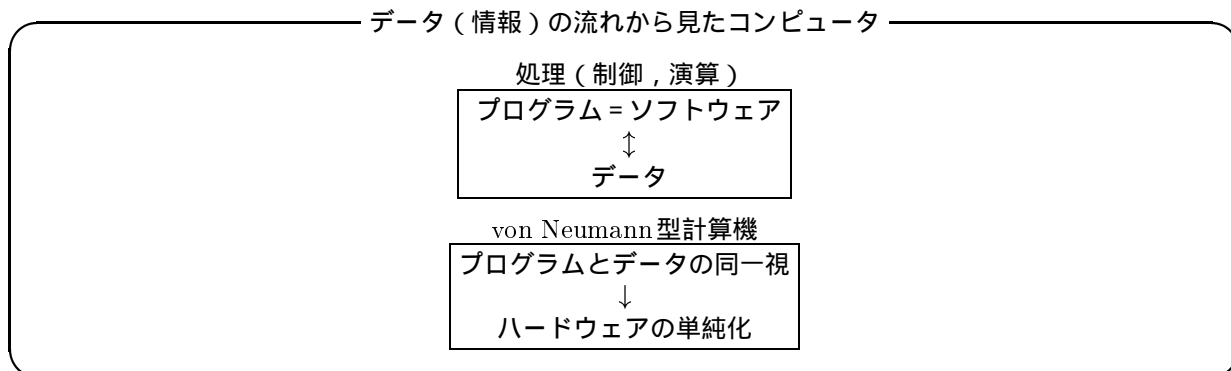
von Neumann 型コンピュータ． 制御や処理の手順をデータとして記憶装置に蓄え, これを一つずつ命令として読み出すことで種々の処理を実行できる(プログラム内蔵方式)．制御や処理の手順をデータとして記憶装置に蓄えたものをプログラムと呼ぶ．現在のコンピュータは全て von Neumann 型コンピュータである．

von Neumann 型コンピュータにおける制御とは, 主記憶においたプログラムを逐次 CPU に読み込んで命令として解釈して, 処理の実行のために各装置を動かす電気信号を送り出すことである (§2.1)．

利点: ハードウェアの機能を単純なものに限定しておける．具体的な仕事はプログラムを工夫することで行える．即ち, 使うときにプログラムを load することで, 一つの機械に複数種類の複雑な情報処理を行わせることができる．

<sup>10</sup>時代によって CPU に納められている機能は変化している．50年前は CPU はなかった．一時補助演算装置 (coprocessor) が演算機能の一部を別の chip で受け持っていた．現在の CPU は主記憶の一部(のコピー)を持っている．

記憶容量と時間さえ十分にあれば原理的にどんな計算も実行できる．特に，ハードウェアや基本的な命令体系が異なっている別のコンピュータのまねをするようにプログラムすることができる (simulator) . (万能 Turing machine の理論 .)

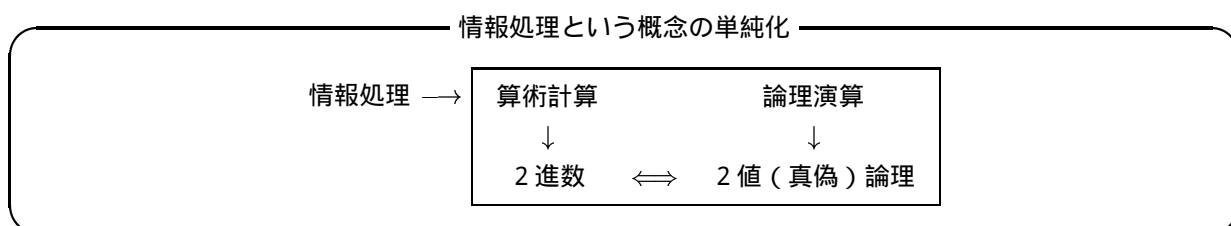


CPU の機能 . von Neumann 型コンピュータの CPU に要求される機能 .

- データを命令として解釈できる (decoder) .
- 最小限の演算 (原理的には足し算と補数に帰着) (ALU) .
- 特定のデータに注目する (register) .
- データを順番に見ていく (program counter) . 注目するデータの値によって次に見るデータの番地を変更する .

この程度あれば十分 (§2)<sup>11</sup> . 電源を入れると IPL (initial program loader) が補助記憶から主記憶に最初のプログラムをコピー (load) して, 制御と処理が始る . あとは芋蔓式に実行中のプログラムが次に行う処理の書かれているプログラムを呼び出して制御を引き渡す .

2進数 . 現在のコンピュータは, 最下部で (電気回路として) は, 全てのデータを 0 と 1 だけで, 即ち 2進数として表現し, 1 を真, 0 を偽, とみなして全ての演算及び制御を論理演算に翻訳して処理する .



- 本質的に全ての情報処理は算術計算 (即ち論理演算) と思える . Well-defined な機能ならば表現できないものはない<sup>12</sup> .
- (電気回路の) 部品の機能を処理に無関係に共通に取れる . 部品の開発や組み合わせに (アナログ素子のような) 職人芸はいらない . 2進数は 0 と 1 で表現するので, 回路素子の観点からは最も単純 . 電圧の高低, 磁化のあるない .
- 処理によっては効率が悪い (記憶容量または処理速度の意味で) 場合がある .

<sup>11</sup>十分条件は必要条件ではない! 1つの chip の中にこれらが納められて大量生産で売り出されている今日, という歴史的状況を見ると, かなりの程度この部分は決まっいて, それを一つの素子と見て, それを多数組み合わせでより複雑なシステムを作る, という方向に世の中は進みそうである . しかし, 全く違う構成で同程度以上の機能を持つ新しい型のコンピュータを作れないということではない . 例えば interface, 特に画像の出力のように非再現性や曖昧さが許される部分では program counter による命令の逐次翻訳ではなく, 画素を並列的に処理する補助 processor を考案してもよいはずである .

<sup>12</sup>哲弥流 . 真偽だけを扱う 2値論理の範囲は問題ない . 数学で扱う「任意の」や「存在する」という論理述語は, 有限なコンピュータでは問題になるが, 実際には有限集合しか扱わないことで原理的な問題を逃れる . 実際上の問題 (連続量は離散近似, 並列処理は整理して逐次処理化, など) がこの原理的な問題と関連するかどうかの考察は, この講義ノート作成時には時間も資料も足りない .

利点が圧倒的（汎用性と大量生産）なので、現在のコンピュータは2進数で全ての情報を表現する。

さらに、2進数の算術計算は論理演算とすることができるので、コンピュータを実現する電気回路は論理回路として理解する §1.2。

デジタル信号。全てを0と1で内部表現するという事は、特に、データを全て離散量（自然数）で表現することを意味する（cf. アナログ信号）。曖昧さが無い（well-defined）、回路の種類を限ることができる（単純化）、拡張が容易、など、利点が多いが、デジタル信号は人間がじかに見ると分かりにくい（針のついた時計を好む人は今でも多い）。

bit. 2進数一桁の情報、最小の情報量。ひいては2進数表示の桁や桁数のこと。1 Byte = 8 bit. 1 KB = 1024 B. 1 MB = 1024 KB. 1 GB = 1024 MB.

補助単位

K: kilo, M: mega, G: giga. キロは千 ( $10^3$ ), メガは百万 ( $10^6$ ), ギガは十億 ( $10^9$ ) を表す。  
m: mili,  $\mu$ : micro, n: nano, p: pico. ミリは  $10^{-3}$ , マイクロは  $10^{-6}$ , ナノは  $10^{-9}$ , ピコは  $10^{-12}$ 。

注. (i) 記憶容量などの情報の量を表すときは  $K = 2^{10}$ ,  $M = 2^{20}$ ,  $G = 2^{30}$ . 計算するときは  $2^{10} = 10^3$  と思って十分なので、最初は違いをあまり心配しなくてよい。

(ii) 英語の数の数え方は千倍で次の呼び方に移る。One thousand, ten thousand, a hundred thousand, one million. 一万, 十万, 百万, 一千万, 一億。ので, K, M, G, m,  $\mu$ , n, p を覚えなさいといけない。

数 実数, 整数それぞれについて2進数への変換規則が決まっている。実数は有効精度が決まっている（プログラムを作るとき誤差が蓄積しないように注意をする必要がある）。整数は表現できる最大の数が決まっている。それより大きな整数を扱いたい（多重精度）ときはプログラムを作る ([5])。

文字 英数字 1B/1字 (ascii code), 和字 2B/1字 (JIS 第1,2水準文字), の2進数 - 文字変換規則が定められている。和字については複数種類の規則があるので、異なる計算機でメールなどの通信を行うと文字化けすることがある。

## §0.4. ソフトウェア

von Neumann 型コンピュータの情報処理能力はハードウェアの性能と、その性能を効率よく利用できるプログラムの開発の両方に依存する。ハードウェアと相補的な関係にあるものとしてプログラムを見たときに、これをソフトウェアと呼ぶ<sup>13</sup>。ハードウェアとソフトウェアの関係とはゲーム機とゲームソフトのことである<sup>14</sup>。

(von Neumann 型) 計算機

ソフトウェア (プログラム)  
ハードウェア (もの)

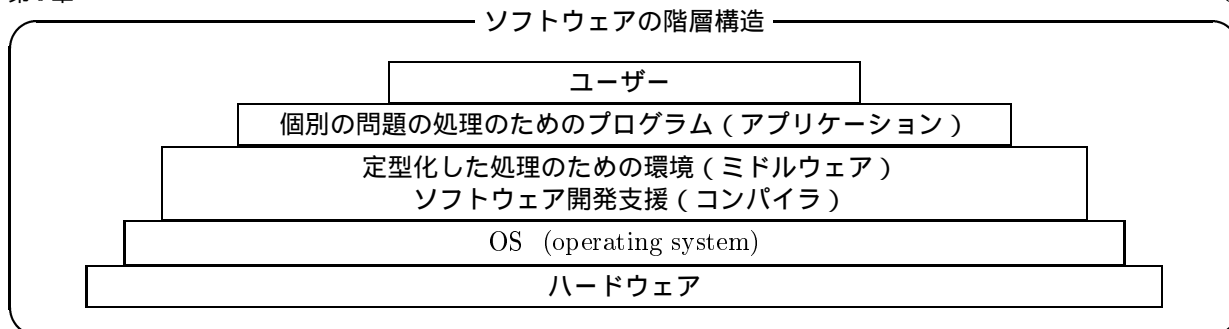
ソフトウェアまで込めるとシステムとして開放系になっている。

昔は基礎的なところまでソフトを自作していたが、今日ではユーザーに近いレベルまで購入あるいはシステムの外注が普通になっている。それでも、標準化が完了していないため、コンピュータに詳しい人がいないと動かない。

コンピュータは万能であるが故に、複数の言語やソフトが同じ問題を解決しうる。自由主義市場では結果としてソフトウェア間の互換性が失われる恐れがあるが、さもないと事実上の独占が生じる恐れがある。

<sup>13</sup> 哲弥流。

<sup>14</sup> 哲弥流。いわゆるゲーム機は、ユーザーに公開された機能が極端に制限されていることを除けば、計算機そのものである。



ソフトウェアの階層化はまだ進行中で、まだ確立した分類はない。この図は我流である。

何故ソフトウェアに階層構造があるか？ コンピュータは原理的に（ハードウェアが十分大きければ）どんな情報処理もプログラムの取り替えによって行える。しかし、ハードウェアは徹底的に単純化したので、どんな処理を行うのも非常に複雑なプログラムを書かざるを得ない（単純な機械で単純な選択肢しかなければ、わずかな種類の処理しかできないはず。例：他社路線の連絡切符も買える切符販売機は慣れていないと使いにくい。）そこで、機能を制限することで人間がしなければならない指示を減らしたのが市販のソフトウェアである。計算機が高性能化して人の期待も膨らむにつれてソフトウェアの階層化は進行する。人間にとっての使いやすさ (interface) という、システム論の立場からの要求のもとでは、ソフトウェアの階層構造は不可避である。

市販のソフトウェアが中心になることで、「何でもできる」というコンピュータの最大の特徴・魅力は、一般ユーザーから取り上げられて、コンピュータ会社（と少数のエキスパート）だけのものになりつつある。

OS (operating system) . (特に CPU 以外のハードウェア資源を有効に利用することで) 処理効率や interface を改善する目的のために、恒常的に動作させるプログラム。個々の情報処理は、OS の上で動くプログラムをユーザーやソフトウェア会社を書くことで行う。これらのアプリケーションプログラムは OS のデータとして読み込まれ、OS が解釈して適切な命令 (の集まり) に翻訳する。結果として OS は本来の CPU と見かけ上異なる環境を提供する。OS が資源の有効利用を計算するので、プログラマーはハードウェア資源の有効利用を具体的に考えずにプログラムを書くことができるようになる<sup>15</sup>。

ハード資源が大きくなるほど効率よく利用するための OS が難しくなる。可能性が大きくなると、人間の要求も多くなり、interface としての OS も難しくなる。ハードウェアの進歩が緩やかにならない限り、OS も開発を続けざるをえない。

コンパイラ。 機械語でプログラムを作るのは大変。

高水準言語 プログラム作成を支援するプログラムの仕様。

コンパイラ 高水準言語で書かれたテキストファイルを機械語に変換するソフトウェア。

コンパイラをハードウェアの機種毎に作っておけば高水準言語は共通に用いることができる<sup>16</sup>。

アプリケーションソフトウェア。 個々の問題は各自でプログラムを作って処理を計算機で行う、というのが元の考え方だった。これを容易にするプログラム作成支援は OS や compiler の重要な目的であった。

ハードウェアの性能向上・複雑化、計算機に期待する問題の大規模化・複雑化、に伴って、OS や compiler を用いても個々のユーザーがプログラムを作成するのは手間になってきた。今日では定型化した作業はある程度汎用性のあるプログラムをつくって販売する (§5)。

<sup>15</sup> 哲弥流。

<sup>16</sup> 実際は高水準言語に方言があって、完全な共通性は損なわれる。ハードウェアの違いによっては全く同じ機能を再現できない場合がある。ある言語で書かれたプログラムが違うハードウェアでもほとんど修正無しに動くとき、移植性がよいということがある。C 言語で書かれた UNIX という operating system が広まった理由の一つは移植性の良さであったという。



## 第 章 ハードウェア

### § 論理回路

電子回路で情報処理（特に整数の算術演算）が可能なこと（即ちコンピュータの存在証明の概要）を示す。

#### §1.1. 2進法

現在のコンピュータは、最下部（部品のレベル）では2進数で全てのデータと情報処理を表現 §0.3.

2進法による自然数の表現.

$$\text{十進法: } 31015 = 3 \times 10000 + 1000 + 10 + 5 = 3 \times 10^4 + 10^3 + 10 + 5$$

$$\text{二進法: } 10111 = 2^4 + 2^2 + 2 + 1 = 16 + 4 + 2 + 1 (= 23 = 2 \times 10 + 3)$$

この表現法は何進法か言わないと意味がない！10111 は10進法だと一万百十一，2進法の10111 は二十三．31015 は5（以下）進法ではあり得ないが，それ以外は何か分からない．計算用紙やメモには10111 (2) などとしておいたほうがよいかも<sup>17</sup>．

—————  $p$ 進法による表現 —————

- (i) 0 以上  $p - 1$  以下の整数だけを記号として用いる，
- (ii) 記号を複数個並べると左の記号は右隣と同じ記号の  $p$  倍（右の記号は左隣の  $1/p$  倍）を表す，
- (iii) 有限個の記号が並んだ記号列全体は各記号の表す数の和を表す，  
という約束で任意の自然数を（一通りに）表現する方法．

桁数 並んでいる数字の個数．10111 は5桁の数．2進法  $n$  桁で表せるのは0から  $2^n - 1$  の  $2^n$  個の数<sup>18</sup>．

桁目 並びの中のある特定の数字の位置．自然数12345の4桁目は2，1桁目は5．

ビット 2進法のとときは桁のことをビットとも呼ぶ．

基数  $p$ 進法の  $p$  のこと．

注. (i)  $p$ 進法による非負実数の表現．

(a) 小数点の左となりの1が自然数の最小単位，

(b) 小数点を越えて右に無限に続く記号列は各記号の表す数の級数の和を表す，

という約束を追加すると任意の非負実数を（一通りではないが）表現できる．

通常，コンピュータが指定する実数の表現は  $p$ 進法実数そのままではない (§11.1)．

- (ii) ちなみに，文字データも記憶や処理の時点では2進整数で表現されている．入出力のときに変換するが，変換方法は国際的な約束があり，約束に従った変換機能がハードウェアに備わっている (§12.1)．

基数の変換： $p$ 進法から10進法はかけ算（既出）．逆はわり算．

<sup>17</sup> 哲弥流．2進数を  $b10111$ ，16進数を  $FFH$ ，10進数を  $d93$ ，などとしたほうがよく用いられる表記に近い（前につける流儀と後ろにつける流儀があって，さらにつける文字も流儀によるようなので，結局は我流にならざるを得ない？）．

<sup>18</sup> 試験では  $n$  桁で表せる，という言い方のときは  $n$  桁以下の意味に使い，ちょうど  $n$  桁のときは何か修飾語句がつくことが多い．ある数が2進法で何桁になるか，という言い方のときはちょうど  $n$  桁になるときに  $n$  桁と呼ぶ．

23 (10) :  
 $23 \div 2 = 11$  あまり 1  
 $11 \div 2 = 5$  あまり 1  
 $5 \div 2 = 2$  あまり 1  
 $2 \div 2 = 1$  あまり 0  
 $(1 \div 2 = 0$  あまり 1).

あまりのところを下から読むと  $23 (10) = 10111 (2)$  .

16進法 . 2進法だと数字列が長くなる . 人間には読みにくい ( 計算機の出力異常の原因を調べる (debug) ときに , 記憶装置に蓄えられている数字や記憶の番地を人間が調べる必要がある . ) 10進法に直すと内部表現との対応が分かりにくい . 16進法だと , 2進法四桁毎に一つの数に置き換えるだけなので , 数字列が短い上に , 内部表現も分かりやすい . 所以 debug に便利 .

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

$FF (16) = 255 (10) = 11111111 (2)$  .

記憶容量の単位 1Byte = 8bit = 2進数8桁 = 16進数2桁 = 00 ~ FF.

補数 . 自然数の四則演算は足し算と桁移動と補数に帰着する . そこでまず補数を説明する .

負の整数の表現方法 . 内部表現は全て 0 と 1 だからマイナスという記号は最初はない !

$n > 0$  のとき ,  $n$  より大きい  $2^M$  を考えて ,  $-n \mapsto 2^M - n$  と置き換えてみる (  $M$  は  $n$  の桁数より大きくとる . )

- $2^M - n > 0$  ( 即ち , 自然数だからそのまま2進数で表現できる ) .
- $m - n = m + (2^M - n) - 2^M$  (  $n$  を引くということは ,  $2^M - n$  を足して  $M$  桁目を取り除くことに等しいから ,  $2^M - n$  は  $-n$  と同じ役割を果たせる ) .

#### 2の補数

$M$  を  $n$  の ( 2進法表現での ) 桁数より大きくとるとき ,  $2^M - n$  を  $n$  の2の補数と呼ぶ .

$29 (10) = 11101 \mapsto 100000 - 11101 = 32 - 29 (10) = 3 (10) = 11 (2)$  .

計算の仕方 : 2進法表現の各ビット ( 桁 ) の 0 と 1 を入れ換えて 1 を足す .

$11101 \mapsto 00010 \mapsto 00011$  .

( コンピュータの内部で , 枠内のように 10進数に直して補数を計算するのはばかげている . )

$n$  の2の補数を  $-n$  の表現とする . これだと , 例えば  $-29 (10) = 11 (2)$  と  $3 = 11 (2)$  が区別できない , といったことが起きる (  $M$  の値によって区別できない数の組み合わせは変わるが , 区別できないことは変わらない . ) そこで , 自然数の2進法表現に一桁特別の桁 ( 符号ビット ) を追加して , 正の数はそのビットを 0 に , 負の数 ( 補数 ) はそのビットを 1 にする .

4B (32bit) 整数 ( 2進法 32桁 ) の場合 . 先頭ビット ( 桁 ) を符号ビットとし , 残り 31桁で2進法 31桁までの自然数を表す (  $0 \sim 2^{31} - 1$  ) . 自然数は符号ビット 0 , 負の数は絶対値の2の補数 (  $M = 32$  ) .  $1 \sim 2^{31}$  の補数の先頭 bit は 1 になるから , 正の数との混同は起きない .

$29 (10) = 000 \cdots 0011101$      $-29 (10) = 111 \cdots 1100011$  .

2の補数の求め方 . 次のどの方法でも2の補数になる .

- 2進法表現の各ビット ( 桁 ) の 0 と 1 を入れ換えて 1 を足す ( 既出 ) .
- $2^M = 1000 \cdots 00 (2)$  から元の数の2進法表現を引く .
- 右の桁から見て初めて出た 1 より左の桁の 0 と 1 を全て入れ換える .

