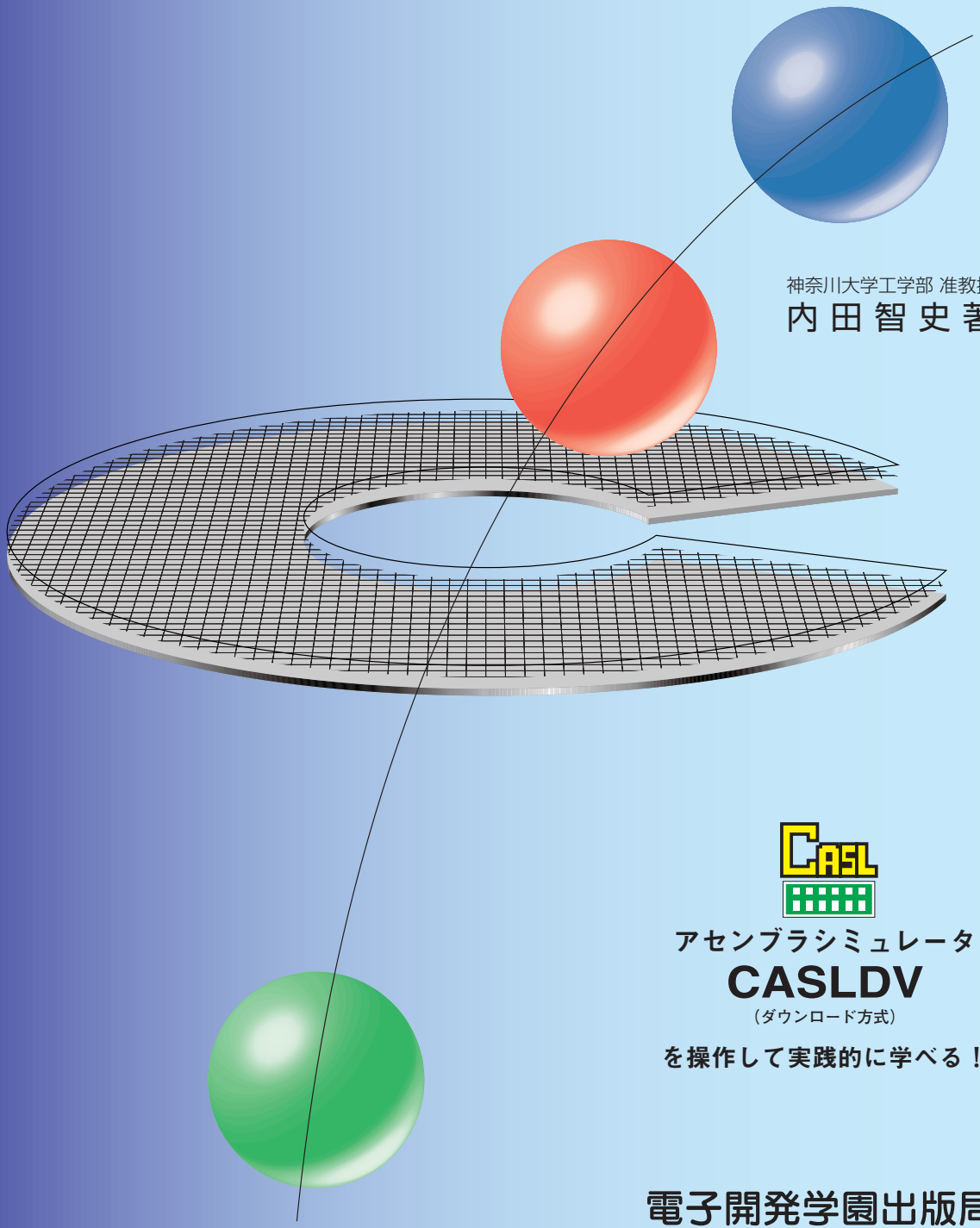
アセンブラ入門
CASLⅡ 第3版



神奈川大学工学部 准教授
内田智史 著

アセンブラシミュレータ
CASLDV
（ダウンロード方式）
を操作して実践的に学べる！

電子開発学園出版局

アセンブラ入門
CASLⅡ 第3版

神奈川大学工学部 准教授　内田智史 著

電子開発学園出版局

序文

COMET II は、経済産業省の情報処理技術者試験の基本情報技術者試験におけるアセンブラ言語の問題を出題するために設計されたコンピュータです。試験のために設計されているので、実在するコンピュータに比べてとても単純な構成になっています。COMET II のハードウェアは、実際には誰も作ったことがないので、仮想的なコンピュータと呼ばれることもあります。CASL II は、COMET II に対するアセンブラ言語です。アセンブラ言語とは、そのコンピュータの機械語に 1 対 1 に対応して作られた言語です。

情報処理技術者試験は、昭和 44 年 (1969 年) に試験制度が発足しました。これにより、第一種情報処理技術者認定試験と第二種情報処理技術者認定試験が実施されました。COMET・CASL は、第一種情報処理技術者認定試験では必須項目、第二種情報処理技術者認定試験では選択項目となっていました。平成 6 年 (1994 年) に、新しい試験制度へ移行しましたが、第一種情報処理技術者認定試験と第二種情報処理技術者認定試験はそのまま残りました。しかし、この時の改訂で、第一種情報処理技術者試験からプログラミング能力を問う問題がなくなり、実質的に COMET・CASL は第二種情報処理技術者試験のみで出題されるようになりました。

平成 13 年 (2001 年) に、さらに試験制度が変更になり、第一種情報処理技術者認定試験がソフトウェア開発技術者試験に、第二種情報処理技術者認定試験が基本情報技術者試験に代わり、COMET・CASL も新しく改訂され、COMET II・CASL II という名称になりました。本書では、この COMET II・CASL II を対象としています。

本書は、CASL II を学習するための書籍であり、2001 年に第 1 版が発行され、本書は第 3 版となります。第 3 版の改訂にあたり、本書を LaTeX で組み直しました。2 色刷りにして、多くの図表を改訂し、読みにくいところ、分かりにくい所を手直しして、読みやすさについては、かなり改善されたと思います。

なお、本書を書く際にあたっては、最初から読み進んで一度読めば、必ず分かるように心がけました。しかし、プログラミングの入門書である以上、CASL II のリファレンスマニュアル的な部分も必要なため、COMET II の機械語の詳細な説明を第 6 章にまとめてあります。この部分は、プログラミングの概念と密接に絡むため、最初に読んだ段階では、ピンと来ない部分も多いと思います。したがって、第 6 章は、最初は、「流し読み」をして、第 7 章以降を読みながら必要に応じて「読み戻る」というのがよろしいかと思います。

また、このようなプログラミング言語の習得では、実際に、プログラムを作成してコンピュータにかけて実行するという訓練が必要になってきます。そこで、本書では、Windows(Windows 7、Windows 8 など) をお持ちの読者に対しては、筆者が作成した CASL II の開発環境 CASLDV をインターネットからダウンロードできるようにして、読者が学習しやすいように配慮しました。CASLDV は、エディタやデバッガを含む本格的なもので、読者が作成した CASL II のプログラムが COMET II 上で動作する様子をグラフィックスでよく分かるようにしました。また、本書に示したサンプルプログラムもインターネットからダウンロードできるようにしてありますので、本書を学習しながら、ぜひ、サンプルプログラムも実行させてみてください。CASLDV およびサンプルプログラムをダウンロードするには、筆者の e-Learning 用の Web ページ (`http://www.officeuchida.com/CASL/`) をご覧下さい。