問． 以上の方法が全て2の補数を与えることを示せ．

何故単に符号ビットを1にするだけで負の数と約束しないか？ 足し算（従って四則演算全て）が統一的に行える．

単に符号ビットを1にするだけで負の数と約束すると，負の数を足すときと正の数を足すときで全然計算規則が変わってしまう．2の補数にしておけば，符号ビットだけ気をつければよい．

2進法による四則演算．

- 足し算の計算方法は明らか（半加算と桁上がり）．
- 2倍は桁を一つ増やして，一桁目を0とする（桁移動 (shift)）． $11 \times 10 = 110$  ( $3 \times 2 = 6$  (10))（ $n$ 倍は桁の増加と足し算に帰着できることを確認せよ！）
- かけ算は $2^n$ 倍と足し算に帰着する．
- 割り算は引き算（と桁移動）の繰り返しに帰着できる．

$$\begin{aligned} 10000 - 1100 &= 100 && (1100 = 11 \times 10^2) \\ 100 - 11 &= 1 \end{aligned}$$

だから， $10000 \div 11 = 101$  あまり 1．

特に，2による割り算は桁を一つ減らして，一桁目の数字があまり．

- 引き算は引く数の符号を逆にして（2の補数をとって）足せばよい．

足し算と補数と桁移動が処理できれば任意の計算が処理できる．

実際の計算機では計算の効率上，このような徹底した単純化は行わずに，頻出の演算は別の回路に任せることが多い．素子の進歩や計算機の大きさや目的によって具体化は異なる．

問． 2の補数と符号ビットで負の数を表しておけば，足し算 $n + m$ は $n, m$ の正負で場合分けをすることなく，機械的に計算できることを確認せよ（但し，桁数が考えている最大桁を超える (overflow) 計算はしないこととする）．符号ビットのところはどのような約束で計算すればよいか？最大桁を3くらいに小さく取って調べよ．負の数を補数で表さず，単に符号ビットの変更だけで表したときはどうか？

## §1.2. 論理演算と論理回路

論理定数 真 ( $T, 1$ )，偽 ( $F, 0$ )<sup>19</sup>．

論理変数  $A \in \{0, 1\}$

論理演算子 和 (OR,  $\vee, +$ )，積 (AND,  $\wedge, \times$ )，否定 (NOT,  $\neg$ )，排他的論理和 (EOR (exclusive or),  $\oplus$ )

2項演算子 ( $\{0, 1\} \times \{0, 1\} \rightarrow \{0, 1\}$ ): OR, AND, EOR

単項演算子 ( $\{0, 1\} \rightarrow \{0, 1\}$ ): NOT

### 論理演算子の定義

$$\begin{aligned} \neg 0 &= 1, \quad \neg 1 = 0, \\ A \wedge B &= 1 \text{ if and only if } A = B = 1, \\ A \vee B &= \neg(\neg A \wedge \neg B) = 0 \text{ if and only if } A = B = 0, \\ A \oplus B &= (A \vee B) \wedge \neg(A \wedge B) = 1 \text{ if and only if } \neg A = B. \end{aligned}$$

<sup>19</sup>以下，0, 1を記号として主に採用．数なのか論理定数なのかは用いる演算子で区別する．

NOT と AND だけ定義しておけば，他の演算子はこれらを用いて表せることに注意．

更に， $A \text{ NAND } B = \neg(A \wedge B)$  を定義しておくとし， $\neg A = A \text{ NAND } A$ ， $A \wedge B = \neg(A \text{ NAND } B)$  とする．

全ての論理演算子が NAND だけで表現できる．

論理素子（即ち回路素子）を徹底的に単純化したいときは便利．回路構成上 NOT を入れた方が（トランジスタによる信号増幅が入るので）よい．

演算子の優先順位：かっこ，算術演算，NOT，AND，OR，EOR<sup>20</sup>

公式．

$$A \vee A = A \wedge A = \neg(\neg A) = \boxed{(1)}, \quad A \vee B = B \vee A,$$

$$A \wedge (A \vee B) = \boxed{(1)},$$

$$A \wedge (B \vee C) = (A \wedge B) \boxed{(2)} (A \wedge C) \text{ 分配法則},$$

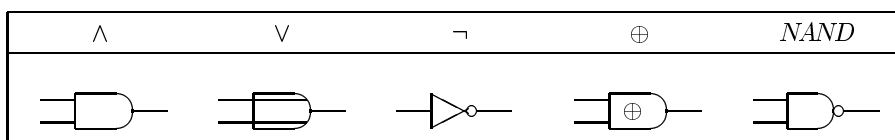
$$\neg(A \wedge B) = (\neg A) \boxed{(2)} (\neg B) \text{ de Morgan の法則}.$$


以上の公式で  $\vee$  と  $\wedge$  を完全に入れ換えた公式も全て成り立つ．いろいろな等式（公式を含む）の証明は

- (i)  $(A, B)$  の値全て  $((0, 0), (1, 0), (0, 1), (1, 1))$  に対して右辺と左辺の値（どの場合も 0 か 1 かどちらか）が一致することを見る（特に，変数が少なく式が簡単なとき．また，コンピュータに調べさせるときはこの方法が簡単）．表にすると見やすい．
- (ii) 既に証明した式を用いて変形して両辺の一致を見る（特に，変数が多い場合や複雑な式の場合）．
- (iii) 論理変数を集合に対応させてベン図，中間的な表現としてベッチ・カルノー図，などを利用すると分かりやすい場合もある<sup>21</sup>．集合算の観点からは EOR 以外の演算子が基本的ということになるが，排他的論理和は日常言語の「または，どちらか」に近いのでなじみやすい．また，半加算は EOR である．

問． 上記公式の空欄 (1), (2) を埋めよ．

論理回路． コンピュータに情報処理をさせるためには，論理演算を電気回路で表現しなければならない．



左に伸びている線が 0 または 1 の状態のとき，右に伸びている線の状態が論理演算子の値になっている，という記号．他は慣用の記号だが， だけの記号．

問． NAND 回路だけを用いて次の回路を作れ．

- (i) NOT 回路．
- (ii) EOR 回路．

### §1.3. Clock — 論理回路への時間の導入

計算機は複数の（多くの）制御や演算を行うから，多数の論理変数や自然数を用いる．電源を入れてから切るまでに入れる入力変数の数だけ配線を用意するとたいへんなことになる．もちろん実用的ではない．実際には少数個の入力端子に順番に変数を入れていく．つまり，変数  $A$  を時間の関数  $A(t)$  とすることで膨大な変数を処理する．時間を論理回路に導入するために clock という概念がある．Clock による論理回路の動作は Flip-Flop 回路によって実現される．この節で以上について説明する．

<sup>20</sup> 2 項演算子が複数あるときは，かっこを必ずつけるほうが良いと思う．

<sup>21</sup> 省略．

ビットパターン．  $0, 1$  の列をビットパターンと呼ぶ．

- 論理変数の列は  $T \rightarrow 1, F \rightarrow 0$  の対応でビットパターンと思える．
- 自然数は2進数で表して四則演算を忘れればビットパターンと思える．

ビットパターンの演算は，各桁毎の論理演算で定義する．

例．  $101 + 011 = 101 \vee 011 = 111, 101 \times 011 = 101 \wedge 011 = 001.$

$+, \times$  が，自然数の四則演算の記号とビットパターンの論理演算の記号と同じなのに意味が違う．注意すること．

四則演算は足し算と桁移動と補数で表せる §1.1．これらの演算をビットパターンに対する論理演算で表せれば，自然数としての四則演算も論理回路で表せることになるので，全ての情報処理を論理回路で表せる<sup>22</sup>．

この節の以下で，任意の桁の自然数同士の足し算を行う論理回路（全加算器）を例にとって，自然数の演算が clock 付きの論理回路で表せることを示す．

**Clock.** 一定の時間毎に CPU の状態を変化させていくことが CPU での処理の実行になるように回路を組んである．電気パルス信号（0 と 1 を周期的にくり返す信号）によってによって実現．CPU のどの部分（回路素子）も一斉に状態を変化させるという意味で同期をとる，という．

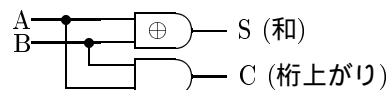
単位時間にどれだけ pulse があるか（周波数）が速さの目安．現在のパソコン（pentium CPU）で 100MHz（十億 clock/秒）の何倍，といった程度．パソコンの CPU の性能に 100MHz などとあるのは，パルスの周期が  $10^{-8}$  秒であることを表す．Clock は周期的なパルス信号のことだから，電子時計のように，水晶発振器とトランジスタ（またはこれらと等価な半導体素子）で作れるのはほぼ明らか（省略）．

問． 最新鋭の大型計算機を随時導入する機関としては，学術目的と軍事目的以外では気象庁が有名である<sup>23</sup>．ある記事によると 1996年3月に気象庁が導入したスーパーコンピュータ<sup>24</sup>は1秒間に約3億回の命令を実行する．一つの命令に clock 1周期かかり，CPU は1つしかないとして，1 clock に光（真空中で 30万 km/sec）がどれだけの距離進めるか計算せよ．（真空中の光の速さより速い情報伝達手段はないので，clock で同期を取るためには，CPU をこの大きさより十分小さくしないといけない．現在の「げじげじ」は一辺 5mm 程度の中にびっしり回路が組み込まれている．計算と比べると，clock の高速化は限界に近づいていることが分かる<sup>25</sup>．）

時間の関数としてのビットパターン． ビットパターン  $A$  の最後の桁から順番に時刻  $t$  を対応させることで  $A$  を時刻  $N$  から  $\{0, 1\}$  への関数  $A: N \rightarrow \{0, 1\}$  とすることができる．例：  $A = 1101$  のとき  $A(1) = 1, A(2) = 0, A(3) = 1, A(4) = 1$ ．時刻は CPU の clock の一周期が単位だから，例えば 100MHz の CPU ならば  $10^{-6} \text{s} = 1\mu\text{s}$  が  $t$  の単位．以下では簡単のため時刻の単位を取り直して 1 を単位として説明する．

桁（ビットパターン）を時刻（clock 単位）で表現する．

半加算器． 1ビットの自然数の足し算を行う論理回路．一桁なので clock は必要ない．



<sup>22</sup>もちろん，省略した話がいっぱいあるから，これで計算機が存在が数学的に証明終わり，というわけにはいかないが，四則演算を論理演算で表せることが本質的な一歩であることは推察できると思う．

<sup>23</sup>気象の数値予報，即ち，流体力学の物理法則（偏微分方程式）を解くことは，コンピュータの創造者の一人 von Neumann も，計算機の意義の一つ（Navier-Stokes 方程式の数値解析）としてあげていたはずである．

<sup>24</sup>厳密な定義があるかどうか知らないが，計算向きに高速性を重視して設計された最新鋭のコンピュータの通称として通常用いられる用語．

<sup>25</sup>本当は一つの命令には複数周期かかるし，現在のスーパーコンピュータは複数個の CPU が並列に命令を処理している．だから，間の仮定自体は非現実的だがプラスマイナス考えると，結論はあまり変わるとは思えない．即ち，極微加工技術の極端な進歩が，いままで用いなかった物理原理による素子か，CPU の数を増やす並列計算機か，今後のスーパーコンピュータは clock のスピードを上げるにはいずれかの方向を必要とする．

注. 黒い丸は導線をつないでいることを表し、丸がない交差は導線が立体交差している（つながっていない）ことを表す。

全加算器. 時刻が桁を表すことにする。即ち、 $t = 1, 2, 3, \dots$  に対して時刻  $t$  における端子  $A, B$  の値  $A(t), B(t)$  がそれぞれ入力データ  $a, b$  の  $t$  桁目を表す。

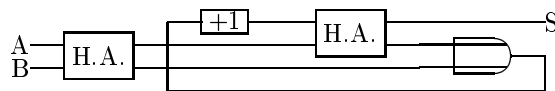
桁上がりは時刻の遅れによって表現できる。出力が入力の 1 clock 昔の値になる回路（遅延回路）をここ

だけの記号で  $\boxed{+1}$  で表すことにする。 $\boxed{+1}$  の左右の端子の時刻  $t$  での値をそれぞれ  $I(t), O(t)$  と書くと、 $O(t) = I(t-1)$  となる。

データの和  $s = a + b$  の  $t$  桁目が端子  $S$  の時刻  $t$  での値  $S(t)$  になっている回路を全加算器と呼ぶ。

何桁の数の加算でも、回路もプログラムも変える必要はない<sup>26</sup>

半加算器を  $\boxed{\text{H.A.}}$  と書くことにすると、次の回路は全加算器になる。

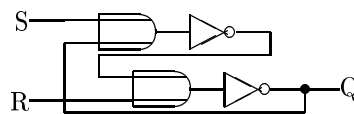


問.

- (i) 全加算器の回路が正しく加算を行うことを確かめよ。
- (ii) 論理回路（全加算器も一つの箱で表してよい）を用いて 2 の補数を計算する回路を作れ（入力として何を用意するかも明示すること。）このことと今までに述べたことから clock と NAND 回路と遅延回路があれば、任意の整数に関する演算が論理回路で表現できることになる。

Flip-Flop 回路. 時刻で桁を表すためには遅延回路  $\boxed{+1}$  が必要である。これは 1 clock の間データを保持すること、即ち、動的な記憶回路を作ることに帰着する。これを実現する回路として Flip-Flop 回路が知られている<sup>27</sup>。

RS (reset-set) Flip-Flop 回路. 単に Flip-Flop 回路といえは次の回路を指すほど有名な回路である。



但し、二つの NOT の出力は初期状態として、(0, 0) でも (1, 1) でもない、とし、また、入力 (S, R) はどの時刻でも (1, 1) とはならないと仮定する（このとき、出力は任意の時刻で (0, 0) にも (1, 1) にもならない。）

$S(\text{set}) = 1$  かつ  $R(\text{reset}) = 0$  ならば  $Q = 1$ ,  $(S, R) = (0, 1)$  ならば  $Q = 0$ . 特徴は  $(S, R) = (0, 0)$  のとき、 $Q$  の値はそれまでの値がそのまま残る。

Flip-Flop 回路は記憶素子の機能を持つ (§3.1) (同じ入力でも過去の状態によって出力が違う。)

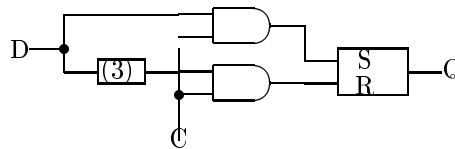
<sup>26</sup>one-chip microprocessor 以前の電卓は本質的に桁数の分だけ半加算器を並べていた。電卓は今日商業的には日本が世界制覇しているそうだが、時間を桁とみなす発想は U S A で開発されたものを日本人が勉強してきた技術であるとの主旨の NHK の「電子立国日本の自叙伝」での証言があった。今日の CPU でも、半加算器を複数つないで何桁が同時に処理すれば早いはずだが、回路を大きくすると信号伝達の遅れが生じるので clock speed に影響する。回路の集積度との兼ね合いで回路を決める必要がある。

<sup>27</sup>もっと単純な方法として、長い導線に信号を流す、という方法がある。光も有限の速度でしか進まないで、遅れを得る。先の間でみたように、長いといってもそんなに長くする必要はないので、これはありうる。但し、欠点としては、エネルギー損失による情報伝達率低下、正確な同期の困難、材質の周波数特性による波形の乱れ、などが考えられると思う。特に、clock が時間的に正確に周期的でないといこの方法は全く使えない。

問 .

- (i) Flip-Flop 回路の前までに出てきた回路は , 各入力に対して出力が一意的に決まっていたことを確認せよ .
- (ii) RS Flip-Flop 回路で  $(S, R) = (1, 1)$  のときどうなるか ?

**D (delay) Flip-Flop 回路 .** RS Flip-Flop 回路を (これも , ここだけの記号で)  $\boxed{\begin{smallmatrix} S \\ R \end{smallmatrix}}$  と書く . RS Flip-Flop 回路は状態の記憶という本質を持っている . これを用いて構成した次の D Flip-Flop 回路は  $C$  (clock) 入力端子が 1 になったときだけ  $D$  端子の値がそのまま出力  $Q$  となり ,  $C = 0$  のときはそれまでの出力の値を維持する .

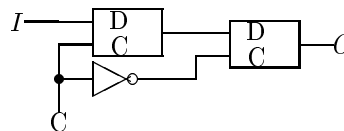


問 . 上の  $\boxed{(3)}$  に適切な論理素子を入れて D Flip-Flop 回路を完成せよ .

**遅延回路 .** D Flip-Flop 回路を (ここだけの記号で)  $\boxed{\begin{smallmatrix} D \\ C \end{smallmatrix}}$  と書く . Clock  $C = C(t)$  は 0 と 1 を周期的に繰り返す関数とし , 簡単のため , clock の立ち上がり (0 から 1 に変わるとき) の時刻  $t$  を整数にとる . 即ち ,

$$C(t) = \begin{cases} 1, & n < t < n + 0.5, n \in \mathbf{Z}_+, \\ 0, & \text{otherwise} \end{cases}$$

とする . このとき , 次の回路は clock の立ち上がりで見たとき , 遅延回路  $I \xrightarrow{\boxed{+1}} O$  ( $O(t) = I(t - 1)$ ,  $t \in \mathbf{N}$ ) になる<sup>28</sup> .



問 . Clock の立ち上がりの時刻  $t \in \mathbf{N}$  で見たとき ,  $O(t) = I(t - 1)$  になっていることを具体的な  $I(t)$  を例としてあげて確認せよ .

注 . Flip-Flop 回路によって遅延回路を作る方法では , clock は実際の時間とは何の関係もなく , 0 と 1 がくり返される毎に時刻が 1 進んだと考えることができる . つまり , clock を実現する際に , 発振器の特性は論理的には重要ではない !<sup>29</sup>

時刻 (clock) で桁を表す約束の下で , 任意桁の任意の自然数の演算を論理回路で表現できる . 任意の論理回路はパルス発生器 (clock) と NAND 回路で表現できる .



電子回路による算術演算 (つまりコンピュータのこと) が可能なのは , 次に見るように NAND 回路が実際に電子回路の素子で表現できるからである .

<sup>28</sup>Clock は一つの CPU に種類なので記号では clock 端子  $C$  を省略している .

<sup>29</sup>もちろん , 実際の電気素子の特性との関係や計算機の性能 (スピード) という意味では重要 .

§1.4. 半導体

論理の中心的な機能を電子回路の中で受け持つ基礎的な回路素子は diode と transistor である。

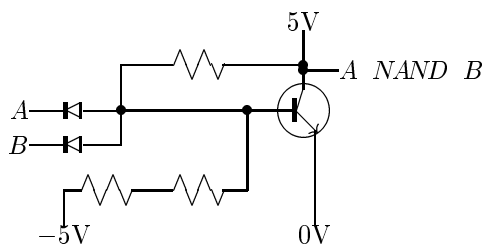
diode		2層の半導体	AND, OR
transistor		3層の半導体	NOT

いずれの素子も各半導体層から1本ずつ端子が出ていてそれを他の回路部品とつないで回路を作る。

Diode は (例えば) 矢印の根に (抵抗を通して) 正電圧  $V$  をかける。矢印の先の電圧  $V'$  が  $V$  より低いときはほぼ導体 (ショート) になり, 矢印の根は電圧が  $V'$  (電圧降下),  $V' = V$  だと絶縁体のように振る舞うので根は  $V$  のまま。矢印の先の電圧の変化で矢印の根の電圧を制御できる (diode のスイッチ作用)。

Transistor (npn型の場合) は矢印の先 (エミッタ E) を接地 (0ボルト) して, 反対の端の層 (コレクタ C) に (抵抗を通して) 正の電圧をかける。中間の半導体層 (ベース B) が 0ボルト以下だと E-C 間は絶縁するので C の電圧は正。B が正電圧だと E-C 間はほぼ導体になるので C の電圧は 0 になる (transistor のスイッチ作用)。Transistor には信号の増幅作用もあるので, 複雑な回路でも情報が減衰しないための役割も担っている (Diode も transistor もスイッチ作用があるので, 上記以外の使い方もできる。) <sup>30</sup>

NAND 回路は次のように実際に電気回路素子を用いて構成できる。



以上によって, この節 §1 で電子回路で情報処理が可能なこと (の基本的なアイデア) が証明できた。以後は具体的な回路や論理の詳細までは至らずに, コンピュータの情報処理の諸側面を説明する。しかし, 実際にコンピュータが動いているということは以下の節で説明する機能も実際に回路で書くことができるということの意味する。

現在これらの回路素子多数を一つの小さな部品 (げじげじ) におさめたものが通常用いられる <sup>31</sup>。コンピュータは情報伝達・処理が目的なので大電力 (エネルギー) を扱わないから大電力の場合に比べれば部品を小さくできる。基本的な回路素子 (diode, transistor, 抵抗, コンデンサー) は全て半導体に少量の不純物を混ぜて作ることができる (端子は金属)。半導体の板 (ウェーハース) に不純物で絵を描くように <sup>32</sup> 回路を作れる。

部品が小さいことが決定的

半導体 → 高集積 → 小型, 高速, 大量生産

diode, transistor → IC (integrated circuit 集積回路)

LSI (large scale integration): gate (diode, transistor)  $10^3 \sim 10^4$  → VLSI (very large SI):  $10^5 \sim$   
one-chip microprocessor

小型なので, 巨大なシステムが小さく安くできる上に動作が速い。300MHz ならば  $3.3 \times 10^{-9}$  秒に一動作。宇宙で最も速い光 (30万 km/秒) ですら, 1m しか動かない。つまり, 1m より大きな CPU はもはや現在のパソコンの動作速度では動かない!