なお、Unix 版のシミュレータ (ただし、コマンドライン版のみ) をご希望の方も、前記の Web ページをご覧下さい。

また、CASLDV は単にプログラムを組むことを支援するだけでなく、COMET II の機械語命令の動作の様子を CPU や主記憶装置との関係とともにグラフィカルに示してくれるので、初心者が機械語の動作のイメージを学習するにはたいへん適していると考えています。初心者の方には、CASLDV のこの表示機能を活用していただき、アセンブラ言語の理解を深めていただきたいと思います。

CASLDV のマニュアル (PDF ファイル、約 50 ページ) もまた、前記の Web ページからダウンロードすることができます。

本書には、演習問題、総合問題が記載されていますが、この問題の解説・解答は、前記の Web ページから PDF ファイルとしてダウンロードできるようにしてあります。

アセンブラ言語を勉強すると、他のプログラミング言語、たとえば C 言語や Java などの理解が、さらに一歩深まります。その意味では、多くの方に CASL II を学習していただきたいのですが、実際問題としては、試験対策のために本書を読まれる方も多いと思われます。そこで、本書の中にはできるだけ多くの演習問題を取り入れました。第 9 章までは比較的簡単な演習問題を取り入れてあります。これらを解いて、問題を解くコツをつかんでください。第 10 章からは、実際の試験問題に近い本格的な問題を数多く取り入れてあります。できるだけ、独力でこれらの問題をたくさん解いて力をつけてください。第 13 章には、過去の実際の問題を解く方法について解説してあります。ただし、毎年の情報処理技術者試験問題の解説は、その都度書籍の中に含めるわけにはいきませんので、これらの情報についても、前記 Web ページからダウンロードできるようにしてあります。

第 1 版から第 3 版を執筆するにあたり、SCC 出版局の方々には、遅れがちな原稿を辛抱強く待っていただき、本当に感謝しております。また、本書の作成に携わった多くの方々に感謝いたします。特に本書の初版の草稿をお読みいただき、さまざまなご指摘をいただいた電子開発学園の先生方に感謝致します。さらに、本書で使用する CASL II 開発環境 CASLDV の動作確認などをしていただいた方々に感謝したいと思います。

2012 年 7 月
内田智史

目 次

Coffee Break

アセンブラ入門
CASLⅡ 第3版

第1章　COMET IIの概要とその位置付け

1.1　COMET IIとCASL IIの概要

1.1.1　はじめに

これから私たちは、**COMET II**(コメットツー) というコンピュータを動かすための **CASL II**(キャッスルツー) という言語を学習していきます。そのためには、多少なりとも、コンピュータについての理解が必要になります。コンピュータというと今では誰もがパーソナルコンピュータを思い浮かべるでしょう。でも、パーソナルコンピュータとCOMET IIの間には、とてつもなく大きなギャップが存在します。このギャップは、コンピュータのことを良く知らないとなかなか理解できません。そこで、まず最初に、コンピュータがどのように発明され、発展していったのかについて簡単に説明したいと思います。この説明の中で、コンピュータの動作原理、機械語やアセンブラの役割について説明します。

なお、COMET IIは、情報処理技術者試験のために考え出されたコンピュータであり、現実には存在しません。COMET IIの詳細な定義は、情報処理技術者試験案内の中にある「仕様」に記載されています。なお、この「仕様」は本書の最後にも記載されていますが、情報処理技術者試験の問題の中にも記載されています。

1.1.2　初期のコンピュータ

計算を補助する装置として、最近ではあまり見かけませんが、算盤 (そろばん) があります。このような計算に関する装置は、算盤を始め、歯車を用いた機械的な装置が数多く発明されています。しかし、コンピュータは電子の力を利用しています。電子式の最初のコンピュータは、1937年から1941年頃に開発されたジョン・V・アタナソフとクリフォード・ベリーのアタナソフ・ベリー・コンピュータであると言われています (米国アイオワ州立大学) [1]。これは、線形連立方程式を解くためのコンピュータで、2進法の採用や論理演算回路という現在のコンピュータでは常識となっている基本原理を確立しました。そののち、1942年から1946年にかけて、非常に有名な **ENIAC** (Electronic Numerica1 Integrator And Computer、図1.1(2ページ) 参照) という世界で最初に実用化されたコンピュータを、米国ペンシルベニア大学ムーア電気工学科のジョン・モークリーとプレスパー・エッカートが開発し、それ以降の輝かしいコンピュータの発展が始まったのです。ENIACは18,000本の真空管[2]を使ったもので、30トンの重量があり、メンテナンスが非常に大変でした。また、ENIACに計算を行わせるためには、その手順をENIACに指示しなければなりません。この指示は、パッチボードと呼ばれる特殊な配線盤

[1] アタナソフについては、クラーク・R・モレンホフ著、最相力/松本泰男共訳、ENIAC神話の崩れた日、工業調査会、に詳しく載っています。

[2] 真空管については、3ページの *Coffee Break 1-1*(3ページ) を参照してください。

の上に配線を行うことによって行っていました。計算手順を変更するには、パッチボードの配線を変更しなければならず、大変な作業だったようです。

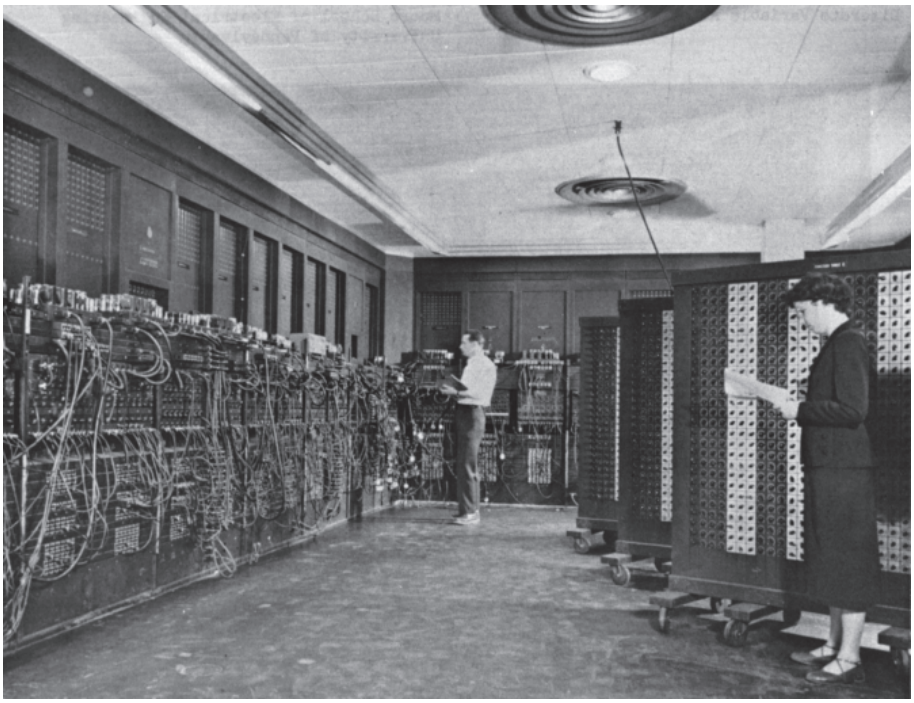


図 1.1: ENIAC：世界最初の実用的なコンピュータ
18,000 本の真空管が必要な ENIAC は、現在のコンピュータに比べると非常に大きなものでした

1.1.3　プログラムの登場とプログラム内蔵方式

そのころ、フォン・ノイマンという学者が、コンピュータに対して計算手順を示すには、ENIAC のように配線に頼るのではなくて、ちょうど我々が文章を書くように、計算手順を示した命令書を作成して、それをコンピュータの主記憶装置の中に計算するデータとともに格納し、コンピュータは、その命令書に書かれた命令文を主記憶装置から逐一取り出して、それを実行するべきであるという提案を行いました。この計算手順を示した命令書のことを**プログラム** (program) と呼びます。フォン・ノイマンのこの考え方は、プログラムとデータを区別なく主記憶装置に格納するという点が優れており、**プログラム内蔵方式**と呼ばれ、現在のほとんどすべてのコンピュータが動作する基本原理になっています。