<sup>30</sup> transistor にはここにあげた bipolar 型 transistor 以外に, MOS (metal oxide silicon) 型 transistor がある。それぞれが更に何種類もあり, 用途に応じて使い分けられる。

<sup>31</sup> 回路の集積という考えは1970年前後に急速に発展した。アポロ計画が重要なきっかけであった (月ロケットに載せるために小型化が必要)。1980年代にはパソコンや microprocessor 搭載家電製品は普通になった。

<sup>32</sup> もちろん微細な「絵」なので機械が行う。



## §

- 制御装置（～命令の解釈）と演算装置（～命令の実行）を一つの chip（げじげじ）に収めた装置。
- 計算機の数（可能な clock 周波数）を決める重要な要素（一つの処理に対してやらなければならない作業が大きく変わることはないから，CPU が一つであるような計算機では速度向上とは clock 周波数をあげるといふこと，CPU が電子回路的にスピードに対応できないといけない）。
- 具体的な構造の詳細は CPU の種類によって大きく違う．Intel<sup>33</sup> は CPU を IBM-PC(パソコン製品の defacto standard) 用に採用させることに成功して大儲けした（Apple 社は Motoroller の CPU ）。
- Intel 社は 10 年で 5 種類の構造の大きく違う新製品を発売した．同じ種類の中でも細かく違う製品がある．データ（プログラム (§0.3)）の大規模化の圧力に伴う主記憶装置の大容量化に対処できることと，clock スピードの向上への対処．

## §2.1. 制御装置

プログラム（命令）は（2進法の数字列で書かれた）データとして読み込み<sup>34</sup>，論理回路（即ち計算）によって処理する (§0.2, §0.3)．主記憶装置は数字列（データ）を並べてあるだけで切れ目がない。

一つの命令の区切りをどうやってつけるか？

最も簡単な解決方法：長さの単位を決める．

$W$ = 主記憶 1 番地あたり bit 数 = 1 word (語)
$D$ = 1 回の主記憶 access で読むデータ量 = 主記憶と CPU の間でデータ (含命令) を読み書きする配線 (data bus) の本数
$A$ = 主記憶容量を表すに十分な桁数． = CPU が指定可能な最大の番地 (word 総数)． = CPU から主記憶の番地を指定する信号の通る配線 (address bus) の本数．
$C$ = 命令の (データとしての) 長さ．

番地． 制御装置が主記憶に access (§0.2) するときの参照番号「主記憶に番地がついている」というのは，数字が主記憶装置に記録してあるのではなく，CPU が主記憶を access するとき，数字に対応する場所を計算してそのデータを access するという意味なので，主記憶に関する説明に見えて，実は CPU に関する述語．Intel は番地の単位を 1 word = 1 B = 8 bit，つまり  $W = 8$  と選んでいるようだ<sup>35</sup>．

一つの命令は word の自然数倍の長さ．実際は重要な命令は  $C = W$ ，細かい命令は（サブメニューのように） $C = 2W, 3W$ ，などと使い分ける．

$D$  も  $W$  の自然数倍．CPU 内部の処理に比べて記憶装置への access は時間がかかるのでまとめて読み込むほうが早い，CPU の回路は大きくなる．その兼ね合い．かつては  $D$  を CPU の bit 数と表現していた（例 32 ビットパソコン）<sup>36</sup>．

Intel の主な CPU の変遷（8080A 以降）．

<sup>33</sup> パソコン用 CPU の defacto standard.

<sup>34</sup> von Neumann 型計算機の最大の特徴は命令もデータであるという点だが，通常，命令のことをデータと呼ぶのはハードウェアの話をしているときに限る．数学者以外にとっては混乱するので，応用に近いところではデータというプログラムを指さないようだ．

<sup>35</sup> 哲弥流．

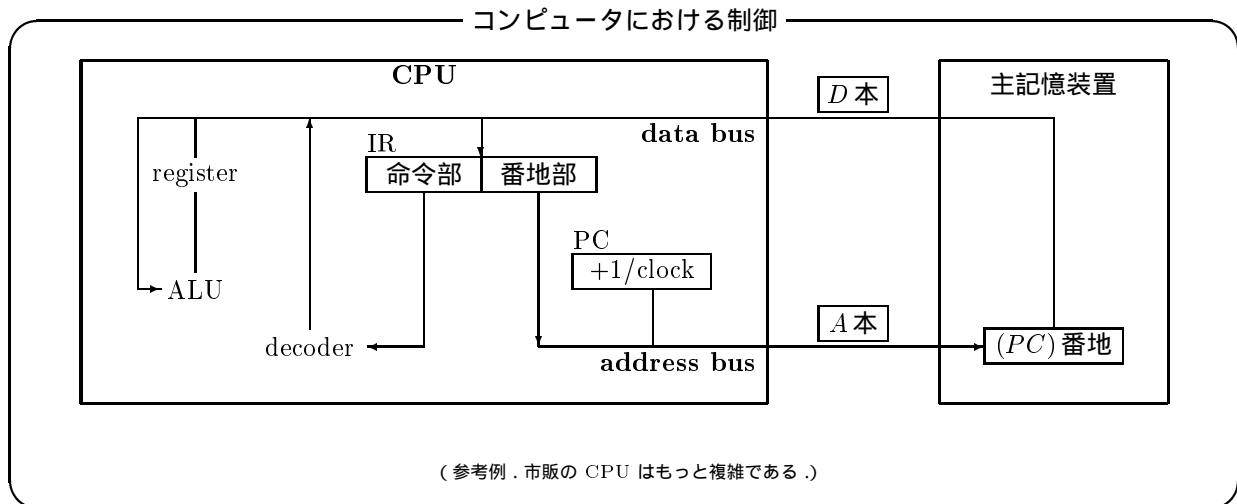
<sup>36</sup> Pentium 以降はいわないみたいだ．

CPU	発売年	Clock <sup>37</sup>	D	A	最大番地 2 <sup>A</sup>
	単位	MHz	bit	bit	word
8080A	1974		8	16	64K
8086	1978	8	16	20	1M
80286	1982	12	16	24	16M
80386	1985	25	32	32	4G
80486		40	32	32	4G
pentium		100	64	32	4G

問. 上の表の空欄を調べて埋めよ. W についても調べよ.

注. 主記憶を access する命令のことを考えると  $D > A$  とするのが自然だと思う<sup>38</sup>が, 1980年代までは  $D \leq A$  が普通だった. 極端に言えば 1 bit ずつ ( $D = 1$ ) 読み込んでも処理は可能だが, A の小さい CPU や OS は大きな主記憶容量を扱えないことが欠点になるため, A の値が先に急速に大きくなったのではないかと<sup>39</sup>. 将来の拡張も考えて大きめに取るのが理想だが, 今日では  $A = 32$  なので十分<sup>40</sup>これからはばらくの間 CPU 製品は A を固定して D を増やそうとするだろう<sup>41</sup>.

しかし, OS を含めたプログラムが word 長を意識した設計でないとハードウェアの機能を生かし切れない (§6).



PC 制御装置の内部構造の話のときは PC は program counter のこと. 次に実行すべき命令が主記憶上のどの番地かを記録してある. 命令を読み出す度 (clock) に内容に 1 を加えるようになっている. ジャンプ命令の場合は IR の番地 (または主記憶のその番地に入っている数字) に置き換わる. 電源を入れたときと reset ボタンを押したときは 0 になる (とにかく, 決まった値にする).

IR Instruction register. 主記憶から読み出した命令の CPU における保存場所.

decoder 命令の解読装置. 各装置に信号が送られて一単位の処理の実行が起こる. ここでいう命令やプログラムは, CPU が理解できる (decoder が解読できる) 命令という意味で機械語とも呼ばれる. 日常会話でコンピュータのプログラミングと呼んでいるのは高水準言語で別のもの (§7.2).

<sup>37</sup> Clock 周波数は私が自宅で使った計算機製品の実例.

<sup>38</sup> 哲弥流.

<sup>39</sup> 哲弥流. 意味のある仕事ができるためにはプログラムはある程度大きくならざるを得ない. ケインズ経済学で, 生きていくための消費はある程度以下は小さくできない (ために, 単純なアダムスミス以来の市場原理ではうまくいかない) というのと同じ.

<sup>40</sup> <sup>32</sup> = 4G word まで対処可能. HD (§3.2 でも  $O(1)$ GB しか容量がないから, 十分. パソコンでの主記憶の実装で私が知っている例は 100MB 以内だが, この点は仮想記憶 (§6.3) 参照.

<sup>41</sup> 哲弥流.

以上の機能は、遅延回路 (§1.3) で説明したような考え方で、全て電子回路で実現できる。もちろん、単純な遅延回路に比べればとてつもなく製作は大変である（さもないければ Intel がもうかるはずがない）<sup>42</sup>。

例． 主記憶装置の各番地の記憶内容が下記のようになっていて、program counter の値が ( $PC$ ) = 100 とする<sup>43</sup>。

番地	値
000	
	⋮
100	MOV
101	200
	⋮
200	5
	⋮
$2^A$	

MOV という数値が命令として解釈されると、引数の番地のところの数値を第 0 register に読み込む（コピーする）処理が行われる（ように CPU が設計されている）としよう。次のように処理が進む。

- (i) ( $PC$ ) = 100 なので address bus を通して主記憶 100 番地が指定され、100 番地の数値 (MOV) が IR の命令部に data bus を通して読み込まれて ( $PC$ ) = 101 になる。
- (ii) ( $PC$ ) = 101 なので上と同様に主記憶 101 番地の数値 (200) が読み込まれるが、MOV はアドレスを引数に持つので（CPU がそう設計されているので）読み込まれた値は IR の番地部に読み込まれて ( $PC$ ) = 102 になる。
- (iii) Decoder が IR の MOV 200 を解釈して、address bus を通して 200 番地が指定され、その数値 (5) が data bus を通して第 0 register に読み込まれる。
- (iv) ( $PC$ ) = 102 なので address bus を通して主記憶 102 番地が指定され、次の命令が読み込まれる。

例 2． 主記憶装置の各番地の記憶内容が下記のようになっていて、( $PC$ ) = 300 とする。

番地	値
000	
	⋮
050	
	⋮
300	JMP
301	50
	⋮
$2^A$	

JMP という数値が命令として解釈されると、引数の番地のところに飛ぶとしよう。次のように処理が進む。

- (i) ( $PC$ ) = 300 なので address bus を通して主記憶 300 番地が指定され、300 番地の数値 (JMP) が IR の命令部に data bus を通して読み込まれて ( $PC$ ) = 301 になる。

<sup>42</sup> 哲弥流。しかし、他の全ての数学と同様、計算機も神業ではなく、人間の技に過ぎないという認識は重要だと思う。

<sup>43</sup> 例に取り上げる命令は実際のものではなく、あくまでも参考例。現在の CPU は複雑になっていて命令の形式も複雑多岐に亘る。

また、実際は MOV という文字が記憶されているのではなく、ある 2 進整数の数値が ( $MOV = 10001011 (2) = 8b (16)$  のように) 記憶されている。この数値を decoder が解釈すると MOV に相当する処理が行われるが、人間に読みやすいようにこの数字を MOV とおく。以下、このような注釈を省略して、あたかも MOV と主記憶に書かれているように比喩的に説明する。なお、このように機械語を単純に英単語（など）置き換えたものは実際にプログラミング言語として用いられ、アセンブラ (§7.1) と呼ばれる。

- (ii)  $(PC) = 301$  なので, MOV と同様に 301 番地の数値 (50) が IR の番地部に読み込まれて  $(PC) = 302$  になる .
- (iii) Decoder が IR の JMP 50 を解読して, PC の値を 50 にする .
- (iv)  $(PC) = 50$  なので address bus を通して主記憶 50 番地が指定され, 次の命令が読み込まれる . つまり, プログラムが 50 番地に飛んだことになる .

## §2.2. 算術演算装置

**ALU** Arithmetic logical unit. 全加算器 (§1.3) などの具体的な基本的計算回路が複数収まっている . 全加算器で例示したように, 全ての計算が電子回路として実現できることが重要 . 回路に入っていない計算は回路の計算を組み合わせて実現するようプログラムを組む (Compiler を使うと, 良く知られた関数は自動的に機械レベルのプログラムに直すので, プログラマーは電子回路なのかプログラムなのか意識しないですむ (§7.2) .)

どういう回路を収めておくかは回路の複雑さ (製造技術水準) とプログラムの複雑さや計算速度との兼ね合い . 1980 年代のパソコン CPU では実数計算の回路は別の補助 CPU に入っていた (かプログラムを作った) が, 今日ではこれらも一つの chip に入っているようだ<sup>44</sup> .

ALU は主記憶のデータどうしを直接計算しない . 足し算のように複数のデータを処理する場合は, 少なくとも一方を CPU 内の記憶装置 (register) にコピーしておく .

**Register** CPU 内の記憶装置の総称 . ALU が直接演算できるデータ格納場所というのが元々の機能 . 他の機能としては, instruction register, flag register, stack pointer 等がある . ハードウェア (CPU) を作る時に機能毎に場所を分けて (信号の通り方を場所毎に分けて) 設計することもできるし, ハードウェアは機能を区別せずに register を複数個用意しておいて, プログラム (機械語) を書くときに自分で機能を割り当てる設計にもできる . 回路の複雑さとプログラムの作りやすさ (長さ) や処理速度の兼ね合い .

## § 記憶装置

性質	意味	用途	素子
揮発性	電源を切ると記憶内容が消える性質	主記憶装置	RAM (半導体素子) (§3.1)
不揮発性	電源を切っても記憶が消えない性質	補助記憶装置	磁気材料, CD-ROM (§3.2)
		micro program (§3.1)	ROM (半導体素子) (§3.1)

問 . なぜ, 主記憶装置は揮発性の記憶素子でよいのか? どんな記憶でも電源を切る度に内容が消えたら次に使うときに困らないか?

## §3.1. 半導体メモリ

素子	特徴 (利点)	用途例
RAM	書き込み高速	
SRAM	速度	キャッシュ
DRAM	容量 (単価)	主記憶
ROM	不揮発	
mask ROM	消去不能	micro program
EEPROM <sup>45</sup>	消去書き込み容易	IC card

<sup>44</sup> 哲弥流 .

RAM Random access memory. 任意の番地をいきなり読み書きできる<sup>46</sup>.

SRAM Static RAM. Flip-Flop 回路 (§1.3) による記憶 (static (動きがない) といっても, 電流は常時流れるので消費電力は大きい).

DRAM Dynamic RAM. CCD による記憶. リフレッシュ ( $10^{-3}$  sec 程度毎に電荷の放電消失分を補充) 必要.

ROM Read only memory. 書き込み不能<sup>47</sup>.

CCD Charge coupled device (複数の MOS を密着させて電荷のやりとりができるような構造の素子).

MOS Metal oxide silicon (金属 - 酸化物 ( $SiO_2$ ) - 珪素 ( $Si$ ) (半導体)) 半導体素子.

micro program 比較的基礎的な演算は論理回路をハードウェアで設計することもできるし, もっと基礎的な回路にプログラムを通して計算することもできる. 後者の立場で比較的基礎的な演算を行うプログラムを作ってそれを ROM に記憶して, CPU の一部としたもの. 外部から見るとそのような演算を行う回路が CPU に組み込まれているのと等価.

IC card 補助記憶装置 (§3.2). 立教大学の学生証.

問. DRAM は CCD 記憶素子だが, ROM が今日主に何で作られているか調べよ<sup>48</sup>.

### §3.2. 補助記憶装置

「システムとしての記憶装置」から見た主記憶装置.

小容量 現在 (1990 年代前半) のパソコンの主記憶装置は 100MB 以内. 個人が常時利用するデータ (含プログラム) は 1GB に迫る. 利用頻度が少ないが保存が必要なデータ (辞書など) を含めると膨大.  
揮発性 主記憶装置は RAM だから電源を切るとデータが消える. 何度も使いたいデータやプログラムが残せない.

交換不可能 主記憶装置は計算機に固定しているので, ネットワーク (§13) でデータのやりとりがしにくい他の計算機や電子機器には, その内容を転送しにくい. また, データのバックアップは原理的に取り外しできる交換可能な媒体が望ましい.

→ 補助記憶装置

注. • 主記憶装置は 32 bit CPU では 4GB を越えられない. 実装はもっと少ない. 私が今この文書を書くのに使っている計算機は家にある中で一番主記憶装置が大きい, それでも 40MB より小さい. 一方, この文書は  $O(100)$ KB, この文書を読むのに必要な  $T_E X$  というプログラムは, アプリケーションプログラムとしては小さい方だが, それでも  $O(100)$ KB. これだけなら容量が足りるが, 実は,  $T_E X$  を動かすのに必要な font などのデータが  $O(10)$ MB もある. 主記憶装置しかないとはほかの仕事のためのデータやプログラムが残せない.

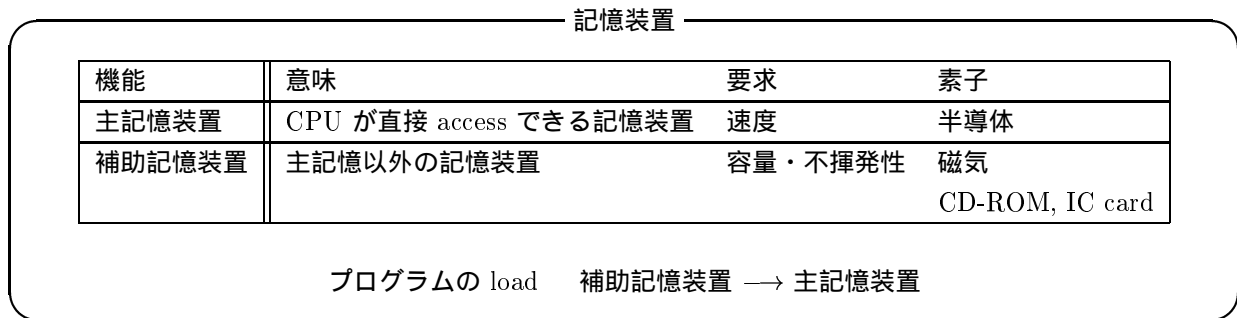
<sup>45</sup>Electronically erasable programable ROM.

<sup>46</sup>初期の頃は半導体が主記憶, 磁気テープが補助記憶装置だった. 磁気テープは目的のテープ位置まで巻かないと読み書きできないのが, 主記憶装置に向かない最大の欠点. それとの対比で random access が特徴. RAM と ROM という語呂合わせもあるのかも知れない.

<sup>47</sup>書き込み可能なものがなかった時代に RAM との語呂合わせで名付けたか. 半導体性の random access 可能な記憶素子という意味では RAM の一種と分類すべき ( $[1, \text{ロム}]$ ) かも知れないが RAM と ROM は通常は対立概念として使われる. 今日における定義はむしろ電源を切っても消えない不揮発性メモリのこと.

<sup>48</sup>磁気パブル記憶素子は不揮発性半導体記憶素子だから ROM として使える. 他方, CCD についても  $\log(\text{放電速度}) \propto \text{access 速度}$  ならば RAM, ROM の両方に用いることができるはず. 問題はそれが可能か, 可能としても ROM として良く用いられるのは何か, という事.

- 1980年代の日本においてなお、FD (floppy disk) を宅急便で運ぶのとネットワークとどちらが有利かという議論が真剣に行われていた。飛行機がどんなに高速であろうと、「燃料を捨てて飛ぶ」飛行機は原油輸送には使えないから、造船技術が無意味になることはない。同様に、ネットワークがどんなに進もうとも、既存の計算機社会が続く限り、交換可能記憶媒体の意義は残るだろう<sup>49</sup>。
- バックアップは情報処理システムの原理的な要求 (§0.1) だから、計算機関係機器製造・販売会社がバックアップの必要性を積極的に一般ユーザーに強調し、便利で安価な商品を宣伝しようとしないうのは、極めて遺憾である<sup>50</sup>。



- (i) CPU は bipolar 型半導体による高速処理を行う。Clock speed が 100MHz ならば  $10^{-8}$  秒 = 10nsec (ナノ秒) だから、主記憶は速さが最大の要求。半導体 RAM。
- (ii) 補助記憶はデータの長期保存や主記憶装置に入りきらないデータの一時保持 (仮想記憶 (§6.3))、および、媒体交換によるデータ転送が目的。磁気記憶素子。半導体 ROM は不揮発性だが高価なので、大量のデータを記憶しておくのは非現実的。
- (iii) 補助記憶の内容はいったん主記憶にコピーしてから CPU が処理する (プログラムの load)<sup>51</sup>。主記憶の内容は計算機の電源を切ると消えるので補助記憶にコピーして長期保存する。

新素材。 補助記憶装置は長く磁気記憶素子 (古くは MT (magnetic tape, 磁気テープ) 等、最近では HD (hard disk, ハードディスク), FD (floppy disk, フロッピーディスク)) だけだったが、高速化と大容量化の要求に伴い、特に、交換可能な記憶装置について新素材も急速に発展。

CD-ROM レーザーで硬い反射性の材質を焼いて細かい溝を作って、弱いレーザーの反射でそれを読む (レコードの針が光線になっただけ)。不揮発性で  $O(10^9)B = O(1)GB$  という大容量。書き込み不能なのでアプリケーションソフトウェアやデータの販売に使われる。<sup>52</sup>

MO Magnetic optical disk. 原理を知らない<sup>53</sup>。大容量、低速、交換可能。CD-ROM から HD にコピーしたデータは大容量なので、カスタマイズ (購入した標準設定 (default) の状態を個人が使いやすい状態に細かい変更を行うこと) 後のバックアップのためには書き込み可能で交換可能な大容量の記憶媒体が必要。1980年代末に登場したが、バックアップ用として急速に広まるだろう。

IC card ROM 半導体を名刺サイズの薄いカードに収めたもの。小容量だが不揮発性で読み書き可能かつ高速。

<sup>49</sup> 哲弥流。  
<sup>50</sup> 哲弥流。パソコンには HD バックアップ媒体を付属して売らねばならない。  
<sup>51</sup> 1996年現在、CPU の高速化に伴って、補助記憶装置を含めた周辺装置の遅さが性能を決めるようになったため、また、man-machine interface への要求に伴い大量のデータを高速処理する要求が高まったため、周辺装置間のデータの移動は専用の microchip に任せる傾向が見える (§4, §3.3)。  
<sup>52</sup> 音楽の CD と同様、レコードの延長上で規格を作ったため、コンピュータの CD-ROM 読みとり装置で CD の音楽も聴けるという利点 (?) の一方、random access がやりにくい (転送速度が損) 欠点もある。この間隙を狙って、また、データ密度向上などの技術的進歩とともに、DVD (digital video disk) などの新しい媒体・規格がこれからも現れるであろう。  
<sup>53</sup> 哲弥流。多分、磁気テープ型記憶媒体だが、レーザーで高密度に磁気的な記録を行うのだろう。

磁気記憶素子． 古典的な補助記憶装置は全て磁性体を塗ったテープまたは板に電磁石を近づけて磁化したり磁化を電気信号に置き換えることで読み書きする．現在でも大容量読み書き可能揮発性記憶素子の主流の原理．古典的には MT (magnetic tape) が最も有名．極めて安価で，テープを長くすれば大量のデータを入れられる．現在の技術をもってすれば，大容量の記憶素子になれると思う<sup>54</sup>が，規格を変更すると読みとり装置を取り替えなければならなくなるためか，古い(いまとなっては小容量の)規格しかない．

例．2400feet<sup>55</sup>, 6250bpi<sup>56</sup> の規格だと

$$2400 \times 6250 \times 12/8B = 22.5MB.$$

FD (floppy disk, 1.25, 1.44MB) 15 ~ 20 枚分だから，交換可能性とコストを考えると使えるが，MO (magnetic optical disk) など新素材に押されて消えるだろう<sup>57</sup>．

HD と FD． 円板に磁性材料を塗って磁化によって記憶する媒体．回転させて，head<sup>58</sup>で一周分(1 track)のデータを読み書きする．Head に腕がついていて，半径方向に円板の縁から中心付近まで動くことで，何周分ものデータを読み書きできる．

FD Floppy disk, フロッピーディスク．軟らかい円板を厚紙や硬質プラスチックで保護したもの．軽くて安く，交換可能記憶媒体として便利．カセットテープやビデオテープと同様に head を円板に密着させて使用．1.44MB (DOS/V) または 1.25MB (NEC)<sup>59</sup>．

HD Hard disk, ハードディスク．硬い円板を読みとり装置とともに密封したもの．円板と head の相対位置を固定できる上にゴミが入りにくいため，head を円板に接触させずに近づけて読みとることができるので，高密度大容量高速 access に向いている．O(1)GB．

読みとり装置が円板にぶつくとゴミが発生して壊れるので，運搬は慎重を要する<sup>60</sup>．交換可能媒体にはあまり向いていない<sup>61</sup>．

FD の記録方式． 3.5inch<sup>62</sup> 2HD (両面高密度倍トラック) の FD を NEC-PC<sup>63</sup> の format<sup>64</sup> で利用する場合．

Format (入出力) の単位 = 1sector = 1KB．FD 一つあたり，2面(裏表)，各 77track (半径方向の分割)，1track あたり 8sector．合計  $2 \times 77 \times 8 = 1232KB \approx 1.2MB$ ．セクタ番号が track 内一周，表裏，の順でついている．

例題． 片面半径方向に 2track に亘って傷が入った．読めないセクタの間隔は何セクタ (KB) か？

解． 8sector/track で表裏の順にセクタの番号がついているので， $8 \times 2 = 16KB$  間隔で読めないセクタが生じる．

Sector あたりの access 時間は seek time と search time とデータ転送時間の和．

<sup>54</sup> 哲弥流．

<sup>55</sup> 英尺の長さの単位．1foot = 12inches  $\approx$  30cm.

<sup>56</sup> bit per inches. 記録密度

<sup>57</sup> 哲弥流．情報処理技術者試験では，まだしばらく磁気ディスクとともに計算問題が出るかも知れない．

<sup>58</sup> 読み書きするための電磁石を収めた小さな部品．記録面に近づくように読みとり装置が作られている．レコードの針に相当する部品．

<sup>59</sup> Apple 社の format は全然知りません．

<sup>60</sup> Retract 機能 (計算機の終了手続きを行ったときに，head を安全な場所に移動する) がついているのが普通のはずだが，精密機器であり，壊れたときのデータ損失の被害は重大なことには変わりがない．特に，使用中にいきなり電源を切ってはならない，と，昔は厳しく注釈があった．

<sup>61</sup> HD の back up 用に等しい HD をもう一つ用意する方法はよく聞かすが，私は実例を見たことがない．

<sup>62</sup> 8.75cm. 円板の直径．

<sup>63</sup> 日本のパソコンの defacto standard．自動車における左ハンドルのように，USA のパソコンが日本語処理という商売上の当然の配慮をさぼっていたため，日本だけ IBM-PC が流行らなかった．しかし，NEC も EPSON 製の互換機に対 EPSON プロテクトという時代遅れな嫌がらせを行うなど自分の首を絞めたため，どっちもどっちになってしまった経緯がある．以上は哲弥流の観察．

<sup>64</sup> 記録方式のこと．記録方式を登録して利用できるようにする (初期化する，今までの記録内容を完全消去する) ことも format と呼ぶ．後者は大変危険なのに避けて通れない作業だったが，今日では format 済みで売られていることが多いためか覚えて分りにくいかも知れない．なお，最新の NEC-PC は IBM 系の format が読めるが，逆は読めない．

seek time head 位置決め時間 . ヘッドを該当 track まで動かすのに要する時間 . 114ミリ秒 .

search time 平均待ち時間 . データ転送速度 62KB/sec , 回転速度 360 回転/分 . 1 回転に  $60 \times 1000 / 360 = 167$  ミリ秒 . 平均するとデータ読み込み命令が来てから , 該当のセクタが head のところに来るのに  $167 / 2 = 83$  ミリ秒 .

転送時間 データ転送速度が 62KB/sec でセクタが 1KB なので , sector あたり  $1000 / 62 = 16$  ミリ秒 .

問 . 私は今手元に NEC-PC しかないので , NEC-PC の format で説明した . IBM の format (1.44MB) について各自で調べて計算してみよ .

FD と HD の使用上の注意

- FD 円板を触らない . 変形 (折るなど) しない .  
磁石 (電磁鍵など) や飲み物を近づけない . 極端な高低温を避ける .  
使用開始時にパソコン (OS) に対応した format を行う .  
使っていた FD を format すると , それまで入っていたデータがなくなる .
- HD 動作中 (普通は動作ランプがついている) に電源を切らない . 落とさない .

### §3.3. キャッシュメモリ

キャッシュは速い素子を遅い素子のコピーとして利用する .  
素子間の動作速度や access 速度の違いが大きいから .

装置	動作時間	素子	容量
CPU	$O(10^{-9})$ 秒	SRAM	—
キャッシュメモリ	$O(10^{-9})$ 秒	SRAM	$O(10^4)B^{65}$
主記憶装置	$O(10^{-8})$ 秒	DRAM	$O(10^7)B$
ディスクキャッシュ	$O(10^{-8})$ 秒	DRAM	$O(10^6)B$
補助記憶装置	$O(10^{-2})$ 秒	HD	$O(10^9)B$

- CPU (bipolar 型半導体) の動作速度が clock あたり  $10^{-9}$  秒とすると , clock 速度は最大  $10^9 \text{Hz} = 1000 \text{MHz}$  だから ,  $O(100) \text{MHz}$  が clock 速度の限界になる .
- 主記憶へのアクセス時間は待ち時間 (address bus) と転送時間 (data bus) の和 . 半導体メモリのデータ転送速度 (bit/sec) は data bus の本数 (§2.1)  $D$  をアクセス時間で割ったもの .
- DRAM の refresh は  $O(10^{-3})$  秒 (§3.2) だから , 「非常に長い」時間記憶を保持していると言える . 但し , 偶然本屋で目にとまった雑誌 ([13]) によると ,  $O(10^{-5})$  秒ごとに refresh しているとのこと . なぜだろう?
- FD のアクセス時間は §3.2 .

例えば主記憶にデータを取りにいくなかに CPU は 10 動作分無駄に待つことになり , 結果として CPU の速度が遅いことになってしまう .

キャッシュは , 論理的分類 (システムの要素) と素材の分類をずらすことでこの無駄を (期待値の意味で) 減らそうとする . 例えばキャッシュメモリは CPU の chip に作り込むので , 主記憶装置の CCD 素材に比べて速い . 主記憶装置のデータの一部のコピーがおいてあり , 主記憶装置のうちキャッシュに入っているデータは , キャッシュから読むことで平均的に速くなる .

<sup>65</sup>Pentium PRO は  $512 \text{KB} = O(10^5)B$  のキャッシュメモリを持っている .



上表では速い記憶素子の1割を遅い記憶のキャッシュとして用いている。遅い記憶の0.1%しかキャッシュにはないが、一般にデータの使用頻度は大きな偏りがあるので<sup>66</sup>みかけよりもはるかにキャッシュの効果がある。また、キャッシュがあまり大きいと、キャッシュにデータがあるかどうか探すのに時間がかかるので、大きすぎるのは不適当。キャッシュの大きさに関する理論は存在しないが、経験則によれば適当な値の範囲が存在するらしい。

キャッシュにないデータを呼び出すたびにキャッシュにもコピーをおく。普通 LRU (least recently used) data を追い出す。キャッシュにあるデータを更新する命令が出たときは、例えばキャッシュともとの記憶装置と同時にデータを更新する。

Pentium 以前は安価な半導体記憶素子<sup>67</sup>に比べて CPU が計算機の性能を決めていたが、CPU の高速化に伴って記憶装置の遅さが性能のネックになり、キャッシュは重要な要素になった。キャッシュは装置間のデータ転送を仲介するので、現在のデータ転送は bus でつなぐだけ (§2.1) の機械的なものではなく、転送管理専用の chip (キャッシュの管理は高度な制御を必要とすることに注意) が性能を決める重要な要素の一つになっている。

## § 入出力

進歩が激しい分野で、しかも理論的一貫性の乏しい、細かい話になるので、入門的講義には向かないが、使用上は必要なので実習による習熟を勧める。

私の個人的な経験を年表にする。各自の経験と比べて、変わりやすいもの、変わりにくいもの、その理由などを考えてみよう。

時期	入力	出力	交換可能記録媒体
1970年代後半	パンチカード	line printer	なし
1980年代前半	key board	display, line printer	MT
1980年代後半	key board	display, dot printer	FD (5in)
1990年代前半	key board	display, dot printer, laser printer	FD (5in)
1990年代後半	key board, mouse, scanner	display, dot printer, laser printer	FD (3.5in), CD-ROM

CPU と主記憶装置以外、即ち、補助記憶装置、入出力装置などを周辺装置と呼ぶ。周辺装置のメーカー(製造会社)と計算機のメーカーは異なるので、周辺装置と主記憶装置間のデータ転送や、CPU による周辺装置の制御の方法には規格がある。

制御方式(主記憶装置と周辺装置間のデータ転送の方式)。

名前	内容	欠点
直接制御	CPU (register) 経由で転送	CPU の処理能力が無駄
DMA <sup>68</sup>	CPU と主記憶間を論理的に切断して直接転送	失敗に対応できない
I/O <sup>69</sup> channel	CPU が channel に命令, channel が制御	専用 microchip が必要

パソコンの I/O interface (具体的な接続の規格)

規格	主な想定接続先	採用例
RS-232C	network	MODEM, 電子カメラ
SCSI <sup>70</sup>	補助記憶装置	HD, CD-ROM reader, scanner
セントロニクス	printer	dot matrix printer

<sup>66</sup>DNA 配列の大部分は酵素を作らない junk であり、極めてわずかの部分が致命的に重要なため分子進化が遅い、などの偏りがあることが知られている。

<sup>67</sup> 哲弥流。比較的単純な構造を持つ記憶素子は日本の得意分野だが、円高がその安さに(外国では)貢献したのだろうか。

<sup>68</sup> Direct memory access.

<sup>69</sup> Input and output.

<sup>70</sup> Small computer systems interface.

## 第 章 ソフトウェア

### § ソフトウェアの種類

ハードウェアとソフトウェア． von Neumann 型計算機というシステムをハードウェアとソフトウェアに分けて考えた (§0.1)．

hard/soft	実体	法的保護 <sup>71</sup>	増産コスト	システム上の位置
hardware	もの	特許	大	汎用性
software	プログラム (含仕様, 手引)	著作権	≈ 0	個別性

#### ハードウェアとソフトウェア

システムの側面 一つの hardware に対して複数の software を利用して個別の問題を処理する．  
 社会的側面 hardware と software を買うときは何にお金を払うかが異なる．

注. (i) ハードウェアは不良品交換や version up が難しいがソフトウェアは容易，という議論もあるが，昨今の CPU の急速な発展や，ソフトウェアの巨大化に伴うインストールの難しさを考えると，単に原理的な違いで，実際的には差があまりない気がする<sup>72</sup>．

(ii) コンピュータというシステム構成要素という意味ではその境界は微視的に不明瞭な部分もある (例えば micro program (§3.1)) ．

巨視的に見ても，今日のように大企業で分担して巨大なソフトウェアを開発した場合はもはや個人が書いた本を著作権によって保護する状況よりはものとしての商品に近い面がある<sup>73</sup>．他方，ハードウェアも設計思想を取扱説明書で理解しないと使えない<sup>75</sup>．さらに，ソフトウェアがハードウェアを意識するのは当然だが，今日ではハードウェアの設計は最初からソフトウェアのあり方を前提にしている．

ソフトウェアの階層構造． (§0.4)

階層	内容	細分類・例
User		
Application program	個別問題の処理	自作・order made・package
Middleware	定型化した処理の環境	§8
Tool	ハード・OS 環境監視	システムモニタ・システムエディタ
Compiler	§7.2	
OS	§6	
Hardware		

ミドルウェア，特にデータベース管理システム (DBMS) や通信処理は商業的に極めて重要で原理的にも難しい内容 (前者は大量データを高速処理する際に曖昧さをどう許容するかなど，後者は計算機間の対話確立方法や通信の誤り対策など) だが，この講義では省略する．

<sup>71</sup>ハードウェアはものであって，回路などの設計は特許として保護されるのに対して，ソフトウェアは知的財産であって，プログラムのアルゴリズムは著作権によって保護される．

<sup>72</sup>哲弥流．

<sup>73</sup>哲弥流．例えば，コピーすれば容易に「盗める」という知的財産の特徴に対して，ソフトウェアが巨大になれば盗むのにディスクがたくさん必要，マニュアルやサポートなどの追加情報がなければ動かない，そして，もっと重要なことは，公理 1 の帰結として，バグ<sup>74</sup>のない巨大プログラムはない．従ってバグや不便があちこちに残るため，version up の support を受け続けなければ動かない．これらの理由で容易に「盗め」ないものになっている．この点は，著作権によって保護されてきた伝統的な著作 (絵画，文章) においてはバグという概念が無意味なのと極めて対照的であり，特許によって保護されるものと著作権によって保護されるもの，というハードウェアとソフトウェアの分類は 50 年後にどう言い換えられているか私には見通しが無い．

<sup>75</sup>今日の機械に関しては取扱説明書がないとあまりに動かしようがないためか，技術者は省略して「取り説」と呼ぶのが普通のようなのである．

階層構造の分化． コンパイラ (§7.2) も素人には難しい．他方，問題に習熟するに従って処理すべきプログラムを複雑にする拡張性はユーザーにとっても利点である．また，プログラムの要求の増大や修正・改訂要求（メンテナンス）の急速な増大に伴う software crisis の下で，エンドユーザー（一番何も知らないユーザー）も簡単なプログラミングを行うのが一番手っ取り早い状態が続いている．

Middleware の中には，ユーザーが制限された範囲内ながら自由にプログラムできるように高水準言語機能（マクロ機能）を有するものがある．ソフトウェアの階層分化の進展の可能性という点から興味深い<sup>76</sup>．

## § オペレーティングシステム

特に CPU 以外のハードウェア資源 (resource) を有効に利用することで，処理効率や interface を改善する目的のために恒常的に動作させるプログラム<sup>77</sup>．

### 基本ソフトウェアとしての OS

OS (operating system) ハードウェアから見て常時走っている唯一のプログラム．  
 Application program OS のデータとして読み込まれ，OS のサブルーチンのように実行される．

↓

プログラマーはハードウェアを直接意識せず，OS が供給する命令群を利用できる．  
 ユーザーは OS を意識せず，プログラムが直接実行可能なハードウェアを想定できる．

↓

ハードウェアに依存しない（互換性のある）プログラム．  
 ユーザーの使いやすい環境，interface (§6.4) ．

- CPU の急激な進歩で CPU の理解する命令（機械語）が変化しても，OS だけ対応して作り替えれば，OS の上で走る既存のプログラムが実行可能．
- 一般にハードウェアの違いによって同じ仕事をするためのプログラムが違う<sup>78</sup>．ハードウェアの違いを覆い隠す OS があれば，一種類のプログラムを全ての計算機に共通に利用できる<sup>79</sup>．)
- 個々の情報処理は，OS の上で動くプログラムをユーザーやソフトウェア会社を書くことで行う．これらのアプリケーションプログラムは OS のデータとして読み込まれ，OS が解釈して適切な命令（の集まり）に翻訳する．結果として OS は本来の CPU と見かけ上異なる環境を提供する．OS が資源の有効利用を計算するので，プログラマーはハードウェア資源の有効利用を具体的に考えずにプログラムを書くことができるようになる．

パソコンでは周辺装置，特に補助記憶装置のデータ管理が OS の最初の重要な役割であった (§6.2) ．今日では，ハードウェア，アプリケーションプログラム両方の巨大化・複雑化とともに，両者の調整機能としての OS の役割が増大している．例えば仮想記憶 (§6.3) ．

<sup>76</sup> 哲弥流．この現象は表計算言語やデータベース管理言語としてエンドユーザー言語という言葉 (§7.2) の視点から論じるのが普通らしい

<sup>77</sup> 哲弥流．恒常的に，という点で人が介入することによる CPU 時間の無駄を防ぐ，ということを含む．

<sup>78</sup> ハードウェアが異なるのに全く同じプログラムが使えるとき互換機と呼ぶ．資本主義経済の原則の下では特許や著作権の問題があるので，既存のよく売れる機種と全く同じものを他社は作れないから，互換機が作られる．

<sup>79</sup> まだ IBM がパソコン市場全体の指導的立場だった頃の Microsoft (パソコン用 OS の defacto standard) Windows の対 IBM 戦略の雑誌的解説．OS そのものはハードウェア毎に用意しないとイケないので Microsoft の一人勝ちになってしまう．現状は Microsoft の巧みな宣伝戦略でそうなっているが，このような解説は考察が不十分だと思う．問．どこに問題があるか考えよ．ヒント．私は正解は知らないが，次の点から考えることができる．この雑誌的解説を信じたとして，そのような OS で恩恵を受けるのは誰か（現在プログラムを作るソフトメーカーとユーザーが分離しつつあることに注意．）そのような OS で節約できるプログラム作成コストと，既存の OS と比べたときのプログラマーへの負担増の比較はどのようなか．

- Application program の入出力関係命令の単純化
- Application program 間での補助記憶装置のデータの共有
- ↓
- Hardware resource の効率的運用 (仮想記憶など)
- Hardware resource の自動検出と自動運用 (PnP (plug and play) など)

OS による装置管理の基本は 割り込み (§6.1) である。

OS の進化 .