プログラム内蔵方式を初めてコンピュータに取り入れたのは、イギリス・ケンブリッジ大学のモーリス・ウィルスによる **EDSAC** (Electronic Delay Storage Automatic Ca1culator、1949 年稼動) や、米国ペンシルベニア大学の **EDVAC** (Electronic Discrete Variable Automatic Calculator、ENIAC の後継機、1951 年) というコンピュータです。その後、商用のためのコンピュータ **Univac1** がレミントン・ランド社 (後にスペリー・ランド社、さらに現在はユニシス) によって開発され 1951 年にその一号機が設置されました。Univac1 は、真空管を 5600 本使い、計算は 2 進法 (第 2 章参照) ではなく 10 進法で 12 桁まで計算できました[3]。さらに、1952 年に

[3]コンピュータにおける 2 進法と 10 進法の違いについては *Coffee Break 3-4*(43 ページ) を参照してください。なお、実際には ENIAC も計算方法は 10 進法を採用していました。

IBM が **IBM701** を開発し、これ以降コンピュータの商用利用が爆発的に増大しました。

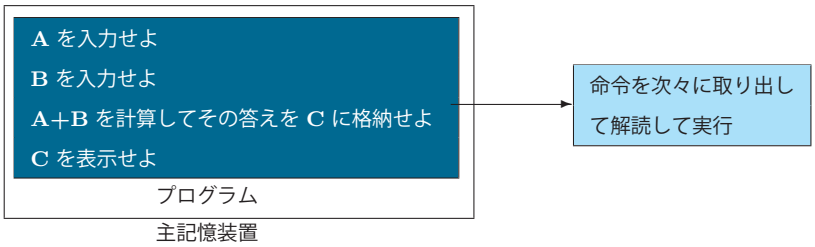


図 1.2: プログラム内臓方式

1.1.4 機械語の意味

EDSAC や EDVAC は、基本的には現在のコンピュータとほぼ同じ原理で動作します。コンピュータを実行させるには、**機械語** (machine language) と呼ばれる言語で書かれた命令書を用意します。これがプログラムと呼ばれるものです。コンピュータは、プログラムの中に書かれた機械語を淡々と実行する単なる機械なのです。

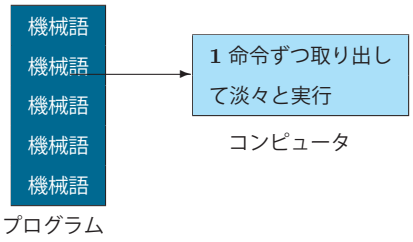


図 1.3: 機械語とその実行

さて、機械語は、コンピュータごとに異なっています。あるコンピュータのために書かれた機械語は、種類の違う別のコンピュータでは動作しません。EDVAC の機械語は、EDVAC というコンピュータのために用意された機械語です。EDVAC の機械語のプログラムは、別のコンピュータ上では動作しません。これと同様に、COMET II の機械語は、COMET II というコンピュータのために用意された機械語です。COMET II の機械語は、別のコンピュータ上ではそのままでは動作しません。

Coffee Break 1-1 ： 真空管

真空管は、トランジスタが発明されるまで、ラジオなどの装置を構成する部品として使われていたものです。ガラスで覆われた筒丈の中に配置された複数の電極があり、さまざまな処理を行うことができました。ENIAC などの初期のコンピュータは、論理演算回路を実現するために、この真空管を大量に使っていました。しかし、真空管の寿命はそれほど長くないので、安定してそれらを稼働させることは大変だったようです。

1.1.5　アセンブラ言語の登場

機械語によるプログラム開発は人間にとってとても困難です。次に示すものは、COMET II の機械語[4]で書かれたプログラムですが、これが何をしているものか分かりますか。

```
0010010000010010
0010010100010011
0101000000010000
```

COMET II の機械語

コンピュータである COMET II は、これを正しく解釈し実行できます。しかし、人間にはそんなことはできません。そこで、この機械語を人間が直接記述するのではなく、この機械語を人間の分かる言葉に 1 対 1 に置き換えた言語が開発されました。これをアセンブラ言語[5]と呼びます。機械語とアセンブラ言語の関係を図 1.4 に示します。図 1.4 では、機械語の隣に同じ意味のアセンブラ言語を示し、さらにその隣に日本語による説明を示します。

COMET II の機械語	アセンブラ言語	日本語による説明
0010010000010010	ADDA GR1,GR2	GR1 に GR2 の値を加えよ
0010010100010011	SUBA GR1,GR3	GR1 から GR3 の値を引け
0101000000010000	SLA GR1,2	GR1 の値を 4 倍せよ

図 1.4: 機械語とアセンブラ言語の関係

図 1.4 では、まだ、GR1 や GR2 が何であるのかを説明していないので、「日本語による説明」の意味がよく分からないと思いますが、「加算して、減算して、4 倍している」ということくらいはわかるでしょう。また、アセンブラ言語の部分を読んでも多少は意味が分かるかと思います (しかし、そうは言っても ADD は「加える」、SUB は「引く」ということを意味しているというのは直ちに英語から分かりますが、SLA が 4 倍というのは CASL II アセンブラ言語を学習しないと分からないですね) [6]。前述のように、COMET II の機械語に対応して作られたアセンブラ言語を **CASL II** といいます。

アセンブラ言語の登場により、機械語で直接プログラムを作成しなくても良いので、人間にとっては大助かりです。しかし、コンピュータは、アセンブラ言語を直接理解することはもちろんできません。そこで、アセンブラ言語で書かれたプログラムを実行するために、そのプログラムをコンピュータの理解できる機械語に翻訳する必要があります。この翻訳を人間が行っていたのでは大変ですから、これもまたコンピュータに任せます。この翻訳作業はアセンブラというプログラムを用いて行います。アセンブラは、アセンブリ言語で書かれたプログラムを入力してそれを同じ意味を持つ機械語に変換する翻訳プログラムです。

図 1.5 に示すように、CASL II で書かれたプログラムを入力して、COMET II の機械語プログラムを出力するアセンブラを **CASL II アセンブラ**と呼びます。

---

[4]本書では、COMET II の機械語は情報処理技術者試験案内の中の参考の部分に示されているものに準拠しています。

[5]Assembly language が英語なので、本来はアセンブリ言語とすべきですが、JIS およびそれに準拠している情報処理技術者試験では、アセンブラ言語としているので、本書でもアセンブラ言語と表現します。