- CPU (機械語) の違いに関係なくプログラムが実行できると言っても, CPU の word 長などのハードウェア資源の進歩を生かすような OS の開発が常に必要 (word 長が 2 倍になっても OS がその情報を利用しなければ clock 当たり 2 倍の処理はできない) (§2.1). 32 bit CPU を意識した Microsoft-IBM の OS/2 は現在立ち枯れ. WINDOWS シリーズは 32bit CPU を生かした設計ということに力点はなく, GUI や network に力点 (つまり Apple 社のまね<sup>80</sup>). 既存のアプリケーションへの影響 (MSDOS との上位互換性) もあって簡単に行かないとはいえ, OS がハードウェアに比べて遅れているのは日本だけではない.
- OS とアプリケーションプログラムの間, OS とハードウェアの間, にさらに OS を挟むことも全く同様に可能である. 例えば日本語処理はキーボードからの入力を主記憶に常駐<sup>81</sup>する日本語処理用のプログラムが入力を受け取って, 日本語 (漢字仮名混じり文) に変換してから OS にデータを渡す. OS の管理下で行われているとはいえ, interface 専用の中間的な OS とも思える. もっと強い意味でのソフトウェアの階層化の動きが今後も数十年は続くだろう<sup>82</sup>.

OS の例 .

MVS	multiple virtual sotrage	大型計算機 (大規模システム)
MS-DOS	Microsoft disk operating system	パソコン
UNIX		ワークステーション (ネットワーク)

### §6.1. 割り込み — OS による装置管理

原始的な計算機は異常 (0 による除算, 周辺装置の不良) が起こると停止して, 人が処理をした. 今日では特定の register に 1bit を代入する (flag を立てる). OS は通常の命令実行を行う度に register を監視して (別の register を割込禁止 flag として利用することで, 割り込まれては困るときには割込監視をしないようにもできる), flag が立ったらプログラム実行を中断して割込処理プログラムに jump する. 割込プログラムの終了時には中断点に jump して戻るように番地を記憶しておく (stack register).

例

仮想記憶の動的再配置 (page fault によるプログラムチェック割り込み §6.3)

Multi task (§6.4) のプログラム切り替え .

<sup>80</sup> 哲弥流. Microsoft の OS は, CP/M, UNIX, Apple 社, と, 他社の優れた OS をものまねすることで成長してきた. かつて日本は「ものまね技術」と批判されたことがあるが, 巨大になった会社は成長の過程で重要なことをまねる. 真の問題は, 技術を自分のものできないためにまねをし続けること, そこに安住すること, 覚悟なく独自路線を踏み出してすぐに行き詰まること, などである. そこで技術力が試される. 成功する会社は自ら試行錯誤などの研究によって乗り越える.

<sup>81</sup> OS と同様に最初に load してそのままずっと置き換わらないプログラム. 一定の規則でファイルを作っておくと OS が常駐プログラムとして load する.

<sup>82</sup> 哲弥流.

## §6.2. Disk operating system としての MS-DOS

この節は MS-DOS という特定の OS のみに成り立つ話だが，WINDOWS シリーズでも (MS-DOS で読めるように<sup>83</sup>) 踏襲している。

パソコンの OS は補助記憶装置を使いやすくするためのソフトウェアとして出発した。CPU が直接 access できるのは主記憶装置で，補助記憶装置のデータは基本的には記憶媒体上の指定された場所のデータを主記憶装置との間でコピーすることだけだった (§3.2)。補助記憶装置のデータに対する処理や管理の作業は主記憶にコピーしてからそこで処理することになる。基本的には必要な処理のためのプログラムを書かなければならない (ハードウェアに命令が用意されていないから)。

しかし，補助記憶装置にデータ (ファイル) を蓄えて利用するというのは，

- (i) 目的のプログラムを主記憶装置に load して実行する，
- (ii) データ解析用のアプリケーションプログラムの命令に応じて特定のデータ (2月4日に採取したデータといったふうな) を load してプログラムに処理させる，
- (iii) 計算結果をファイルとして保存し，覚えやすいファイル名をつけておく，

などであって，ディスクの何セクタ目にデータを入出力するかは人から見れば (システムとしてみれば) 二義的な問題である。

最初はまとまったデータをディスクにセクタ順に記録していても

- (i) 目的のデータを毎回端から探すのは時間の無駄 (主記憶装置のように random access をしたい)<sup>84</sup>，
- (ii) FD は小さいので，データを (編集などのために) 書いたり消したりしているうちに空いた途中のスペースにも新しいファイルの一部を書き込みたい。この場合，新しいファイルは媒体上のとびとびの場所に分散することになる，

などの要求を考えると，最低限，媒体上のどこにどういう名前のファイルが格納してあるかを記録しておいて，媒体への access の最初の機会にその記録 (またはその記録の存在場所) を主記憶に load して，アプリケーションプログラム実行時にはその記録を参照しながら必要なデータを load する必要がある。

個々のアプリケーションプログラムにこれらの管理を任せるのはプログラム作成も手間がかかるし，プログラム間でディスク (データ) を共有できなくなる。OS によって補助記憶装置のデータ管理と命令群を用意して，アプリケーションプログラムはこの OS の上で動くプログラムとするのが望ましい。これが disk 管理ソフトウェアとしての OS の原点である<sup>85</sup>。

この考えは CPU と主記憶装置から見た周辺装置を管理するための基本ソフトウェアとしての OS の役割に拡張される (§6)。

MS-DOS format (NEC でも IBM でも) では，先頭のいくつかの sectors (§3.2) にファイルとセクタを対応させた FAT (file allocation table) を記録するので，データの記録に実際に利用できる容量は format された sector 総数より小さい。

3.5in. 2HD NEC-PC の MS-DOS (1.2MB) format の場合。全 1232sectors (§3.2)。

sector 番号	MS-DOS format	数値の対応
0	volume 名など	
1,2	FAT	1.5B/cluster → $(1221 + 2) \times 1.5 = 1834.5 < 2048B$
3,4	FAT (copy)	
5 ~ 10	root directory	$6 \times 1024/32 = 192$ files
11 ~ 1231	データ	1221 clusters = $1221 \times 1024 = 1250304B$

<sup>83</sup>下位互換性

<sup>84</sup>磁気テープの致命的限界 (§3.1)。

<sup>85</sup>CP/M が 8bit パソコン時代の成功例だが，IBM が 16bit に移行するときに Microsoft の MS-DOS を選んだため，衰退。今日のパソコン界の OS の defacto standard は MS-DOS。この命名は Microsoft disk operating system，つまり，パソコンにおける補助記憶装置の基本の disk (FD, HD) を管理する OS ということ。

データ領域のセクタを cluster と呼ぶ<sup>86</sup>。NEC-PC の MS-DOS の format 命令で format すると、

```
1250304 バイト 全ディスク容量
1250304 バイト 使用可能ディスク容量
```

という表示が出て、データ領域の容量が上表の計算と合う。

クラスタ (1KB, §3.2) 単位で入出力を行うので、小さなファイルや 1KB 未満の端数データがあると記憶容量が無駄になる。例えば 'a' という 1 文字 (1B, §12.1) だけの入ったテキストファイルは MS-DOS では 2B (a と 1AH<sup>87</sup>) しか使っていないのに、1KB のデータ領域を占領する (1022B 分は MS-DOS を通しては読み書きできない)。

FAT は cluster 番号に対してその cluster が使用されていなければ 000H, data file の最後の cluster ならば FFFH, 途中の cluster ならばその file の次の cluster 番号, を並べたデータである。一つの cluster あたり 1.5B (000H ~ FFFH) の数字を使うので、上表の計算通り約 2KB の FAT で cluster を管理できる。

Root directory 領域は root directory にある file の file 名, サイズ, 先頭 cluster, 日時, の情報を, 一つのファイルあたり 32B で記録。上表のように, root directory には 192 個のファイルしか登録できない勘定になるが, 実際確かめることができる。

例えば C に format のすんだ新品の FD を入れる。

```
open(1,file='y.bat')
do 1 i=1,9
write (1,'(a,i1)') 'copy y c:',i
1 continue
do 2 i=10,99
write (1,'(a,i2)') 'copy y c:',i
2 continue
do 3 i=100,193
write (1,'(a,i3)') 'copy y c:',i
3 continue
end
```

というプログラムを書いてコンパイルして走らせると y.bat ができる。Current directory に y という名の小さなファイル (例えば a という文字だけの入ったファイル) を作っておいて, y.bat を走らせると

```
A:\>copy y c:1
1 個のファイルをコピーしました.
```

...

```
A:\>copy y c:192
1 個のファイルをコピーしました.
```

```
A:\>copy y c:193
ファイルが作れません.
0 個のファイルをコピーしました.
```

となって, 最大 192 個しかファイルができないことが分かる (一つの FD に記録したいファイルが 192 個より多いときは, sub-directory を作ってその中に入れればよい.)

問. FORTRAN 言語の代わりに C 言語を用いて同様の確認を行え。

<sup>86</sup>細かく言うと, cluster 0,1 は飛ばして, データ領域は cluster 2 から 1222 までとよぶらしい。

<sup>87</sup>MS-DOS ではファイルの最後を表す。MS-DOS だけなので, ネットワークでほかの OS とファイル交換を直接行うと問題を起こすこともある。

### §6.3. 仮想記憶

主記憶装置（半導体記憶素子）は補助記憶装置（磁気記憶素子）に比べて単位容量（記憶情報量 (Byte)）あたりの値段が極めて高いので、速度を損なわない範囲で実装はなるべく小さくしたい (§3.2) . 今日 (1996年) の CPU は 4GB の主記憶を access できるが、実装は  $O(100)$ MB (§2.1) .

- (i) CPU が同じでも実装している主記憶の大きさによってプログラムが走ったり走らなかったりすると不便ではないか？(人によって予算が違うから実装している記憶装置は違う．アプリケーションプログラムを売る側としては、実装毎に違うプログラムを書くのは大変．)
- (ii) 実装している主記憶装置より大きな巨大プログラムは走らせることができないか？
- (iii) Multi task (§6.4) のようにいくつものプログラムを主記憶に置いておいて、切替の命令（割り込み）でプログラムの間をジャンプしたい．主記憶が巨大でないと諦めるしかないか？
  - (i) 仮想記憶は OS が、実装の主記憶装置よりも大きい仮想記憶空間を設定し、仮想アドレスで番地をつける．番地は一定の長さ（ページ、例えば 4KB）毎に一まとまりとして扱う．
  - (ii) 補助記憶装置の空き領域にページ単位で仮想記憶空間と対応をつける．
  - (iii) プログラム（巨大なものや複数の場合を想定）の load 命令が来ると、OS はプログラムにページ単位で仮想アドレスを対応させ、該当の番地の補助記憶装置にコピーする（静的再配置）．アプリケーションプログラム側からは、プログラムが実際に主記憶に load されたかのように見える．
  - (iv) Clock 毎に、実行する番地を含むページを主記憶に実際に load して実行する．ただし、既に当該ページが主記憶にあれば load しない．主記憶装置も OS がページ単位で分割管理し、ページ枠表を用意して、主記憶上の各ページに現在 load されているプログラムの仮想アドレス、最近参照があったかどうかを示す参照 bit、内容が処理によって変更されたかどうかを示す変更 bit、が記録されている．
  - (v) アプリケーションプログラムは仮想アドレスでデータを指定する．これをページ枠表で実際の主記憶上のアドレスに変換して CPU が処理を行う（動的アドレス変換）．

DAT Dynamic address translator. ページ枠表を調べて該当ページが載っていれば主記憶上のアドレスがわかる．

連想レジスタ 最近参照 bit が 1 のページの情報だけ並べてある．少ないので全数調査で高速に参照できる（プログラムの次の行はたいてい同じページにあるので、参照したページを続けて参照する可能性は高い）

主記憶上に指定された仮想番地を含むページがないとき (page fault) このままでは CPU は命令を実行できないので、プログラムチェック割り込みを起こして、制御を監視プログラムに制御を渡す．

- (vi) 監視プログラムは補助記憶上の該当ページを探して主記憶上の適切なページ（空いているページ、なければ、最近参照 bit の立っていないページ．変更 bit が立っていれば、補助記憶上にコピーしておく (page out)）に load する (page in) .
- (vii) Paging (page out/in) の間は補助記憶の access が行われて CPU の時間が無駄になるので、multi task (§6.4) でほかのプログラムを実行する．

元々仮想記憶は大型計算機 (main frame) の技術だったが、今日のパソコンでは例えば Windows95 は仮想記憶をサポートしている．

注. Paging が増えると明らかに速度は非常に小さくなる．制御がプログラム上の離れたところにしょっちゅう移動する (jump 命令などの多い) プログラムは paging が増える．即ち、建前 (§6) に反して、プログラムを書くときハードウェアを意識しないとイケない．これはおかしいと思う<sup>88</sup> .

<sup>88</sup> 哲弥流．当面は、コンパイラ (§7.2) の optimization の工夫に頼ることになるだろうが．

## §6.4. Multi task

- 一つの CPU を複数のユーザーが同時に利用すること．TSS (time sharing system) ．
- 長い計算をさせている間に文書を読みたい．と、いったようにいくつかの仕事を同時進行で行うこと．
- ある仕事が遅い周辺装置を利用しているとき、速い CPU をほかの仕事に使うほうが効率的、といったように、ハードウェア資源を有効利用すること．

主に大型計算機の技術．パソコンにも本格的に導入されるだろう．

入出力は  $O(10^{-3})$  秒，CPU は  $O(10^{-9})$  秒．一つのプログラムが入出力をしている間に他のプログラムを実行すれば，OS の複雑化による処理の増大を考慮しても，処理効率を  $10^4$  倍にすることも原理的には難しくはない<sup>89</sup> ．

注．疑似 multi task ．思考の中断をさけるために，ユーザーから見て multi task に見えるようにした man-machine interface<sup>90</sup> ．例えば，文書作成作業中に情報が必要になって，文書を見ながら browser で home page を見に行きたい．更に，見に行った先である商品の単価が分かったので，合計価格を文書に書き込むために文書や home page を残したまま電卓計算を行い，その値を残したまま文書作成に戻る，といったこと．これも OS による割り込み管理 (§6.1) の例ではある．

監視プログラム (OS) が multi task を監視する．

- 主記憶上に複数の独立なプログラム (task) を load しておく．
- 各 task は実行状態，実行可能状態，待ち状態，のいずれかにあり，割り込みによって状態を切り替える．

プログラム割込 プログラム実行エラー (0 による除算，overflow，記憶保護違反) ．Task を異常終了 (削除) させる．

機械チェック割込 ハードウェア誤動作．再実行と CPU の停止．

SVC Supervisor call. プログラムから監視プログラムへの入出力要求．監視プログラムはチャンネルプログラムを書いて，当該 task を待ち状態にして，チャンネルプログラムを実行する．

I/O 割込 入出力装置からの入出力終了信号．異常終了の場合は，入出力要求を出した task を異常終了させる．

外部割込 CPU を時分割 (time slice  $O(10^{-3})$  秒) して複数の仕事 (task) が「代わりばんこに使う」タイマー割込．その他の割込．

Spool. 補助記憶装置を仮想的な入出力装置として用いることで，補助記憶装置よりさらに低速な入出力装置への access を multi task 化すること．Multi task の場合は，印刷が混乱しないように spool を行っておいて，別のプログラムによって実際の出力を統一的に管理する必要がある．

Job と task ．

Job	人から見たときの処理すべきプログラム	compile, link, 実行, I/O
Task	CPU から見たときの一つの job の 最小の制御単位	running, ready, wait

Multi task の場合，複数の平行して走るプログラムが一つのデータファイルを参照・更新すると，プログラムから見たとき，知らない間にデータが変わってしまう恐れがある．矛盾が起きることを避けるためにデータ管理が，task 管理と job 管理とともに，multi task OS の重要な機能になる<sup>91</sup> ．

<sup>89</sup> 哲弥流．

<sup>90</sup> 哲弥流．

<sup>91</sup> この講義では省略するが，情報処理技術者試験ではファイルの構造に関連して勉強しておくのが望ましい．



## § コンパイラ

### §7.1. アセンブラ

CPU が処理するプログラムは機械語で書かれ、補助記憶装置にファイルとして記録しておく。実行時に主記憶装置にコピー (load) して、CPU が解釈・実行する。機械語は 0 と 1 の 2 進数の列 (§2.1)。これをひとが作成するのは覚えにくく、間違いやすい。

機械語の命令と一対一に対応する英単語に近いアルファベット文字列と記憶装置の番地などの数字でプログラムを表したものを assembly 言語と呼ぶ (§2.1)。

Assembly 言語で書かれたテキストファイルを作ることは機械語でプログラムを作るよりはひとにとってやさしい。Assembly 言語で書かれたテキストファイルをデータとして読み込んで、機械語のプログラムに翻訳するプログラムを assembler と呼ぶ<sup>92</sup>

### §7.2. コンパイラ

Assembly 言語は機械語と一対一のため、

- (i) (命令の単位が簡単なため) プログラムが長くなり、
- (ii) 有効なプログラムを作成できるようになるのに熟練を要し、
- (iii) 間違いの修正 (バグ取り) や機能拡張改訂など (メンテナンス) が難しく、
- (iv) CPU 毎に命令が違うので違う計算機や CPU の更新毎に最初からプログラムを書き直さなければならない、

等の問題がある。CPU の高性能化に伴い、ソフトウェアの要求は増大し、また、既存のソフトウェアの訂正 (debugging) や改訂 (機能拡張) などのメンテナンスも増大する (software crisis) 状況では上記の assembly 言語の欠点を解決しないとイケない。

機械語との一対一対応を諦め、assembly 言語よりもさらに日常言語に近い言語でプログラムできるようにした言語を高水準言語、これを機械語に翻訳するプログラムをコンパイラと呼ぶ。

#### Compiler と interpreter.

Compiler	機械語のファイルを生成し、それを実行する	速い
Interpreter	高水準言語を一行毎に機械語に翻訳して実行する	debug や機能拡張が容易

Interpreter の例として BASIC, APL などがある。数式処理ソフトは interpreter 型のものが多い。

コンパイラ言語の例。

言語	典型的な文	歴史 (仕様書)	得意とする用途
FOTRAN	10 CONTINUE	IBM 大型計算機 (1954)	数値計算
COBOL	COMPUTE K = K -1	IBM (1960)	事務処理
PASCAL	begin ... end	ALGOL → PASCAL (1968) → ADA	言語理論, 教育, 国防省
C	#include <stdio.h>	UNIX OS (1972)	OS

このほか、list 構造を中心に持った LISP が 1950 年代に開発され、最近数式処理ソフトや人工知能などのプログラム言語として再脚光をあびている。

<sup>92</sup> Assembler は機械語で書かないといけない。但し、一対一なので単純であることと、既に他の計算機に assembler や高級言語があれば、それを用いて新しい CPU の assembler を書くことができる。

**Optimization.** アルゴリズムが同じでも、プログラムは一つとは限らない。仮想記憶 (§6.3) のように、ユーザーが本来意識すべきでないハードウェアにプログラムの効率が影響される場合、一つの解決策は、コンパイラがハードウェアを見ながら翻訳先の機械語のプログラムを選ぶことである。一般に、コンパイラがハードウェアや OS の制約を意識して特定の目的（スピード、記憶容量、など）を達成するために機械語のプログラムへの翻訳方法を選ぶことを optimization と呼ぶ。経験的には、そんなにすぐれた話はないように見える<sup>93</sup>。

## § ミドルウェア

定型化した処理のための環境を提供するプログラム。昔は自作したが、計算機が巨大化してプログラミングが大変になった。他方、ユーザー数が膨大になって、ソフトの制作・販売が商売として成立。そのバランスで「定型化した処理」の具体的範囲が決まる。しかし、純粹に商業的な概念ということではなく、個々のジャンルごとに興味深いプログラムアルゴリズムの概念が含まれている。

例<sup>94</sup>。

機能	概念	例
エディタ		VZ, WZ
ワープロ（文書整形）	font, box	T <sub>E</sub> X, 一太郎
日本語処理	front end processing	ATOK
作表	spread sheet, active cell	Supercalc, Lotus
データベース管理	relation, 探索言語	dBase
通信	記号空間（暗号, 訂正, 変換）	CCT, ALMAIL
画像処理		Postscript, Happy Paint
数式処理	リスト（木）構造	Mathematica, REDUCE

日本語処理については §12.1, active cell については §10.2, 木構造については §9.2.1, も参照。

<sup>93</sup> 哲弥流。

<sup>94</sup> 哲弥流。辺りを見回して書いていただけなので、完全は期していないし、バランスや一貫性もない。需要と供給に応じて急速に変化する表なので、講義の趣旨から考えても完全を期す意味はないだろう。

## 第 3 章 アルゴリズム

### § ソート

整列（定められた順序に一列に並べること）．例：辞書の単語の順序．計算機では、通常、補助記憶装置に保存した多量のデータを読み出して整列して再び補助記憶装置に保存するための計算機プログラムのアルゴリズム (§10.1) を問題にする．

文献． ソートは基本的な算法である上に、データベースの構築という商業的に極めて重要な応用があるので、非常に詳しく研究されていて状況に応じた種々の方法が提案され、そのアルゴリズムとしての性能が分かっている．ここでは手元にあった [9, 4, 11, 12] を参考にしたが、むしろ [6] を勧める<sup>95</sup>．

#### §9.1. 一般論

計算機ではデータは特定のプログラムで読み書きされることを前提にしている．データは、キー（例：見出し語）と内容（例：単語の意味）の組を情報として含む．プログラムはデータを一定の順序で読み書きするか、キーを指定することで特定のデータを読み出すことができる．

以上のことを前提にして、アルゴリズム論で問題にするソートは  $N$  個のデータ  $x = \{x(i) \in X \mid i = 1, 2, \dots, N\}$ , を小さい順に並べなおすアルゴリズムを指す． $N$  のことをデータの大きさ（サイズ）と呼ぶ<sup>96</sup>． $X$  はデータの集合（一般には半順序集合）だが、ここでは  $X = \mathbf{R}$  の場合だけを扱う．

##### §9.1.1. 目安となる量

回数． 理論的な研究のしやすい量として考えられるもの：

- (i) 可能な初期配列のうちで最悪の比較回数（初期条件によらないこと（安定性）の目安）
- (ii)  $N!$  個の可能な初期配列の集合上に一様確率測度を入れたときの終了までの比較回数の期待値（平均比較回数）（アルゴリズムの平均的な性能，速さ，の目安）
- (iii) 最悪の交換回数．
- (iv) 初期配列に一様確率測度を入れたときの交換回数の期待値．

ここでは比較回数を問題にするが、交換回数も目安になる．交換 ( $swap(a, b)$ ) は計算機のハード（メモリ）レベルでは3回の代入  $c \leftarrow a, a \leftarrow b, b \leftarrow c$  として実行されるので代入の回数を目安にするのが適切．

問． 使用環境に応じて入れ替え回数と比較回数のどちらが目安として重要か（あるいは同程度か）は変わりがうる．その判基準は何か？

記憶容量． 入れ替えを行うために必要な補助的な記憶場所（作業領域）が限られている場合には記憶容量も性能の目安になる．個々のデータが大きな記憶場所を占める場合（単なる数字ではなく、数字をキーとする複数フィールドのデータなど）には実際に入れ替えをしない方法が有利な場合もある．

<sup>95</sup> 応用数学的な分野の教科書では特に顕著だと思うが、著者が内容を良く分かっていて表現にも気を使う人でないと、著者の誤解や説明不足のために細かいところで記述に誤りや混乱が生じて、大事どころで勉強の障害が生じる．1980年代から90年代のソート関係の教科書の著者は [6] を勉強して書いている（またはそれを勉強した別の著者の本を勉強して書いた）と容易に推測されるので、[6] を全面的に越えるのは至難の業である．手元にあった本の著者が Knuth を越える実力を持っているとは思えない（そうでなくても例えは [11] の説明は勧められない）．手元になかったにも関わらず [6] を勧める次第である．

<sup>96</sup> 大きさといってもデータ自体の数値の大小ではないことに注意

## §9.1.2. Uniform bound

定理 1. 最悪の回数  $k_{max}(N)$  は

$$\liminf_{N \rightarrow \infty} \frac{1}{N \log_2 N} k_{max}(N) \geq 1$$

を満たす. 即ち,  $k_{max}(N)$  は  $N$  が大きいときほぼ  $N \log_2 N$  以上である.