[6]SLA というのは、Shift Left Arithmetic という意味で、6.3.5 項 (2)(125 ページ) で学習します。

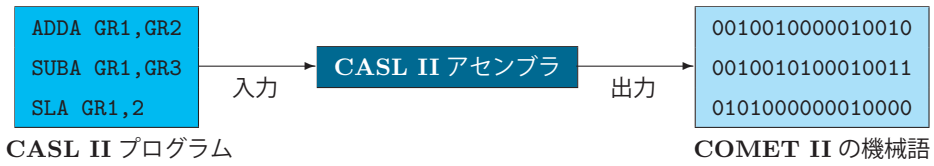


図 1.5: CASL II アセンブラとその役割

前述のように、機械語はコンピュータによって異なります。たとえば、現在のパーソナルコンピュータで主流の CPU の元祖である 8086 [7] 上のアセンブラ言語 (CP/M-86 [8]) の場合には、次のようになります。

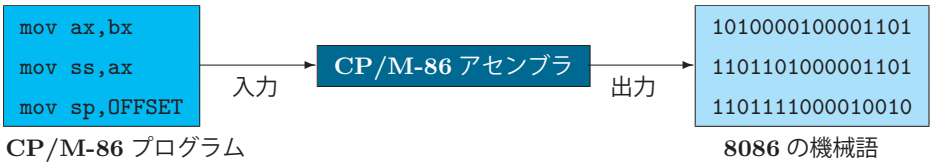


図 1.6: CP/M-86 アセンブラとその役割

CP/M-86 では、小文字の記述を許していますが、CASL II では小文字の記述が許されていないので、プログラムは全て大文字で書くことになっています。このように、コンピュータが異なれば、機械語が異なるので、それに応じてアセンブラ言語も異なります。

Coffee Break 1-2 ： CPU

CPU は、Central Processing Unit の略で**中央処理装置**と呼ばれます。CPU の構成については、第 3 章で学習します。

ここでは、CPU の構成素子とパソコン用の CPU、いわゆるマイクロプロセッサについて外観してみましょう。

コンピュータの初期の時代の CPU は真空管 (*Coffee Break 1-1*(3 ページ)) で作られており、非常に巨大でメンテナンスが大変なものでした。CPU を構成する素子は、真空管からトランジスタ、集積回路 (IC、LSI)、大規模集積回路と進化していきます。

次にマイクロプロセッサについて外観してみましょう。

世界最初のマイクロプロセッサは、インテル社製の 4004 です。これは、日本のビジコン社からの要請でインテル社が開発したものです。4004 の開発には、ビジコン社の社員 (当時) であった嶋正利が大きな役割を果たしました。嶋は、これ以降のマイクロプロセッサの開発に携わり、大きな貢献をしました。嶋は、これらの功績により、1997 年京都賞を受賞しています。

インテル社は 4004 の成功からマイクロプロセッサの可能性を認識し、ビジコン社から 4004 の販売権を買取り、その後、8080、8086、Pentiam などの高性能 CPU を次々と開発していき、パソコンの主流の CPU となっています。

[7] 8086 というのはインテル社製の CPU(1.3 節参照) の名前です。この CPU は、8086 → 80186 → 80286 → 80386 → 80486 → Pentium → ···→ Intel Core i3 と進化してきました。

[8] CP/M-86 とは、デジタルリサーチ社によって開発された 16 ビットコンピュータ用のオペレーティングシステムで、8 ビットコンピュータの世界で標準的に使われていた CP/M の後継版です。CP/M-86 は、16 ビット版の性能を活かすため、新しい機能を取り入れ、複数のプログラムを同時に実行するマルチタスク機能を導入したコンカレント CP/M-86 などがあり、かなりの評価を得ていました。しかし、CP/M との互換性が悪いことやライバルのマイクロソフト社製の MS-DOS に押され、CP/M のようには普及しませんでした。

1.1.6　高水準言語とコンパイラの置場

さて、話をコンピュータの歴史に戻しましょう。1950 年代も半ばになってくると、コンピュータの需要はますます増大し、プログラムを作成する効率が重視されるようになりました。1950 年代前半では科学技術系の研究者が、自分の研究のためにコンピュータを使うには、アセンブラ言語を学習し、さらには、複雑なコンピュータの動作原理を学習する必要があったのです。しかし、**FORTRAN** という言語とその言語によって書かれたプログラムを機械語へ変換する仕組みが 1950 年代中ごろに IBM のジョン・バッカスによって開発され、人間にとって、とても理解しやすい形式でプログラムを作成できるようになったのです。FORTRAN では、プログラムの中に数式を書けば、それがそのまま機械語に変換されます。人間の考え方にどのくらい近いのかという意味で当時は、アセンブラ言語のことを低水準言語、FORTRAN のような言語のことを高水準言語と呼んでいました (現在でも、このように呼ぶことがあります)。高水準言語を機械語に翻訳するプログラムのことをコンパイラと呼び、FORTRAN プログラムを対応するコンピュータの機械語に翻訳するプログラムのことを **FORTRAN** コンパイラと呼びます。FORTRAN コンパイラのイメージを図 1.7 に示します。

図 1.7: FORTRAN コンパイラとその役割

高水準言語は、その後、次々に開発され、COBOL(1960 年)、PL/I(1965 年)、C(1972 年) などが有名で情報処理技術者試験にも取り入れられていました (FORTRAN と PL/I は、現在では、試験からは除外されています)。高水準言語が、アセンブラ言語と異なる点はどこでしょうか。それは、アセンブラ言語が機械語と訪ね対応しているのに対して、高級言語は必ずしも 1 対 1 には対応していないという点が、まず挙げられます。基本的にアセンブラ言語では、機械語の命令一つ一つに対して、それに相当する命令があります。高級言語の場合、多くの場合、一つの命令が複数の機械語命令に変換されます。図 1.8 のプログラムは「1 から N までの和」を求める問題です。それには、次の公式を使えば良いでしょう。

$$S = 1 + 2 + 3 + \cdots + N = \sum_{k=1}^{N} k = \frac{N \times (N+1)}{2}$$

Coffee Break 1-3 ： パソコン上の実際のアセンブラ

パソコン上で圧倒的なシェアを誇るアセンブラ言語は、マイクロソフト社製の MASM でしょう。これは、現在マイクロソフト社から無償で提供されているアセンブラ言語です。

CASL II と C 言語で、このプログラムを作成すると、図 1.8 のようになります。

```
1  SUMUPN   START
2           LD     GR1,N      1 から N までの和を求める
3           LAD    GR7,1      GR7 ← 1
4           LD     GR2,GR1    GR2 ← GR1
5           ADDA   GR2,GR7    GR2 ← N+1
6           XOR    GR3,GR3    GR3 ← 0
7  LOOP     SUBA   GR1,GR7    GR1 のカウントダウン
8           JMI    EXIT       結果が負なら EXIT へジャンプ
9           ADDA   GR3,GR2    GR3 ← GR3+GR2
10          JUMP   LOOP       LOOP へ無条件でジャンプ
11 EXIT     SRA    GR3,1      GR3 ← GR3 ÷ 2
12          ST     GR3,S      S ← GR3
13          RET               おしまい
14 N        DC     100
15 S        DS     1
16          END
```

CASL II によるプログラム

```
#include<stdio.h>

int main(void)
{
        int s, n=100;

        s = n * ( n + 1 ) / 2;
        printf ("s=%d¥n", s);

        return 0;
}
```

C 言語によるプログラム

図 1.8: CASL II と C 言語のプログラム

C 言語によるプログラムでは、計算式が `s=n*(n+1)/2;` [9]と書かれているように 1 行ですんでいるのに対し、CASL II によるプログラムでは、その部分の計算式が 9 行にも及んでいます。つまり、C 言語で書かれたプログラムは、C コンパイラによって複数の機械語に展開されているのです。

1.1.7 最適化コンパイラの登場

FORTRAN は、1950 年代半ばに実用的なコンパイラが登場しました。当時の FORTRAN コンパイラの開発目標の一つが、生成された機械語プログラムの実行効率の良さでした。当時は、研究用のコンパイラは数多く開発されていました。しかし、当時は、コンパイラが生成する機械語は、優秀な人間が必死になって書いたプログラムに比べて、効率が格段に悪いので、FORTRAN のようなプログラミング言語は実用的ではないと考えられていました。そこで、FORTRAN コンパイラが生成する機械語は、人間が手で書いた機械語 (実際にはアセンブラ言語で書いたプログラム) と同等の実行効率を出せるように工夫されたのです。その後、さまざまな研究が行われ、その成果はコンパイラの最適化機能として集約され、現在では、その技術はかなりのレベルに達しています。**最適化機能**とは、プログラムを機械語に変換する際に、極力効率的になるように処理するだけでなく、場合によっては人間の書いたプログラムの実行手順まで変えてプログラムの実行効率をアップさせてしまう機能のことです。実際に、現在の Fortran [10]や C 言語コンパイラの最適化技術は非常に優れています。

[9] C 言語の数式では、乗算を*、除算を/、代入を=で表現します。
[10] FORTRAN は、その後数回の改定を経て、現在は 1995 年に JIS(日本工業規格) として定められた Fortran 95(最新版は Fortran 2008) になっています。

1.2 CASL IIアセンブラ言語を学習する価値

さて、それでは、私たちがCASL IIアセンブラ言語を学習する価値はいったいどこにあるのでしょうか。最適化コンパイラの機能が優れていて、人間がアセンブラ言語で書いたプログラムよりも優れた機械語を生成するのであれば、アセンブラ言語でプログラムを書く必要がないではないかと思う人も多いでしょう。筆者は次のように考えます。

(1) 現在でも、アセンブラ言語で書かざるを得ない部分がある

最適化コンパイラがいくら優れているといっても、何らかのプログラミング言語を使う以上、その言語の制約を受けます。たとえば、C言語を使う場合には、C言語の持つ機能に制約を受けます。C言語で文字データを扱う場合、そのデータの終わりに0を入れるという規則になっていますが、ある種のコンピュータはこの処理方法では実行速度が極端に遅くなってしまいます。アセンブラ言語は、このようなコンパイラ言語の制約に振り回されず、そのコンピュータ独特のチューニング[11]を行うことができるのです。実際、ある調査によると組込み用プログラム(家電製品や自動車のエンジンなどを制御するプログラム)の80 %はC言語ですが、そのうち、半数がアセンブラ言語を併用しています。また、10 %はアセンブラ言語のみによる開発です。

(2) 高水準言語を使う時に、その出力される機械語のパターンが連想できるようになる

高水準言語でプログラムを書く時に、そのプログラムが機械語に変換されるパターンが分かると、割と効率の良いプログラムが書けるようになります。つまり、高水準言語でプログラムを書く時の、知識の幅が広がるのです。それでは、次の疑問として、COMET IIのような現実には存在しないコンピュータではなく、現実のコンピュータの機械語あるいはアセンブラ言語を学習すれば良いではないかと思う人もいるかも知れません。COMET IIは、前述のように情報処理技術者試験のために作られた仮想的なコンピュータです。それも、現実のコンピュータと比べると極端に単純化されたコンピュータです。現在の代表的なコンピュータであるマイクロプロセッサとCOMET IIの比較を表1-1に示します[12]。これを見ると、いかにCOMET IIが単純化されたコンピュータであるかが理解できるでしょう。しかし、現実の複雑な機構を備えたコンピュータを初心者が学習することは、とても大変です。そこで、初心者学習用としてコンピュータのエッセンスを抜き出して作成されたのがCOMET IIなのです。それならば、こんな単純なものを理解しても何の勉強にもならないではないかと思う人もいるかも知れません。しかし、COMET IIは、実際のコンピュータの本質的な部分だけを抜き出しているので、基本的なコンピュータの理解は、COMET IIを学習するだけでも十分にできるのです。逆に現実的でやっかいな問題に振り回されずに、効果的な学習ができるという利点があります。また、COMET IIが仮想コンピュータであるもう一つの理由として、このコンピュータが情報処理技術者の試験のために作られたものであることが挙げられます。試験の公平性を保つためには、特定メーカのコンピュータを取りあげる訳にはいきません。つまり、一部の人が有利にならないようにしているのです。

---

[11] レーシングカーのエンジンをいろいろ設定を変えて、ある条件で最高の出力ができるように調整することをチューニングと言います。これと意味はまったく同じで、プログラムの実行順序やデータの配置をいろいろと変えて、実行するコンピュータのハードウェア特性に合わせて最高の性能を出せるように改造することをチューニングと言います。

[12] 表1-1の内容を見ても、そこに何が書いてあるかまったくわからない人も多いでしょう。ここで、大切なことは、COMET IIというのは、実在のコンピュータをその基本的なところだけを抜き出して作成されたコンピュータであるということを理解してもらうことだけなので、細かなことは分からなくても気にしないで下さい。

表 1.1: 代表的なマイクロプロセッサと COMET II との比較

		代表的なマイクロプロセッサ	COMET II
扱えるデータ		8,16,32,64 ビット	16 ビット
汎用レジスタの数		32 個（RISC の場合)	8 個
仮想記憶		有	無
メモリ保護		有	無
アドレス空間		有	無
命令セット	データ転送命令	○	○
	整数演算	○	△ (乗算・除算なし)
	論理演算	○	○
	浮動小数点演算	○	×
	10 進演算	○	×
	制御命令	○	○
	サブルーチン	○	○
	I/O (入出力)	○	△ (基本機能のみ)

1.3 コンピュータのハードウェア構成

一般にコンピュータのハードウェアは、入力装置、出力装置、演算装置、制御装置、主記憶装置から構成されています。これらの装置をまとめてコンピュータの**五大装置**と呼びます。この構成は、初期のコンピュータから現在のパソコンに至るまで変わっていません。

それでは、現在、最も多く使われているパソコンに対応して直感的に説明しましょう。**入力装置**はデータを入力するための装置で、キーボードやマウスが対応します。入力されたデータは主記憶装置に転送され、そこで記憶されます。**出力装置**は、主記憶装置内のデータを出力するための装置で、ディスプレイやプリンタが対応します。演算装置と制御装置は、**CPU** (Central Processing Unit) と呼ばれる超 LSI の中に格納されています。現在の多くのパソコンには、インテル社製の Pentium(ペンティアム) と呼ばれる CPU の後継 CPU が搭載されています。**演算装置**は、文字通り演算を行う回路です。COMET II では、整数の加算・減算・ビット操作の演算が用意されていますが、ほとんどのコンピュータでは、整数および実数の加算・減算・乗算・除算が用意されています。**制調装置**は、機械語の指令に従い、各装置に制御信号を送るための回路です。**主記憶装置**は、プログラムやデータを記憶するための装置で、パソコンの筐体 (きょうたい：本体) の中に格納された LSI によって構成されます。なお、主記憶装置は、単に記憶装置と呼ぶこともあります。また、制御装置と演算装置を合わせて、**中央処理装置** (英語名では **CPU**(Central Processing Unit)) と呼びます。