証明. 整列したデータ  $x(i) = i, i = 1, 2, \dots, N$ , から出発して  $k$  回の操作を行うことを許すと, 1 回毎に操作を行うか行わないかで最大 2 倍の異なる配列が生成できるので  $2^k$  種類の配列を生成できる. ソートはこの逆操作である.  $N$  個のデータがあるとき,  $N!$  種類の配列をソートしなければならないから,  $2^{k_{max}} \geq N!$  でないと最悪の場合をカバーできない. Stirling の公式

$$\lim_{N \rightarrow \infty} \frac{N!}{N^N e^{-N} \sqrt{2\pi N}} = 1$$

を用いれば定理を得る. □

定理 1 の気持ち.  $\lim_{N \rightarrow \infty} \frac{1}{N \log_2 N} k_{max}(N) = 1$  を満たすアルゴリズムがあれば,  $k_{max}$  に関する漸近的に最善のアルゴリズム<sup>97</sup>である.

定理 2. 平均回数  $k_{ave}(N)$  は

$$\liminf_{N \rightarrow \infty} \frac{1}{N \log_2 N} k_{ave}(N) \geq 1$$

を満たす.

証明. 定理 1 で考察した状況の逆操作が最も効率がよい. 即ち  $k_{ave}$  の下限を与える. 定理 1 の  $k_{max}$  を  $k_m = k_{max}(N)$  と書くと,

$$\begin{aligned} 2^{k_m} &\geq N! > 2^{k_m} - 1, \\ N! k_{ave}(N) &\geq \sum_{k=1}^{k_m-1} k 2^{k-1} + k_m (N! - 2^{k_m-1}) = k_m N! - 2^{k_m} + 1. \end{aligned}$$

これより,  $k_{ave}(N) > k_m(N) - 1$  を得る. □

定理 2 の気持ち.  $\lim_{N \rightarrow \infty} \frac{1}{N \log_2 N} k_{ave}(N) = 1$  を満たすアルゴリズムがあれば,  $k_{ave}$  に関する漸近的に最善のアルゴリズムの一つである.

## §9.2. 主なソートアルゴリズム

一番速いアルゴリズム (複数ある時はどれか一つ) だけを使えばいいように見えるが,

- (i) データサイズ  $N$  が小さいときはすぐ終了するので,  $N$  が大きいとき, 即ち漸近的な評価に主な興味がある. 逆に言えば, データの大きさが小さい場合は漸近的に速いプログラムより, 簡単なプログラムがかえって速く, またプログラムの間違いが少ない,
- (ii) 実用上, 使用目的によって初期配列が一定の傾向がある場合には初期配列として起こりやすいものに重みをつけた確率測度を入れて期待値を考えるべきであり, 一般的には遅いアルゴリズムであっても有利になることがある. 例えば, 部分的に並んでいるデータが多いのが普通,

<sup>97</sup>複数あるかもしれない.

- (iii) 値域の集合が限られているときも速い方法がある． $X = S_N(1, 2, \dots, N)$  ( $(1, 2, \dots, N)$  の置換) のときは  $x$  を自然数上の写像と見てその逆写像  $y$  を求めるのが速い (ハッシュソートは値域の集合を逆写像の計算が速い集合にいったん変換するハッシュ関数を用意する方法である) ,
- (iv) 使用可能記憶容量が少ないときは遅くても作業領域が少なくてすむアルゴリズムが必要である ,
- (v) データが全順序集合 (実数) のキーを持たない (半順序集合である) 場合にソートしたい (この場合 , 一列に並べる「自然な」方法は複数あり得るので , 解が一意でない問題を解くことになる) ,

などの理由によっていくつものアルゴリズムが提案されている . これらの問題の検討は省略するが ,  $X = \mathbb{R}$  の場合の比較的普遍的なソートを列記する .

名前	平均的比較回数	最悪のケース	特徴・制限・利点等
ヒープソート	$N \log N (1 + o(1))$	$N^2$ 平均と同程度? 大 同程度?	二分木データ構造・選択法
クイックソート	$N \log N (1 + o(1))$		再帰的手続き・交換法
マージソート	$N \log N (1 + o(1))$		作業領域大 ( $2N?$ )
ハッシュソート	小		ハッシュ関数・逆写像
シェルソート <sup>98</sup>	$O(N^{1.5})?$		挿入法
基本選択法	$N(N-1)/2$	$N(N-1)/2$	
基本交換法 <sup>99</sup>	$N(N-1)/2$	$N(N-1)/2$	
基本挿入法	$N^2/4 (1 + o(1))$	$N(N-1)/2$	

最後の3つは遅いが自明な方法 . 最初の4つは自明な方法を改良したものといえる . 速くするためには複雑な改良 (§9.2.1) が必要であり , それ故にアルゴリズムという概念が重要になる .

—— アルゴリズムという概念がなぜ重視されるか? ——

殆どの場合 , 理論的に無駄のない (速い) プログラムを書くには直感のききにくい方法 (アルゴリズム) を工夫しないとイケない .

基本選択法 .  $x(k), 1 \leq k \leq n-1$  ( $n = 1, 2, \dots, N$ ) が整列済みの最初の  $n-1$  個の小さいデータになっているとき , 残りのデータを順番 ( $x(k), k = n+1, \dots, N$ ) に配列上の  $n$  番目のデータ (最初は初期値  $x(n)$ ) と比較して , 小さいときは交換することで  $n$  番目のデータまで終了する (最小値候補用作業領域  $x_m$  を用意して比較・代入したほうが代入回数換算の交換回数は少なくなるが , こだわらないことにする .)

比較回数は常に  $N(N-1)/2$  .

基本交換法 (バブルソート) .  $x(k), n+1 \leq k \leq N$  ( $n = 1, 2, \dots, N-1$ ) が整列済みの最後の  $N-n$  個の大きいデータになっているとき , 順番 ( $k = 1, \dots, n-1$ ) に  $x(k)$  と  $x(k+1)$  を比較して大きいほうを右に持っていく (交換が起きると大きい方のデータは続けて右隣と比較するので非常に大きなデータはバブル (泡) のように右に続けて移動する) と  $n$  番目のデータまで整列が終了する .

比較回数は常に  $N(N-1)/2$  .

基本挿入法 .  $x(k), 1 \leq k \leq n-1$  ( $n = 1, 2, \dots, N$ ) が整列 ( $i < j \rightarrow x(i) \leq x(j)$ ) している (小さい方から  $n-1$  個とは限らない) とき ,  $x(n)$  をこれらと小さい方から順に比較して , 最初に大きいのが出てきたときにその直前に挿入すると  $n$  番目のデータまで整列する .

交換回数も比較回数も最悪で  $N(N-1)/2$  .

命題 3 . 交換回数の平均は  $N(N-1)/4$  .

<sup>98</sup>シェル (Shell) のみ人名 .

<sup>99</sup>通常バブルソートと呼ぶ .

証明. データ  $x(i)$  の総移動回数は自分より左 ( $j < i$ ) にある自分より大きい ( $x(i) < x(j)$ ) データと右にある小さいデータの和. これを  $i$  について加えて (交換は二つのデータの移動だから) 2 で割れば交換回数を得る.

期待値は

$$\begin{aligned} & \frac{1}{2} \sum_i \left( \sum_{j < i} \text{Prob}[x(j) > x(i)] + \sum_{j > i} \text{Prob}[x(j) < x(i)] \right) \\ &= \sum_{i, j < i} \text{Prob}[x(j) > x(i)] = \sum_{i, j < i} \frac{1}{N} \sum_k \left( 1 - \frac{k-1}{N-1} \right). \end{aligned}$$

□

$0 \leq \text{比較回数} - \text{交換回数} \leq N - 1$  なので, 平均比較回数の漸近形は平均交換回数のそれに等しい.

問.

- (i) 表の空欄や曖昧な欄を検討して表を完成せよ.
- (ii) 各方法について交換回数の最悪の場合と平均交換回数を求めよ.

ハッシュソート. ハッシュ関数とは  $f: X \rightarrow \{1, 2, \dots, M\}$  なる  $f$  であって,  $M$  が  $N$  に比べてあまり大きくなく, 「なるべく」一対一になっているもの.

通常は,  $X$  が文字データなど整数順序になじまないデータを整列する関数として探索のキーを与える手段として使う. データは全てハードウェアレベルでは2進数に変換されるから, 数式によって  $f$  を表現することはできるが, データのとりうる値  $X$  は  $N$  に比べて巨大であり, それをあまり大きくない  $M$  までの自然数に一対一に対応させるのは不可能である. データの特性に応じて頻出の値が区別できるように関数を選ぶ, というのが主旨である. 運悪く同じ  $f$  の値を与える二つのデータがあれば「衝突が起こった」といって, その場合の対処をアルゴリズムの設計時に用意しないとイケない.

特に,  $X = \mathbf{R}$  で  $f$  が「だいたい狭義」単調増加関数になっていると, 逆写像を用いてソートに利用できる. 即ち大きさ  $M$  の作業領域配列  $z$  (逆写像) を用意して, 値を 0 (正確には  $f(x(\{1, 2, \dots, N\}))$  に含まれない数) に初期化しておく. 次に  $z(f(x(i))) = x(i)$ ,  $i = 1, 2, \dots, N$ , とおき,  $k = 0$  から出発して  $j = 1, 2, \dots, M$  に対して順に

$$z(j) \neq 0 \implies k \leftarrow k + 1, y(k) = z(j),$$

を実行すると,  $y(i)$ ,  $i = 1, 2, \dots, N$ , は整列したデータになる.

特に,  $X = S_N(\{1, 2, \dots, N\})$  のときは, ハッシュ関数を恒等写像に取れて逆写像法と呼ぶべき方法になり, 交換回数・比較回数ともに  $N$  という, 非常に速い方法になる (一般にも極めて速いが, ハッシュの衝突が起きると極端に遅くなる.)

### §9.2.1. ヒープソート

ヒープソートは二分木に基づくヒープ (heap) という概念を用いる高度なアルゴリズム上の概念を含んでいる. しかし, 極端に長くはないプログラムで理論的に最善の漸近的な速度を得る. 代表的なソートの例として, また, アルゴリズムという概念の重要性の指摘を込めて, 特に取り上げる.

データ構造. データ  $X = \{x(n) \mid n = 1, 2, \dots, N\}$  の間に何らかの2項関係 ( $X \times X$  の部分集合  $R$  のこと) が定義されているとき, データの集合とその間の関係を, データ一つ一つを点と見て関係がある点同士を線でつないだグラフとみなすこと.  $(x(n), x(m)) \in R \rightarrow (x(m), x(n)) \in R$  のとき (無向) グラフ, それ以外, 特に,  $(x(n), x(m)) \in R \rightarrow (x(m), x(n)) \notin R$  のとき有向グラフという.

木 (tree) .

$$\begin{aligned} & \exists k \in \mathbf{N}, x(n_i) \in X, i = 1, 2, \dots, k; \\ & (x(n_i), x(n_{i+1})) \in R \text{ or } (x(n_{i+1}), x(n_i)) \in R, i = 1, 2, \dots, k-1, \\ & (x(n_k), x(n_1)) \in R \text{ or } (x(n_1), x(n_k)) \in R, \\ & n_i \neq n_j (i \neq j), \end{aligned}$$

を満たす列  $x(n_1), x(n_2), \dots, x(n_k)$  をループと呼ぶ .

グラフが連結しているとは ,

$$\begin{aligned} & (\forall \ell, m \in \{1, 2, \dots, N\}) \exists k \in \mathbf{N}, x(n_i) \in X, i = 1, 2, \dots, k; \\ & n_1 = \ell, n_k = m, (x(n_i), x(n_{i+1})) \in R \text{ or } (x(n_{i+1}), x(n_i)) \in R, i = 1, 2, \dots, k-1, \end{aligned}$$

を満たすことをいう . ループのない連結したグラフを木と呼ぶ<sup>100</sup> . 有向グラフの木では矢印の元を親 , 先を子と呼ぶことがある .

二分木 (binary tree) . 各点に高々 2 つの子と高々 1 つの親しかない有向グラフの木<sup>101</sup> . どの親の子でもない点が 1 つだけある . それをルートと呼ぶ . ルートを level 0 呼び , 一般に level  $i$  の子を level  $i+1$  とよぶ .

Binary tree の配列上の表現 . 配列  $x(1), x(2), \dots, x(N)$  と最後の 2 段の level 以外の点は子を 2 つずつ持っている binary tree は , ルートのデータを  $x(1)$  とし ,  $x(s)$  の子を  $x(2*s)$  と  $x(2*s+1)$  とする (木の左の子を先にする) , という対応で , 一対一に対応する .

以下 , この対応に基づいて , 実際のデータは配列に並んでいるにも関わらず , あたかも木になっているかのようにアルゴリズムを表現する .

Heap . 最後の 2 段の level 以外の点は子を 2 つずつ持っている二分木において , 木の各点のデータが , どの親子関係についても親のデータのほうが大きくなっているとき heap であると呼ぶ .

かつて二分木のデータからヒープを作るアルゴリズムは次の通り . 子を持つ最後 (配列上最後) の親  $x(\lfloor N/2 \rfloor)$  から始め ,  $\lfloor N/2 \rfloor, \dots, 1$  の順に以下を行う : 親のほうが子より小さければ , 2 つの子のうち大きい方を親と交換する . 交換した子を新たな親としてそこから下に向かって親のほうが小さければ大きい方の子と交換して親子の逆転がなくなるまでくり返す .

Heap sort のアルゴリズム . 配列の初期値に対応する二分木から始めて , heap を作る . ルート ( $x(1)$ ) は最大の数なので最後のデータ ( $x(N)$ ) と交換する .  $N$  番目のデータは正しい位置に来たので以後固定し ,  $N-1$  個のデータからなる配列とみなして以上をくり返す . 帰納的に木のサイズが 1 になれば対応する配列は昇順になっている .

問 .

(i) 以上の説明に基づいて heap sort のアルゴリズムをフローチャートにせよ .

(ii) Heap sort の平均及び最悪の比較回数と交換回数の漸近形を求めよ .

Heap sort プログラム具体例 .

```
/* heap sort (昇順) */
/* C program from C 言語によるはじめてのアルゴリズム入門 , */
/* 河西朝雄 , 技術評論社 , 1992 年 , p.297--p.299 */
```

<sup>100</sup>立教大学佐藤文広先生の指摘により訂正 (19970530)

<sup>101</sup>立教大学佐藤文広先生の指摘により訂正 (19970530)

```
/* 19970413 */
/* ----- */
#include <stdio.h>

void swap(int *, int *);
void shiftdown(int, int, int *);

void main(void)
{
    static int heap[100];
    int i,n,m;

    n=1; /* データを木に割り当てる */
    while (scanf("%d",&heap[n])!=EOF)
        n++;
    n--; /* データ数 */
    for (i=n/2;i>=1;i--) /* 初期ヒープの作成 */
        shiftdown(i,n,heap);
    m=n; /* n の保存 */
    while (n>1){
        swap(&heap[1],&heap[n]);
        n--; /* 木の終端を切り離す */
        shiftdown(1,n,heap);
    }
    for (i=1;i<=m;i++)
        printf("%d ",heap[i]);
}

void shiftdown(int p, int n, int heap[]) /* 下方移動 */
{
    int s;
    s=2*p;
    while (s<=n){
        if (s<n && heap[s+1] > heap[s]) /* 左と右の子の小さい方 */
            s++;
        if (heap[p]>=heap[s])
            break;
        swap(&heap[p],&heap[s]);
        p=s; s=2*p; /* 親と子の位置の更新 */
    }
}

void swap(int *a, int *b)
{
    int w;
    w=*a; *a=*b; *b=w;
}
```



問. [12] では大きさ  $N = 100$  くらいのデータ  $1, 2, 3, \dots, N$  をランダムに並べたものを初期データとして各ソートのプログラムを走らせ、ソートの途中の何か所かで部分的にソートされたデータ  $y_1, y_2, \dots, y_N$  を打ち出させている. 平面グラフに  $N$  個の点  $(i, y_i)$ ,  $i = 1, 2, 3, \dots, N$ , をプロットさせて可視化している. アルゴリズムの理解のためにたいへん参考になる可視化だと思うので, 実行せよ.

## § アルゴリズムとフローチャート

### §10.1. アルゴリズム

問題を解くための手順の有限列.

計算機のプログラミングの観点からは, 最終的にはプログラム言語に用意されている命令 ( に対応する日本語など ) だけを用いて手順を書くべきであるが, 通常プログラムが長い場合が問題なので,

- 大きな流れや複雑な条件判断付き分岐のみを明示したアルゴリズムを最初に構成する ( 各手順の処理内容は日本語<sup>102</sup>や数式で明示するが, その手順そのものは一行のプログラムで書けなくても構わない )
- 大きな流れを書いたアルゴリズムが用意されたあとで, その各手順をプログラム言語の命令で書ける程度に細かい手順に分割する ( 一つのプログラム ( 一つの手順 ) を完成するのに, 重複をいとわず細かさの度合いだけの異なる本質的に同じアルゴリズムを複数設計するのは珍しくない )
- 決まりきった手続きは省略する場合がある ( 自分が省略することは勧められないが教科書のアルゴリズムは説明のために特定部分だけ抜き出してあることが多い ) 例えは, どの装置から入出力するかが決まっている場合は, 最初から変数に必要なデータが読み込んであるとしてアルゴリズムを書く.

高水準プログラミング言語では主記憶装置のデータを変数として指定できるが, アセンブラ以外について説明している場合は, 通常高水準言語を用いてプログラミングすることを想定しているので, 最初から変数にデータが読み込んであるとして変数を用いてアルゴリズムを書いている. 用いられる手順も高水準言語に用意されている命令を念頭に置いて書かれている. 勉強する際は高水準言語によるプログラミングになれていることが必要である<sup>103</sup>.

以上の意味の具体的内容 ( 例 ) は, 実際の問題 ( §9 など ) に即して説明したい.

計算量の理論. アルゴリズムは, 最終的には処理をあらかじめ定められた単位作業に分解して表現したものである. そこで, 二つの処理の複雑さの度合いをアルゴリズムの大きさと比較できる. 計算機で行われる情報処理 ( 計算や数学的証明を含む ) は全ていわゆる算術計算とみなすことができる ( 数学基礎論, 帰納関数の理論, 計算量の理論 ). 単位作業として算術計算をとることで情報処理あるいはアルゴリズムの複雑さを定量的に論じる研究は数学的に意味があり, 計算機理論にとっても重要である<sup>104</sup>.