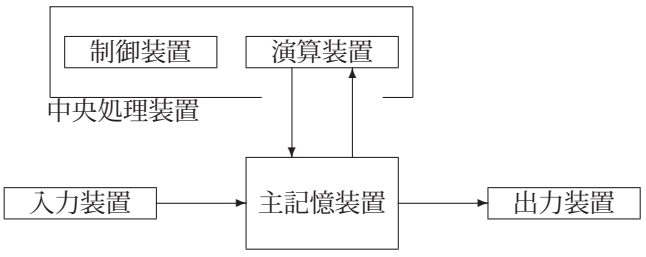


図 1.9: コンピュータのハードウェア (五大装置)

ただし、これだけではコンピュータは作動しません。まず、プログラムやデータを永続的に保

存するための補助記憶装置が必要になります。最も代表的なものは、ハードディスクでしょう。この他に、CD-ROM ドライブやフロッピードライブなどがあります。補助記憶装置は、データの入出力という観点から見れば、入力装置および出力装置と考えることもできます。プログラムを動作させるには、入力装置を介して、プログラムと計算に必要なデータをあらかじめ主記憶装置に入れておき、コンピュータはプログラムを CPU の中の制御装置に一命令ずつ取り込み、その機械語を解読して実行します。一つの命令の実行が終わると、次の命令を取り込んで解読して実行します[13]。さて、コンピュータは、主記憶装置に格納されたプログラムを逐一実行していくと説明しました。しかし、プログラムは初めから主記憶装置に格納されているわけではありません。現在のコンピュータは、主記憶装置に半導体を用いていますので、電源が入っていないと情報を貯えることができません。つまり、電源投入時には主記憶装置には何のデータもプログラムも格納されていません。実際に、パソコンの電源を投入しても、直ちにコンピュータが利用可能になるわけではありません。パソコンの電源を入れると、まず、コンピュータ内部に組込まれた非常に小さなプログラムが主記憶装置に取り込まれます。そして、そのプログラムが起動され、オペレーティングシステム (Operating System、略して OS とも呼びます) という巨大な管理プログラムを主記憶装置に取り込むのです。この過程をブートストラップと呼びます[14]。さて、このオペレーティングシステムは、パソコン用としてはマイクロソフトの Windows 7、Windows 8、Unix 系の Linux、Free BSD などいくつか種類があります。通常、プログラムは、このオペレーティングシステムの管理下で動作します。たとえば、Windows 7 上で、WORD(ワードプロセッサ)、 EXCEL(表計算ソフト)、ソリティア (ゲーム) を実行させていると、これらのプログラムが、オペレーテイングシステムとともに主記憶装置上に置かれるのです[15]。

1.4　本書で提供するソフトウェアについて

さて、世の中には、COMET II コンピュータは実在しないと前述しました。それならば、なぜ、CASL II アセンブラ言語で書かれたプログラムを CASL II アセンブラで COMET II の機械語に変換して実行できるのでしょうか。それは、COMET II の機械語を実行するコンピュータと同じ動作を行うプログラムが用意されているからです。これを **COMET II シミュレータ**と呼びます。

本書では、Windows 系 OS や Unix 系 OS で動作する CASL II アセンブラと COMET II シミュレータを用意いたしました。

また、本書では、Windows 系 OS で動作する **CASL II 開発環境 CASLDV** を読者のために用意しました。CASLDV は、その中に CASL II アセンブラと COMET II シミュレータを含んでいます。CASLDV は、CASL II のプログラムを作成し、実行するまでのすべての段階を一つのウインドウの中で行うことができるものです。なお、これにはエディタやデバッガも付いています。

---

[13] 現在のコンピュータは、実行速度を上げるために、パイプライン処理という方法を採用しています。この方法では、機械語の命令は、(1) 命令の取り込み、(2) 解読、(3) 実行というステップを同時に行うことにより処理効率を高めます。たとえば、プログラムの中に「命令 1、命令 2、命令 3」という 3 つの機械語があると、命令 1 の実行中に、命令 2 の解読が行われており、さらに同時に命令 3 の取り込みが行われています。

[14] JIS では、ブートストラップとは、「命令の集合であって、完全な計算機プログラムが記憶装置に入り終わるまで、後続の命令をロードするもの」と定義されています。

[15] これらのプログラムは、実際のパソコンでは、ハードディスク装置の一部を記憶装置に見立てた仮想記憶装置の中に格納されます。

図 1.10: 実行中のプログラムが主記憶装置に格納されている様子

なお、これらのソフトウェアは、以下に紹介するサイトから無料でダウンロードでき、自由にお使いいただけるようにしています。PC 教室のパソコンなどにもインストールして使っていただくことも可能です。

表 1.2: 本書で提供するソフトウェアと動作環境

動作可能 OS	Windows 系 OS	Unix 系 OS
CASL II アセンブラ	○	○
COMET II シミュレータ	○	○
CASLDV	○	×

入手希望の方は以下の URL から CASLDV のソフトウェアおよびマニュアルをダウンロードしてください。

```
http://www.officeuchida.com/CASL/
```

総合問題 1

次の文章を読み、正しい文章には○ 、誤りのある文章には×を付けてください。誤りのある場合には、その理由も指摘してください。また、条件によってどちらともいえない場合には△を付けて、その条件を説明してください。

1. コンピュータが実際に解釈して実行することのできる言語は機械語だけである。
2. 機械語には JIS 規格があって、どのコンピュータでも同じ機械語が通用する。
3. アセンブラ言語で書かれたプログラムは効率が良いのであるから、すべてのプログラムはアセンブラ言語で開発すべきである。
4. 一般にアセンブラ言語でプログラムを書くよりも高水準言語でプログラムを書いた方が人間にとっての労力がはるかに少ない。
5. 基本的に、アセンブラ言語は、そのコンピュータの機械語に 1 対 1 に対応している。
6. アセンブラ言語を使えば、どんなプログラムであってもそのコンピュータのすべての種類の機械語命令を実行できる (やや難)。
7. プログラムを実行させるには、必ず主記憶装置に配置しなければならない。
8. 以前は、プログラム内蔵方式が主流であったが、今はそうではない。
9. アセンブラ言語で書かれたプログラムは、アセンブルという処理を経るので、その実行効率はほんのわずかではあるが機械語に劣る。
10. C 言語のコンパイラの多くは優れた最適化機能を有しているので、アセンブラ言語で作成されたプログラムよりも常に効率が良くなる。
11. 同じ Windows 系 OS を搭載したパソコンでも、メーカが異なれば異なるコンピュータなので同じ機械語は動作しない。
12. 同じ CPU を搭載したパソコンであれば、Windows 系 OS でも Unix 系 OS でも、同じ機械語プログラムが動作する (やや難)。

第2章　数の表現とその演算

2.1　アセンブラ言語と数の表現

アセンブラ言語を理解するためには、ハードウェアの理解が必要です。ハードウェアの内部を覗くには、2 進法や 8 進法、16 進法といった数の表現方法を理解することが重要になります。日頃、10 進法に馴染んでいる私たちにとって、これらの表現方法は、慣れないと面倒です。しかし、逆に慣れてしまうと、アセンブラ言語で書かれたプログラムを読む際に、そこに書かれている数値が 16 進法で書かれた数値でないとなぜかしっくりいかなくなります。たとえば、255 という 10 進法で表現された数値よりも、16 進法で 00FF と表現された数値の方が読みやすく感じるようになります。とはいっても、みなさんは、日常とは違う数値で書かれた世界と初めて接する訳ですから、これが、アセンブラ言語を習得する最初の難関であると思って頑張ってください。なお、2 進法で記述されている数値を **2 進数**、8 進法で書かれている数値を **8 進数**、10 進法で書かれている数値を **10 進数**、16 進法で書かれている数値を **16 進数**と呼びます。また、数の表現を学習する際に大切なことは、「ただ単に暗記することではなく、その原理を理解する」ことです。そうすれば、数の表現に関する知識は自然と身についていきます。

2.2　10 進法

一般に、私たちは 10 進法を用いています。その理由は、「私たちの手の指の合計が 10 本であったからだ」と言われています。10 進法では、0、1、2、と数えていきます。そして、9 の次が 1 つ繰り上がって 10 になります。そして、さらに、10、11、12、と数えていき、19 の次が 20 となります。ちょうど、10 進数の各桁には 0〜9 の 10 個の数字のどれかが対応することになります。さて、10 進法では、12345 という数値は、

$$\mathbf{1} \times 10000 + \mathbf{2} \times 1000 + \mathbf{3} \times 100 + \mathbf{4} \times 10 + \mathbf{5}$$

というように表現できます。つまり、12345 という数値は、10000 が 1 個、1000 が 2 個、100 が 3 個、10 が 4 個、1 が 5 個ある数ということになります。これをもう少し、数学的に書くと

$$\mathbf{1} \times 10^4 + \mathbf{2} \times 10^3 + \mathbf{3} \times 10^2 + \mathbf{4} \times 10^1 + \mathbf{5} \times 10^0$$

と書くことができます。各桁の 10^k をその桁の**重み**と呼びます。12345 という 10 進法による数値を図的に表現すると次のようになります。

10 進数値の桁		5 桁目		4 桁目		3 桁目		2 桁目		1 桁目
桁の重み		$10000(=10^4)$		$1000(=10^3)$		$100(=10^2)$		$10(=10^1)$		$1(=10^0)$
その値		10000		2000		300		40		5
合計値	**12345** =	10000	+	2000	+	300	+	40	+	5

前述のように 10 進法の場合、各桁の重みは 10^k となります。一般に、n 進法では、k 桁目の重みは n^k となります。

ポイント 2-1 ： 10 進法の重み

10 進法では各桁の重みは 10^k となります。

$$12345 = 1 \times 10^4 + 2 \times 10^3 + 3 \times 10^2 + 4 \times 10^1 + 5 \times 10^0$$

演習問題 *2-1* ：10 進法以外の進法

私たちは、日常的に 10 進法を用いていますが、例外的に 10 進法以外の進法も身近に用いています。どのような進法を何に用いているか例を挙げて説明してください。

2.3　2 進法

2.3.1　2 進法の表現

それでは、まず、**2 進法**について考えてみましょう。2 進法では、0 と 1 の 2 つの数字だけを使います。ゼロは 2 進法でも 0 と表現しますし、1 も同様に 2 進法で 1 と表現します。しかし、10 進数と 2 進数の表現が同じなのは、この 2 つ (0 と 1) だけです。それでは、10 進法の 2 は、2 進法ではどのように表現するのでしょうか。数が 0 と 1 しかないので、1 桁では 2 という値は表現できません。そこで、桁を 1 つ繰り上げて、10 という数で 2 を表現します。

表 2.1 に、10 進数の 0〜8 に対応する 2 進法の数値を示します。数字が少ないために、10 進数の 8 を表現するのに 2 進数では 4 桁も必要になってしまいます。各桁の繰り上がりの様子を観察し、何も見ないでもこれらの数値を書けるようにしてください。

ここで大切なことは、2 進法の各数値を暗記するのではなく、桁の繰り上がりの規則 (0 の次が 1 となり、その次に繰り上がる) を理解し、それを再現できるようにすることです。

表 2.1: 0〜8 の 2 進法の表現

10 進数	0	1	2	3	4	5	6	7	8
2 進数	0	1	10	11	100	101	110	111	1000

さて、それでは、この 2 進法で表現された数値、いわゆる 2 進数について考えてみましょう。たとえば、10101 という 2 進数は、次のように解釈され、10 進法の 21 であることが分かります。

$$\mathbf{1} \times 2^4 + \mathbf{0} \times 2^3 + \mathbf{1} \times 2^2 + \mathbf{0} \times 2^1 + \mathbf{1} \times 2^0 = \mathbf{1} \times 16 + \mathbf{0} \times 8 + \mathbf{1} \times 4 + \mathbf{0} \times 2 + \mathbf{1} \times 1 = 16 + 4 + 1 = \mathbf{21}$$

すなわち、k 桁目の重みが、2^k と解釈されます。

2 進数値の桁		**1**	**0**	**1**	**0**	**1**
桁の重み		2^4	2^3	2^2	2^1	2^0
		16	8	4	2	1
その値		16	0	4	0	1
合計値	**21** =	16 +	0 +	4 +	0 +	1

2 進数で 10101 という数は、私たちの分かりやすい 10 進数では、16+4+1=21 という数になります。なお、数値が 2 進数であることを明示的に示すために、数値の最後に $_{(2)}$ を付けて、$10101_{(2)}$ と表現することもあります。一般に、n 進数の場合、$_{(n)}$ を付けますが、10 進数の場合には、通常省略します (表 2.2 参照)。

表 2.2: n 進数の示し方

進数	例
2 進数	$101101_{(2)}$
8 進数	$57_{(8)}$
10 進数	45
16 進数	$2D_{(16)}$

表 2.3 に 0 から 63 までの 2 進数とそれに対応する 10 進数の表を示します。各桁の位置関係が分かるように表 2.3 では全ての 2 進数を 7 桁で示し、先行するゼロを明示します。

表 2.3: 0 から 63 までの 2 進数とそれに対応する 10 進数

10 進	2 進	10 進	2 進	10 進	2 進	10 進	2 進
0	000000	16	010000	32	100000	48	110000
1	000001	17	010001	33	100001	49	110001
2	000010	18	010010	34	100010	50	110010
3	000011	19	010011	35	100011	51	110011
4	000100	20	010100	36	100100	52	110100
5	000101	21	010101	37	100101	53	110101
6	000110	22	010110	38	100110	54	110110
7	000111	23	010111	39	100111	55	110111
8	001000	24	011000	40	101000	56	111000
9	001001	25	011001	41	101001	57	111001
10	001010	26	011010	42	101010	58	111010
11	001011	27	011011	43	101011	59	111011
12	001100	28	011100	44	101100	60	111100
13	001101	29	011101	45	101101	61	111101
14	001110	30	011110	46	101110	62	111110
15	001111	31	011111	47	101111	63	111111

表 2.3 の数値ををすべて覚える必要はありません。2 進数を 10 進数に変換する必要がある場合には、まず、ノートに次に示す 2 進数の各桁の重みを右から 1、2、4、8、…と書きます。

… 2048 1024 512 256 128 64 32 16 8 4 2 1

これらは、1 から始めて数を 2 倍ずつしていったものです。もちろん、これらの数値はいちいち計算して書くのではなく、即座に書けるようにしておく必要があります。しかし、これもいちいち暗記するのではなく、l を 2 倍していく計算を何度か行えば自然に覚えられます。さて、たとえば、$10011011010_{(2)}$ という数を 10 進数に変換することを考えましょう。まず、この各桁の数値 (0 か 1) を重みの下に書きます。