- 注. (i) データ管理 ( sort, search, update など ) や巨大配列の科学技術計算などでは, 計算量 ( スピード ) 以外に記憶容量という制限を考慮する必要がある. 利用できる記憶容量とのかねあいで利用できるアルゴリズムが決まる ( スピードを犠牲にしても少ない記憶容量で計算できるアルゴリズムを選ぶ, など )
- (ii) アルゴリズムは決まった手順のことであるけれども, 対象とするデータ ( 初期条件 ) によって計算量は異なる ( 例えは, 最初からそろっているデータの sort はゼロ時間で終わる ) 従って計算量は初期条件の関数であり, 応用上, 初期条件は決まっていないから, 初期条件に関する適当な分布を仮定して論じるべき確率変数である. このほか, 疑似乱数を用いることにより, 確率的な手順を論じることができる.

<sup>102</sup> 自分や, 誰かにプログラミングを頼む場合はその相手にも分かる言語という意味.

<sup>103</sup> 哲弥流. この指摘はアルゴリズムの本質ではないが, 初学者がアルゴリズムという言葉を理解する際に大きな障害になる. アルゴリズムをプログラミング言語から切り離して抽象化する作業が見せかけだけに終わっている現状はエキスパートの責任であるが, 実際の役に立たないアルゴリズム論に実用上の価値がないという現実問題もある.

<sup>104</sup> 計算量の理論はそれだけで単独の講義とすべき大きさを持つので, この講義では扱わない. §9 などの具体例でその視点を想像されたい.

これらのことは当たり前すぎて指摘されずにすぎているように見える．計算量の理論における確率論的視点は，例えば暗号解読の理論において極めて重要である．

フローチャート．流れ図．アルゴリズムのうちプログラムの手順（基本的な個々の手順は長方形，始めと終わりは楕円，判断は菱形，ループは二つの角をとった長方形，などで，各手順を囲み，隣り合う手順を矢印付きの線で結ぶ）やデータの流れ（データの入出力は利用する装置によって，例えば人による入力には台形で囲むなど）を図や記号で表すことで見やすくした図<sup>105</sup>．見やすいので，実際のプログラムに翻訳するときには各自工夫して利用するのがよい<sup>106</sup>．

問． §9.2 で説明した基本的な3つのソートの方法について，説明を参考にしてアルゴリズムをフローチャートで書け．

## §10.2. アルゴリズムからプログラムへ

プログラムは（情報処理の）問題を計算機で解くためにある．問題の解はかならず手順（解法）を持つ．よって全てのプログラムはアルゴリズムを持つ．

Q しかし，それならなぜわざわざ「アルゴリズム」という概念を計算機プログラミングにおいてのみ取り上げて重視するのか？（他の数学の分野ではわざわざ「アルゴリズム」などという単語は使わない<sup>107</sup>）

A 正しくて拡張の容易なプログラムを能率的に作成することと，そのことをプログラマーが意識することが極めて重要だから．

Q なぜその当たり前のことが重要か？

A 人間は間違える<sup>108</sup>．プログラミングにおける人間の間違いを減らす方法として，プログラムの背後にある「アルゴリズム」に意識を集中することが奨励された．

Q なぜ計算機の分野で特に言われ始めたのか？

A 恐らく，既存の学問や仕事の分野では，人間同士がチェックしあうので無意識に「ヒトがおかしやすい間違い」を理解して，相手が修正するのでこれまで強調する必要がなかったのではないかと<sup>109</sup>？計算機は融通がきかないという比喩的な説明は古い世代には多かった<sup>110</sup>．

例． 高水準言語．人間は使い慣れている方法や道具ほど間違えにくいという常識に基づいて，日常言語や通常の数式に近い記号や規則を用いたプログラミング用の言語，ということが出来る<sup>111</sup>．

問．

(i) 上記の常識を検証せよ．

(ii) プログラミングの能率化に役立つ方法論をこの節に述べられているもの以外にも考えてみよ．

<sup>105</sup> 哲弥流．

<sup>106</sup> 標準の記号も定められているが，私が育ってきた時代は標準の記号が変化してきたので，講義では特定の記号にはこだわらない．講義ノートでは作成の時間やスペースの関係でフローチャートを用いない．各自でフローチャートになおす練習をすること．

<sup>107</sup> このごろは使うこともあるが，これは計算機時代になって，学ぶ側が数学という言葉よりもアルゴリズムという言葉に慣れているのではないかと，教える側の数学者の軽い心の痛みが入っているのかも知れない．

<sup>108</sup> 哲弥流．機械もネットワークなどは間違える．すべからずシステムは事故を起こす（公理1）．しかし，人間の間違いが圧倒的に多い．

<sup>109</sup> 哲弥流．なお，この問答は立教大学佐藤文広先生からの指摘に基づいて説明を強化した（19970530）．

<sup>110</sup> そうするとやがては「アルゴリズム」は全ての人にとって当然の概念になって，強調されることがなくなるだろうか？作る人と使う人の分離や，interfaceを意識したプログラミング環境という方向もあるので，そうはならない可能性もある．

<sup>111</sup> 哲弥流．

品質のよいプログラム .

- 仕様書に指定された機能を持っている<sup>112</sup> .
- バグが少ない .
- 実行効率がよい (速い, 必要記憶容量が少ない) .
- 操作性 (man-machine interface) が良い (見た目がきれいで優しい, いらいらさせない, 初心者でもわかりやすい, 使い慣れた人に便利な機能がある) .
- 保守性がよい (訂正, 機能の拡張, カスタマイズ<sup>113</sup>が容易) . 複数のプログラマが参加して混乱なくプログラムできる (意思統一, 分担可能性) .

簡単だが品質向上に役に立つ方法 . チェック用の出力を書き出すように余分な命令を加えておく . 簡単なデータの場合についてプログラムを実行できるようにしておいて, 手で計算した結果や本の実例との比較を行う . 出力の範囲の検査を行う命令を加えておいて, 異常な値が出たら停止したり警告を出力するようにしておく<sup>114</sup> .

アルゴリズムとプログラムの構造化 . 人間の間違いのうちプログラミングの障害になるものの一つに人間の非論理性がある<sup>115</sup> . プログラマーがアルゴリズムという概念を意識することによって, 手順を基本単位に分解し処理の流れを明示する態度を身につければ, 非論理性による誤りを防ぐ<sup>116</sup> . 例えば, 場合分けの不足の発見や, プログラムが異常終了した場合に問題点を発見するのに役立つ .

フローチャートによる図示化が特に望ましい .

誤りの防止訂正を含むプログラミングの能率化のための種々の方法論が, アルゴリズムの方法論としても提案されてきた . 中心的な理論が構造化 (モジュール化) .

構造化 (モジュール化) . サブルーチン (subroutine) . 目標となる処理の一部でありながら, まとまった作業 (処理に必要なデータも処理した結果のデータも比較的少ないか単純なものになる作業) になっている部分を, 部分目標プログラム (アルゴリズム) として別に設計し, 全体のアルゴリズムの中では, その部分をサブルーチンとして既存の命令であるかのように扱う .

いろいろなところに応用できるサブルーチンはライブラリとして登録する .

Object oriented . 数字ではなく機能に名前を付け, 名前を呼び出すとある作業が行われる, という構造を持ったプログラミング言語の規則 . この概念は構造化の最新の成果なのだと思うが, 良く知らない<sup>117</sup> .

GOTO無用論 . 構造化を意識しないと software crisis を乗り切れないという歴史的な標語 . 1960年代頃このような議論が盛んだった高水準言語には jump (次の行以外に制御を移すこと) 命令として, 特定の行に飛ぶよう命令する GOTO 型の命令があるが, これはプログラムの構造を見にくくして保守性が悪くなる原因であるとして, ループ (do while, do until) や条件分岐 (if then else endif) などの構造化された jump 命令だけを用いてプログラムを書くべきであるという主張 .

問 . なぜ do や if のほうが goto より保守性がよいか?

<sup>112</sup> プログラムは目標がある . その目標を明示した文書を仕様書と呼ぶ .

<sup>113</sup> 利用者ごとの好みに合わせて interface などの詳細を変更すること .

<sup>114</sup> 哲弥流 . これらは debug (虫取り) の理論ということになるのであろうが, そのような理論体系を知らないし, あまり強調されているように見えない . 実用上は知っておいた方がよい .

<sup>115</sup> 哲弥流 . 人間が完璧に論理的な生物であったならば, 今日の文系と理系に相当する知的活動の大分類は社会の中での位置がほぼ逆転し, 純粋数学の主要部分は常識として当然視され, 社会の中で重視される「数学」は我々が応用数学と呼んでいるものであつたらう .

<sup>116</sup> この節冒頭の問答の脚注参照 .

<sup>117</sup> 哲弥流 . 作表ソフト (§8) の active cell の概念が object oriented の最初の一般化した実現であると私は思うが, これも知識不足かもしれない .

## § 数値計算

### §11.1. 浮動小数点法

自然数のハードウェア表現は2進法による (§1.1). Compiler などのソフトウェアもその前提で設計されている。これに対して、実数は通常 (2進実数 (§1.1) ではなく) 浮動小数点法で表現する。

理由. 実用上、実数を扱う計算は桁と有効数字を分けて眺めることが多い (実験値など誤差を伴う数値であることが多いため、桁と最初の数桁 (有効精度) の数字が重要であるから.)

IEEE (Institute of Electrical and Electronics Engineers) 規格 実数  $r$  を以下のように 32 bit の 01 列 (2進自然数) で表す (単精度実数<sup>118</sup>).

$r$  が与えられたとき、1 bit の2進数  $S$ , 8 bit の2進数  $e$ , 23 bit の2進数  $f$  を次の規則で決めて、 $Sef$  と並べた 32 bit 2進自然数を  $r$  の表現とする。

- (i)  $r \geq 0$  のとき  $S = 0$ ,  $r < 0$  のとき  $S = 1$  とおく (符号 bit),
- (ii)  $|r| < 2^{-126}$  のとき,  $e = 0$ ,  $f$  は  $|r| \times 2^{126} < 1$  を2進実数で表したときの小数部 (0 と 1 よりなる列) の最初の 23 bit とおく. 即ち,  $|r| = 0.f \times 2^{-126}$ , また,  $0 \leq 0.f < 1$ .
- (iii)  $2^{-126} \leq |r| < 2^{128}$  のとき,  $e$  を,  $2^{e-127} \leq |r| < 2^{e-126}$  を満たす自然数とする. このとき,  $1 \leq |r| \times 2^{127-e} < 2$  となるので, この実数の小数部の最初の 23 桁を  $f$  とおく. 即ち,  $1 \leq 1.f < 2$  であり,  $|r| = 1.f \times 2^{e-127}$  である. 定義から  $1 \leq e \leq 254 = 2^8 - 2$  なので,  $e$  は 8 bit (8 桁の2進数).

- $2^8 - 1 = 255$  も 8 bit だが,  $e = 255$  は実数扱いせず  $e \leq 254$  とする.
- $f$  は 23 bit で表し, 小さい桁は切り捨てるので 情報落ち が起きる.
- 表現可能な最大の実数は約  $2^{128} \approx 10^{38}$ , 即ち, 10進法 39 桁の数<sup>119</sup>.
- 表現可能な 0 に最も近い正の実数は

$$\begin{aligned} & 000000000000000000000000000000000001 \text{ (実際のデータ)} \\ & = 2^{-23} \times 2^{-126} = 2^{-149} \text{ (そのデータの実数としての意味)}. \end{aligned}$$

- $2^{-126} \leq |r| < 2^{128}$  のとき, 表現可能な隣り合う実数の差を  $\delta r$  (有効精度) とおくと, 上と同様に  $|\delta r| = 2^{-23} \times 2^{e-127} \approx 2^{-23} \times |r|$  だから相対誤差  $|\delta r/r| \approx 2^{-23} \approx 10^{6.9}$ , 即ち有効精度は 10進法約 7 桁である.

問.

- (i) 表現可能な最大の正の実数の正確な値を求めよ.
- (ii)  $|r| < 2^{-126}$  のとき,  $|\delta r|$  を求めよ.

(IEEE 規格に従っているかどうかはともかく,) 通常のパソコンは浮動小数点計算専用の回路と命令を持っている。たいていのプログラミング言語も浮動小数点法で実数を表現する<sup>120</sup>。

### §11.2. 情報落ちと桁落ち

実数は浮動小数点法で表現される (§11.1) ので誤差を伴う。

<sup>118</sup> 64 bit で表す倍精度実数の規格もある。

<sup>119</sup>  $2^{10} = 1024 \approx 10^3$  は非常にしばしば用いられる便利な公式。是非覚えておくのがよい。

<sup>120</sup> ハードウェアとプログラミング言語が同一の規格の浮動小数点法でなければ compiler は表現の翻訳を行う。

情報落ち .  $\frac{1}{N} \sum_{i=1}^N x(i)$  を計算するのに

```
s=0.0
do 1 i=1,N
s=s+x(i)
1 continue
answer=s/N
```

というプログラムを書くと、IEEE単精度実数は2進法23桁しか小数部がないから  $2^{23} - 1 = 8388607$  番目以降は  $8388609 \approx 2^{23}$  となって、いくら足しても  $s$  が増えなくなる。これは、 $s$  がたくさんの  $x(i)$  の和なので  $x(i)$  に比べて非常に大きくなり、 $x(i)$  を加えても値の増加が誤差の範囲に入ってしまうからである。結果として答え  $answer$  は真の値に比べて  $N > 2^{23}$  のときはどんどん小さくなる。これを情報落ちという。

```
a=0.0
do 1 i=1,N
a=a*(i-1)/i+x(i)/i
1 continue
answer=a
```

とすれば  $x(i)$  があまり変動が大きいかわ、大体全て正でかつ大体小さい順に並んでいれば誤差は累積しない。

問 . 二つ目のプログラムが正しく  $\frac{1}{N} \sum_{i=1}^N x(i)$  を計算していることを証明せよ。なぜこのプログラムのほうが情報落ちの影響が小さいのか。

桁落ち . 簡単のため仮に計算機が十進法4桁精度とする。二つの実数  $x, y$  は計算機では  $x', y'$  と表されたとすると  $|(x-x')/x| < 10^{-4}$ ,  $|(y-y')/y| < 10^{-4}$  であるが、差を計算すると  $|((x'-y')-(x-y))/(x-y)| < 10^{-4}$  とは言えない。例えば、 $x = 1$ ,  $y = 0.99994$  のとき  $x' = 1$ ,  $y' = 0.9999$  なので、 $|((x'-y')-(x-y))/(x-y)| = (0.0001 - 0.00006)/0.00006 = 4/6 = 0.67$  となって、2桁目すら合っていないこと(有効精度一桁)が分かる。大きな数から大きな数を引いた結果答えが小さな数になるときは有効精度が計算機の規格より減ることを桁落ちという。このような計算をしないように、アルゴリズムを工夫しておかなければならない。

問 . 基本的な数値計算のアルゴリズムにはニュートン法、数値積分、行列に関する数値計算、などがある。また、並列処理計算機が今後の主流になることから、並列計算機向けのアルゴリズムという新しい分野が重要になっている。これらについて調べよ。

## § 文字処理

### §12.1. 文字コード

文字データもハードウェアでは2進整数で表現される(文字コード)。入出力のときの変換は国際的な約束(ISO: International Organization for Standardization)があり、約束に従った変換機能(表示や印刷のためのフォントを含む)がハードウェア(CPUとは別の場所にあることも多い)に備わっている。但し、例えば欧米向けの計算機には日本語のフォントは入っていないので、外国の計算機で日本語の読み書きをするには適当なソフトウェアを入れる必要がある<sup>121</sup>。

<sup>121</sup> パソコンならば計算機ごと持っているのが一番簡単。

アルファベット. ISO 7bit. ASCII (Americal National Standard Code for Information Interchange) 8bit (ISO 7bit 符号に最上位の0を加えて8bit=1Bとしたもの). 1Bは16進法で2桁(0~2<sup>8</sup>-1=255=16<sup>2</sup>-1). 16進法2桁の数を *nnH* のように表すことにすると,

20H	21H	22H	23H	24H	25H	26H	27H	28H	29H	2AH	2BH	2CH	2DH	2EH	2FH
□	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/
30H	31H	32H	33H	34H	35H	36H	37H	38H	39H	3AH	3BH	3CH	3DH	3EH	3FH
0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40H	41H	42H	43H	44H	45H	46H	47H	48H	49H	4AH	4BH	4CH	4DH	4EH	4FH
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50H	51H	52H	53H	54H	55H	56H	57H	58H	59H	5AH	5BH	5CH	5DH	5EH	5FH
P	Q	R	S	T	U	V	W	X	Y	Z	[	≠	]	~	_
60H	61H	62H	63H	64H	65H	66H	67H	68H	69H	6AH	6BH	6CH	6DH	6EH	6FH
'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70H	71H	72H	73H	74H	75H	76H	77H	78H	79H	7AH	7BH	7CH	7DH	7EH	7FH
p	q	r	s	t	u	v	w	x	y	z	{		}	~	

00H から 1FH と 7FH は文字そのものではなく、通信における文字送受信の制御コード。20H の □ は空白(スペース)のこと。

日本語. アルファベットは大文字小文字合わせて  $26 \times 2 = 52$  種。6bit ( $2^6 = 64$ ) では句読点まで入れると余裕がなさ過ぎるが、7bit ( $2^7 = 128$ ) で十分。実際 ISO規格は 7bit だった。ASCIIの規格 1Byte (8bit) で表現可能な記号は  $2^8 = 256$  種。日本ではアルファベットとカタカナ(半角カタカナ)を表している(アメリカではカタカナの代わりにトランプの記号などの簡単な図形)。これらを日本では(次の 2Byte 文字と区別するために) 1Byte 文字とも呼ぶ。1Byte 文字は漢字を表現するには全然足りない。

記号(文字, 数字)の入出力は 1Byte を単位にするのが普通なので、長さを2倍にして 2Byte で表すことにすると  $2^{16} = 65536$  の記号を区別できる。日本の漢字の目安として常用漢字が定められているが、[2]によると 1981年制定時 1945字なので十分に記号化できる。JIS 第1水準と呼ばれる平かなカタカナ漢字の一覧がほぼこれに相当する。

十分余裕があるので、JIS 第2水準としてさほど頻出とは言えない漢字も多数登録されている。手元にある中サイズの漢和辞典 [3] は総画索引に 5937字登録してある。この範囲は余裕を持って登録できることが分かり、日常生活には支障がない。もちろん、世の中にある全ての漢字を 2Byte で表すこともできないし、JIS 第2水準までに存在しない漢字を人名で見かけることがたまにある。まだ定義されていない部分が十分残っており、第3水準以降の制定も続いている。これらは日本特有である(外国のハードやソフトには通常用意されていない。MSDOS や WINDOWS のような外国のソフトも全て日本で売り出すときは日本語化される)。

2Byte 文字には 1Byte 文字に入っているアルファベット, 記号, カタカナが全て入っている。両者を区別するために、2Byte 文字を全角文字, 1Byte 文字を半角文字, とも言う。計算機のハードウェアに標準で組み込んである文字が、2Byte 文字が 1Byte 文字の2倍の横幅を持つからである。

問. エディターで半角文字と全角文字を書いて画面表示の横幅を比べて見よ。

日本語変換前処理. キーボードにはアルファベットがある。日本のキーボードにはカナキーがついていて、カタカナも入力可能なので、半角文字は概ねキーボードから直接入力できる(1回のキーストロークで入力できる)。2Byte 文字は1回のキーストロークでは入力できない<sup>122</sup>。

今日では全角文字入力は通常アルファベット(ローマ字入力)の平かな変換表示と、変換キー(スペースキーまたは XFER キーなど、ソフトウェアによって異なる)によるかな漢字変換、の二段階によってソフトウェア的に行われる。

<sup>122</sup>昔一度だけ漢字がずらっと並んだ漢字入力装置を見たことがある。しかし狭い部屋で使うパソコンには向かない。

OS は入出力の制御を行う。キーストロークをそのまま画面に表示する、というだけでも実際は OS というプログラムが常時動いているから可能である！しかし、ハードやソフトの種々の環境に対処できるように、通常 OS は立ち上げの時<sup>123</sup>に、入出力制御を変更することができるようになっている。他の処理を行うプログラムを用意してそれを然るべき形に保存しておく、OS が立ち上げ時にそれを読んでそのプログラムを含む形で入出力制御を行う。このプログラム部分も常に主記憶装置にいて動いているので、常駐プログラムと呼ばれる。

特に入力データ（キーストローク信号）を取り込んで適当な処理を行ってから OS に送る常駐プログラムを昔は front end processor（前処理）と呼んだ。今日のアルファベット入力かな表示と引き続く変換キー入力によるかな漢字変換はこの前処理によって行われる。私の知る限り、1980年代に VJE と呼ばれるソフトウェアがこの方式の最初の着想であり、今日商業的に成功している全てのソフトウェアはもっとも本質的な部分はそのまねに過ぎない<sup>124</sup>。

文字化け。 計算機にとって文字データが約束に過ぎないことは、「文字化け」という現象で知ることができる。一つの計算機を使っているときは意識しないが、UNIX という OS が動いている計算機（workstation）のデータをそのまま（network であつかも一つの directory のようにつないで）MSDOS や WINDOWS95 が動いている計算機（パソコン）に持ってくると、日本語が文字化けする（変な記号列になって読めない）。変換ソフト（例えば nkf, ekc）が必要。

メールで送受信するときや telnet で他の計算機に入って使うときも、通常は計算機の OS や mail プログラムが自動的に変換するが、設定すを間違えると文字化けする。Netscapeなどで web site を見て、日本語の文字化けが（殆ど）ないのは、netscape が読み込んだデータ（html file）から日本語コード体系を自動的に検出して翻訳しているから。Floppy disk で別の計算機に持っていても必ず読めるのは、その disk が読める（format が同じ）ような計算機は同じ OS（または互換な OS）で動いている計算機なので、文字コード体系も同じだからである。

漢字と 2Byte01 列数字との対応の規約は最初 JIS 漢字として定められたが、これは KI/KO (Kanji In/Kanji Out) と言って、1Byte 文字から 2Byte 文字に変わる度に決まった制御記号（今日では複数種類ある）を入れ、1Byte 文字に戻る時も別の制御記号を入れる方式である（間は 2Byte で変換すると約束する）。殆ど英文、または殆ど日本文、のテキストならば問題ないが、全角と半角が混ざった文書では不経済である。そこで MSDOS を日本語化するとき shift-JIS という別の約束を導入した。これはテキストに 1Byte 文字として未使用の記号を 1Byte 目に用い、これらが出てきたときは 2Byte 文字として処理する、という約束である。これ以外に少なくとも EUC という規約があるが、その定義は知らない。

文字化けは送信側と受信側の文字コード体系の違いや、通信時に誤りが起きて制御記号などが届かなかったときに起きる。今日のソフトウェアは文字コードを自動的に検出するし、通信も誤り検出を行うので、設定を誤ったときというのが一番多い。しかし、いろいろな機械で入手したテキストを切り張りして文書を作ると、知らないうちに文字コードが混ざったり制御文字を部分的に取り込む場合があり、それをメールすると受信側の自動検出機能が弱いと途中から文字化けすることもあるように思う。

## §12.2. 文字統計

### §12.2.1. 文字統計プログラム

問。 文書をテキストファイルから読み込んで、現れる文字種の度数を測るアルゴリズムを以下をヒントにして作り、それに基づいてプログラムを作れ。

ヒント（仕様書）。 私が大学時代に自作したプログラムは、2文字接続の統計まで含んで、コメントとインターフェースを除けば FORTRAN 500 行以内のプログラム。アルゴリズムとしては単純。プログラムの注意点は入力の整備と結果のソート。

<sup>123</sup>OS プログラム開始時のこと。OS はいつも動いている唯一のプログラムなのでユーザーから見るとプログラムという意識が少ない。そこで OS プログラムを実行すると言わず OS を立ち上げるなどということが多い。

<sup>124</sup>哲弥流。今後は文法解析による長文一括変換などの重要な課題で獨創性を発揮しなければならない。

- (i) テキストファイルといっても驚くほどいろいろな記号が出てくる．長いテキストだと制御文字などの誤ったデータが紛れ込んでいる可能性もある．数えやすくするために入力データファイルを整形するプログラムを先に走らせる．
- (a) 意図的な改行（段落替え）は紙面上は行末までの空白として現れ、意味があるが、面倒な割に文字統計の意義は薄いので、改行を除去する．改行を除去する以上は空白記号を数えるのは意味がないので空白文字 (20H,8140H) も除去する．
  - (b) 文字にならない制御記号 (00H - 1FH) を除去する．
  - (c) 1Byte文字を対応する 2Byte文字に変換して、全て 2Byte単位にすることでプログラムの無用の煩雑を避ける．
- (ii) 文字列の統計をとるので各文字に配列の要素が対応する必要がある．配列要素は整数なので文字を整数に変換する手続きが必要．最も簡単なのは、文字列の内部表現が 01 列なのでそのまま 2 進整数とすることである（前処理で 2Byte 固定長になっていることに注意）．文字列と 2 進数の変換規則は shift-JIS のコード表から分かる．2 進数を変数に記憶するときに 1Byte 単位に切り放して順序を入れ換えることがある．これはコンパイラの仕様<sup>125</sup>（文字列を変数に代入したときの 01 列としての表現など）としてコンパイラのユーザーガイドを見たり、テストプログラムで文字を変数に代入した後で整数とみなして出力させれば分かる．
- (iii) 漢字を含めると可能な記号は多いので、全て配列要素として用意しておくとは大きくなりすぎる．出てくる毎にリストに追加して統計を取る．リストは上述のように shift-JIS コードでソートしておくとも統計を取るのに便利．各時点で配列の  $i$  番目の要素がどういうコードで、いくつ出てきたか、という二つの配列を用意する．
- (iv) 二文字連接（隣り合う文字のつながり方）の統計も殆ど同様にできる．
- (v) 結果は、出現回数で多い順にソートして表示する．各回数毎にその回数出てきた文字種類を表示させるのが見やすい．
- (vi) 出現回数の少ない文字種類は多いので書ききれない (overflow) ことを想定して、ある程度以上の文字種類が出てきたらあふれ処理をすること．

### §12.2.2. 情報量

情報量． 文章に出てくる記号の度数を測ると全ての記号が同じ割合で出てくるのではないことがすぐ分かる．一人の人（または一つのシステム）が書く（出力する）文書は大体ある一定の割合で各記号を用いる．これを、逆にとらえて、システムは固有の記号の出力割合（確率）を持っていると考えるのが普通である<sup>126</sup>．

仮に文字が  $N$  種類あった（平仮名なら 50 種ほど）として、長さ  $M$  の文書は  $N^M$  通り可能である（意味のない「文書」が殆どだが、可能性を議論している．）この中には aaaaaaaaaaaaaa·a のように極端に偏った文字の使い方をした文書もある．

（ある程度以上長い文書は）各文字の出現頻度が決まっている場合を考える． $i = 1, 2, \dots, N$  に対して文字  $i$  が頻度（確率） $p(i)$  で現れるような長さ  $M$  の文書は、 $i$  が  $M p(i)$  回出ているはずだから、多項定理より可能な文書の種類を  $2^{MI(p)}$  とおく<sup>127</sup>と

$$2^{MI(p)} = \frac{M!}{(Mp(1))!(Mp(2))! \cdots (Mp(N))!}$$

<sup>125</sup> 哲弥流．口言葉では「慣習とくせ」と同義．ずっと昔から文字列も数字も順序を入れ換えて変数に記録することになっている．MSB (most significant bit) といった単語で順序を指定する．変な風習だ．なぜなのか私は知らない．

<sup>126</sup> 哲弥流．言語を一つ決めるとその語彙や文法規則から記号の出現頻度が制約され、書き手のくせを加味するとほぼ頻度が決まるからだ、と考えるのだと思うが、厳密な証明があるとは思えない．しかし、出力確率が決まっていると考えると以下のように非常に面白い研究が可能になるので、既に 20 世紀初頭には通信における情報量の理論が確立した．暗号解読可能性の基礎理論でもあるので、競争や外交と絡んで研究に予算が投じられたことも早く進歩したと無縁ではあるまい．

<sup>127</sup> つまり、可能な文書の総数の 2 を底とする対数を、文書の長さで割ったものを  $I(p)$  とおくのだが、なぜそうおくかはすぐ明らかになる．



である．Stirling の公式

$$\lim_{M \rightarrow \infty} \frac{M!}{M^M e^{-M} \sqrt{2\pi M}} = 1$$

と  $\sum_{i=1}^N p(i) = 1$  より<sup>128</sup>，

$$\begin{aligned} \lim_{M \rightarrow \infty} I(p) \log 2 &= \lim_{M \rightarrow \infty} \left( - \sum_{i=1}^N p(i) \log p(i) - \frac{1}{2M} \left( (N-1) \log(2\pi M) + \sum_{i=1}^N \log p(i) \right) \right) \\ &= - \sum_{i=1}^N p(i) \log p(i) . \end{aligned}$$

即ち文書が十分長ければ（数学的に言えば，文書の長さが無限大の極限で  $I(p)$  は収束して）可能な文書の総数は  $2^{MI(p)}$  の速さで  $M$  とともに指数関数的に増大する．

$$(12.1) I(p) = - \sum_{i=1}^N p(i) \log_2 p(i)$$

を  $(p = (p(1), \dots, p(N)))$  で指定される送信装置の) 情報量 (entropy) と呼ぶ．

命題 1.  $p = (p(1), p(2), \dots, p(N))$  は  $N$  個の非負実数の組で  $\sum_{i=1}^N p(i) = 1$  を満たすとする．このときエントロピー (12.1) は以下を満たす．

$$(i) 0 \leq I(p) \leq \log_2 N .$$

(ii) 等号は  $I(p) = 0$  となるのは，どれか一つの  $i$  が  $p(i) = 1$  になるときで， $I(p) = \log_2 N$  となるのは，全ての  $i$  について  $p(i) = 1/N$  となるとき．

(iii)  $I$  は「上に凸」である．即ち， $I(\frac{1}{2}(p+q)) \geq \frac{1}{2}(I(p) + I(q))$ ．ここで  $\frac{1}{2}(p+q)(i) = \frac{1}{2}(p(i) + q(i))$ ．等号は  $p = q$  のときのみ．

問． 命題 1 を証明せよ（やさしい微分の問題）．

冗長度．  $R(p) = 1 - (2^{I(p)}/N)$  を 冗長度 (redundancy) と呼ぶ． $0 \leq R(p) \leq 1$  であって，命題 1 より，冗長度ゼロ ( $R(p) = 0$ ) となるのは全ての記号が等確率に現れる場合のみである．全ての  $i$  について  $p(i) > 0$  を仮定すれば  $R(p) > 0$  である．

定義より  $2^{I(p)}$  は一文字あたりの可能な文書の総数と見ることができる．本来「一文字あたり可能な文書の総数」とは記号の種類のことのはずだが，命題 1 より， $p(i) = 1/N, i = 1, 2, \dots, N$ ，でなければ  $2^{I(p)} < N$  となって，記号の数が実際より少ないように見える．即ち，全ての記号が等確率に出ない送信装置は，十分長い時間送信を続けると，實際上使っている記号が少ないように見える．このことは単なる説明のための説明ではなく，以下に示すように実際的な意味がある．

通信や計算機の内部表現では， $N = 2$  つまり 01 列である．これは「文字統計」とは信号の見方が変わるが，通信の理論では重要になる．このとき， $0 \leq I(p) \leq 1$  となる． $I(p)$  の単位をビットと呼ぶのは自然である．

<sup>128</sup>  $M$  が十分大きくて  $Mp(i) \gg 1$  が全ての  $i$  について成り立っているとす． $p(i) = 0$  なる文字  $i$  は出てこないということだから，最初から除外しておく．

圧縮． 次のことが知られている．

命題 2． 通信の一文字あたりの情報量が  $I$  ならば，任意の  $\epsilon > 0$  に対して  $N$  種類の文字を使用して平均的な文書の長さを漸近的に元の文書の  $\frac{I}{\log_2 M} + \epsilon$  倍にするような可逆な変換規則（翻訳）が存在する．

元の通信も変換後と同じ  $N$  種類の文字を用いる場合，一般には  $2^{I(p)} < N$  だったので， $\frac{I}{\log_2 M} < 1$  となる．可逆とは，変換後の記号列から元の文書を再現できるということである．

文書は 冗長さの分だけ 圧縮プログラムによって短くできる．

このような規則をプログラムにして自動的に圧縮するソフトを作ることが可能である．実際に，圧縮の手續きはいろいろ提案されていて，Lha, compress などの非常に多くの圧縮ソフトがある．記憶装置の少ないパソコンに大量の文書を蓄える場合や，遅い通信線を通して大きな文書を送る場合に非常に有効である．自動的に圧縮して送受する通信ソフトもある．

例． 命題 2 の証明は省略するが，実際に圧縮が可能であることを簡単な例で示そう． $N = 2$  で 0 が 0.25, 1 が 0.75 文書に含まれるとする．例えば元の文書が，

01011110111011011111011111001111

であったとする ( $M = 32$ )． $11 \rightarrow 1, 10 \rightarrow 01, 01 \rightarrow 001, 00 \rightarrow 000$ , という変換で，

0010011011011001110011100011

となり，28字に減少する．

問． 逆変換で元に戻る（復号）をたしかめよ．

誤り訂正． 通信における誤りは主に交換機などの通信の「つなぎ目」におけるノイズの発生による．ノイズは通信信号に横から信号が割り込んできて加算されるとみなす．殆ど場合は誤りがないから<sup>129</sup>，ノイズは殆ど 0 という記号からなり，ときどき 0 以外（2進数で考えれば 1）が入っているような記号列とみなせる．その情報量  $I_N$  は一般には小さい．例えば  $p(1) = 10^{-6}$ ,  $p(0) = 1 - 10^{-6}$  ならば  $I(p) \approx 2 \times 10^{-5}$ ．

次のことが知られている．

命題 3．  $N$  文字の通信で一文字あたりの情報量が  $I$  の通信がノイズの情報量  $I_N$  の通信線を通るとき， $I + I_N < \log_2 N$  ならばノイズによる誤りを（平均的には）訂正することができる．

この命題の正確な意味と証明は省略する．ノイズの混入を暗号化とみなすことで次の暗号解読の実例が参考になるはずである．

直感的には次のことを指す．例えば「今日は天気び良い」という文は「び」が「が」の誤りである（多分），と容易に分かる．わずが 8文字で 1文字の誤りの訂正が可能であることを表す．もちろん，訂正不可能な誤りもたくさんある．主張は，文章が長くなると訂正不可能な誤りの比率がゼロに近づく（訂正可能な誤りのほうが多い），という意味である．

冗長度がゼロでないことがこれを可能にする．人間は間違いやすいので，人間の使う言語は元々冗長度が非常に高くないと実用にならない．

但し，この誤り訂正の理論はこのままでは実際の通信で使うのは難しい．実際の通信では次々と信号が来て，それを素早く表示したり保存したりする必要があるので，十分時間をかければ訂正できる，という理論では実用にはならないのである（命題 3 は誤りを訂正するのにどれくらい先まで文書を読まないといけなしか触れていないことに注意）．

<sup>129</sup> 殆ど誤りならば通信できない．

暗号解読． 例えばアルファベット小文字と空白で書かれた文書を次のような方法で暗号化しよう（簡単のため句読点などは省略する）．

- (i) 自然数  $n$  を決める．
- (ii) abc... を順に 123... と番号づける．スペースは 0 とする．
- (iii) 文書の各文字（スペースを含む）毎に以下を行う．
  - (a) アルファベットの対応する番号に  $n$  を加えて 27 を法としていくらと合同になるかを求める．
  - (b) 対応するアルファベットを暗号とする．

例えば ‘zyab’ を  $n = 2$  として暗号化すると ‘a cd’ となる．

この方法による暗号であることが分かっている場合、これを解読するのはたいへん容易である．例えば、暗号文が ‘apq hq hihxmv’ だったとしよう．これ自体は明らかに英語でない．各アルファベットに  $n = 1$  を加えた ‘bqrairaijiynw’ も英語でない．これを 27通り全ての  $n$  について試すと

```
apq hq hihxmv
bqrairaijiynw
crsbjsbjkzox
dstcktcklk py
etudludlmlaqz
fuvemvemnbr
gvwfnwfnoncsa
hwxgoxgopodtb
ixyhpyhpqpeuc
jyziqziqrqfvd
kz jr jrsergwe
l aksakstshxf
mabltbltutiyg
nbcumcmuvujzh
ocdnvsnvwvk i
pdeoweowxwla j
qefpxfpxyxmbk
rfgqygqyzyncl
sghrzhrz zodm
this is a pen
uijtajtabaqfo
vjkubkubcbrgp
wklvclvcdshq
xlmwmdwdedtir
ymnxenxefeuj s
znoyfoyfghvkt
opzgpzghgwlu
```

となるが、既に最初の 5 文字目までで、英語になる可能性があるのは ‘this is a pen’ だけだと決まる．

暗号化はノイズの混入とみなすことができるので、命題 3 から次のことが言える．

命題 4．  $N$  文字で書かれた一文字あたりの情報量が  $I$  の文書がある．一つの文書を翻訳したときに可能な暗号文が、文書の長さ  $M$  とともに増える速さが  $(N/2^I)^M$  より小さいならば、十分長い文書を与えられれば暗号は解読できる．

命題4の正確な意味も証明も省略する。

例は  $N = 27$  の場合で、可能な暗号文は文書の長さに関係なく 27 しかないので、 $(N/2^I)^M > 27$  を解くと  $M > \log_2 27 / (\log_2 27 - I)$  となって、これが成り立てば平均的に解けてしまう。 $I$  は英文の 1 文字あたりの情報量である。 $I = 4$  位だと、6 文字くらいで解けてしまう（正確な数字が今手元にないが、空白を入れた英文の情報量はかなり小さいはず。）

軍事や外交上の暗号に対する暗号解読は受信時の誤り訂正と異なってある程度時間をかけられるし、軍事費用で大きな計算機を使用することもできるので、命題4は実用上(?)の意味がある。これに対して最近、平和時の商用などの暗号が、インターネットを利用した商売（ホームショッピング、ホームバンキング）が現実化するとともに、盛んに研究されている。商売の場合は、他人の暗号を解読する（産業スパイや反国家的取引を取り締まる政府・警察）立場に立つと、安価に高速に解読できないと意味がないので、命題4（原理的解読可能性）だけでは実用にならない。実際、今日のネットワーク安全上の主要な暗号方式である公開鍵暗号は、原理的には解読可能であるが、解くまでに極めて長い時間を要する（と思われる）ような整数論の問題を利用している。

## § ネットワーク

私が計算機を勉強した頃はネットワークは日本ではできたての頃で、本当のエキスパート以外は何も知らない時代だったが、今日ではアプリケーションソフトでも、メール、web browser、ftp用ソフトウェア、など、ネットワークがなければ意味のないものはいくらでもある。殆どの場合、研究者ですら計算機が一番の使い道はメールによる連絡ではないかと思われる。

本講義ではとてもここまで用意できない（1年の講義で初歩からここまで来るのは多分不可能）が、ネットワークはこれから重要になっていくので各自で勉強していただきたい（[7, p.204-300]）。

なお、§12.2.2 の情報量の理論は基礎的な通信の可能性と限界に関する理論としてネットワークの基礎数学の一つである。圧縮は特に画像通信などで非常に重要であり、暗号化はクレジット番号や電子マネーなどの個人情報の保護の点から商業的にも重視されている。

今日のネットワーク技術の基本的な思想については日本のインターネットの基礎技術を育てた最大の貢献者の一人による初心者向けの解説書（[10]）を是非読んでいただきたい。

問.

- (i) 圧縮はなぜ画像の通信で（文書の通信よりもさらにいっそう）重要になるか？
- (ii) インターネットプロバイダに加入したい。いくつかの候補（プロバイダ会社）があるときどのような点を比較すべきかを考えよ。

## 参考文献

- [1] 岩波理化学辞典，第4版，1987年，岩波書店（もっと適切な辞書があるはずだが，手元にこれしかなかったの。）
- [2] 広辞苑，第4版，1991年，岩波書店．
- [3] 漢和中辞典，第9 2版，1968年，角川書店．
- [4] 河西朝雄，C言語によるはじめてのアルゴリズム入門，1992年，技術評論社．
- [5] 木田祐司，UBASIC 86，日本評論社．
- [6] D. E. Knuth, *Art of Computer Programming: Sorting and Searching*, Addison & Wesley, 1973. (あいにく手元になかったので参考にできなかったが，このシリーズはバイブルと呼ばれるプロ必携の教科書．ソートはその4分冊目だったと思う．)
- [7] 広松恒彦，情報処理技術者試験 [新2種重点ガイド 1] ハードウェア，増補版，1996年，日本経済新聞社．
- [8] 広松恒彦，情報処理技術者試験 [新2種重点ガイド 2] ソフトウェア，増補版，1996年，日本経済新聞社．
- [9] 広松恒彦，情報処理技術者試験 [新2種重点ガイド 1] アルゴリズム，増補版，1996年，日本経済新聞社．
- [10] 村井純，インターネット，1995年，岩波新書(416)．
- [11] 涌井良幸・涌井貞美，わかる超高速ソートプログラミング，1986年，誠文堂新光社．
- [12] ???，データ構造とアルゴリズムシリーズ3，ソート・検索，1995年，CQ出版社（本屋で見かけたが，著者名だけメモし忘れた．知っている人は教えて下さい．)
- [13] Oh! PC, 1997年3月号.

シラバス

計算機演習 I II III IV (CA103–CA106) 数学科 2 年 選択 2 コマ通年 (講義と実習) 1 × 4 単位 . 担当 : 服部哲弥 ( 6 号館 2 階研究室 ) .

広く浅い知識の伝授 , 各自が勉強を始めるきっかけ . 第 2 種情報処理技術者試験のレベル . 計算機を使ってみること .

成績評価 : 演習 , 小テスト , 課題製作等 .