…	2048	1024	512	256	128	64	32	16	8	4	2	1
		1	0	0	1	1	0	1	1	0	1	0

次に、数字が 1 の部分だけを下に転記し、その合計を加えます。

…2048	1024	512	256	128		64	32	16		8	4	2	1		
	1	0	0	1		1	0	1		1	0	1	0		
	1024	+		128	+	64	+	16	+	8	+	2		=	**1242**

この合計値 1242 が求める 10 進数となります。この計算方法だけを覚えておけば、暗記に頼る必要はありません。

ただし、2 進数の表現に関して、いくつかの特徴を理解しておく必要があります。表 2.3 をみると、奇数のときに最下位桁が 1 になり、偶数のときに最下位桁が 0 になっていることに気付きませんか。これは、2 進数 $(b_5b_4b_3b_2b_1b_0\ _{(2)})$ の定義

$$b_5 \times 2^5 + b_4 \times 2^4 + b_3 \times 2^3 + b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$$

$$= b_5 \times 32 + b_4 \times 16 + b_3 \times 8 + b_2 \times 4 + b_1 \times 2 + b_0 \times 1$$

を思い出せばすぐに理解できるでしょう。2 桁め (b_1) 以降の重みは必ず、2 の倍数になっているので、その数が偶数か奇数かは、最下位桁 b_0 の値で決まるのです。b_0 が 1 ならその数は奇数であり、b_0 が 0 ならその数は偶数です。同様に、下 2 桁が 00 になっているとその数は 4 の倍数であり、下 3 桁が 000 になっていると 8 の倍数であることが分かるでしょう。一般に、2 進数の下 n 桁すべてが 0 になっていると、その数は 2^n の倍数となります。このことを知っておくと、奇遇判断 (数が偶数か奇数の判断) や 2^n の倍数かどうかの判断を、後述するビット操作[1]を使って簡単に行うことができます。

ポイント 2-2 ： 2 進数を 10 進数に変換する

2 進数 $10011011010_{(2)}$ を 10 進数に変換するには「2 の重み (2^k)」を書く

…2048	1024	512	256	128		64	32	16		8	4	2	1		
	1	0	0	1		1	0	1		1	0	1	0		
	1024	+		128	+	64	+	16	+	8	+	2		=	**1242**

演習問題 *2-3-1* ：2 進法

(1) 次の 2 進数を 10 進数に変換してください。

(a) $1010_{(2)}$

(b) $111011_{(2)}$

(c) $10101011_{(2)}$

(2) 次ページの表の 2 進数値が「奇数」、「2 の倍数 (＝偶数)」 、「4 の倍数」、「8 の倍数」のどれに相当するか次の表を○× (○:相当する、×:相当しない) で埋めてください。参考のために、表の最初に簡単な例を載せておきます。

[1] ビット操作については、第 10 章で詳しく説明します。

数値	奇数	2 の倍数 (=偶数)	4 の倍数	8 の倍数
$10_{(2)}$	×	○	×	×
$100_{(2)}$	×	○	○	×
$101_{(2)}$	○	×	×	×
$11011_{(2)}$				
$100000_{(2)}$				
$111110_{(2)}$				
$110100_{(2)}$				
$100011_{(2)}$				

2.3.2 2 進法の計算方法

それでは 2 進法の計算方法について考えてみましょう。10 進数では $1+1=2$ となりますが、2 進数では $1+1=10$(正確に書くなら、$1_{(2)}+1_{(2)}=10_{(2)}$) となります。それでは、もっと桁の多い 2 進数の加算はどうするのでしょう。

まず、$3+4$ を考えてみましょう。この答えは 7 ですが、2 進数では次のように計算します。2 進数では、$11_{(2)}$ が 3、$100_{(2)}$ が 4 であることは、もう理解していますよね。桁を合わせるために、次の式では $11_{(2)}$ を $011_{(2)}$ と 3 桁にしていることに注意してください。

$$\begin{array}{cccc} & 0 & 1 & 1 \\ + & 1 & 0 & 0 \\ \hline & 1 & 1 & 1 \end{array}$$

上記のように、2 進数の各桁を加えます。この場合、すべての桁が 1 となり、$111_{(2)}$ となります。この数は 10 進数では 7 となり、計算が正しいことがわかります。

それでは、$20+40=60$ の計算をしてみましょう。$20=10100_{(2)}$、$40=101000_{(2)}$ ですから、次のようになります。

$$\begin{array}{cccccc} & 0 & 1 & 0 & 1 & 0 & 0 \\ + & 1 & 0 & 1 & 0 & 0 & 0 \\ \hline & 1 & 1 & 1 & 1 & 0 & 0 \end{array}$$

これも同様に各桁を加えればよく、その答えは $111100_{(2)}=60$ となります。

それでは、$13+9=22$ を計算してみましょう。$13=1101_{(2)}$ 、$9=1001_{(2)}$ です。$1+1$ の計算を行う桁が 1 桁めと 4 桁めにあります。この場合、繰り上がって $10_{(2)}$ となりますので、次の桁にその部分を足します。

$$\begin{array}{cccccc} & 1 & 1 & 0 & 1 & \\ + & 1 & 0 & 0 & 1 & \\ \hline & 10 & 1 & 0 & 10 & \text{各桁の合計} \\ 1 \leftarrow & & & 1 \leftarrow & & \text{繰り上がり部分} \\ \hline 1 & 0 & 1 & 1 & 0 & \end{array}$$

結果は、$10110_{(2)}=22$ となります。

次に、$15+1=16$ はどうなるでしょうか。$15=1111_{(2)}$、$1=0001_{(2)}$ ですから、次のようになります。

```
      1   1   1   1
+     0   0   0   1
-----------------------
      1   1   1  10
              1 ←┘      繰り上がり部分
-----------------------
      1   1  10   0
          1 ←┘          繰り上がり部分
-----------------------
      1  10   0   0
      1 ←┘              繰り上がり部分
-----------------------
     10   0   0   0
  1 ←┘                  繰り上がり部分
-----------------------
  1   0   0   0   0
```

答えは、$10000_{(2)}=16$ となります。このように、繰り上がりは次々に伝播していくので注意が必要です。

なお、一般的な 2 進数の演算では、表現できる桁数に制限はありませんが、COMET II の場合、16 桁までの 2 進数しかデータを保持できない (2.3.3 項参照) ので、計算結果がその桁数を超える値になった場合にはその部分は無視されます。計算結果の桁数が表現できる桁を越えることをオーバーフローと呼びます。

```
              ←―― 16 桁 ――→
              1111111111111111
+                            1
-----------------------------------
      1       0000000000000000
      ↑
      オーバー
      フローした
      部分は無視
      される
              |
              ↓
              0000000000000000   ←最終的な答え
              ←―― 16 桁 ――→
```

なお、減算については、2.6.3 項「2 の補数と減算の仕組み」(26 ページ) で説明します。

演習問題 *2-3-2* ：2 進法の計算

2 進法で、次の計算を行ってください。

(1)　$11_{(2)}+10101_{(2)}$

(2)　$1001_{(2)}+110111_{(2)}$

2.3.3 2 進数とハードウェアの関係

それでは、なぜ、2 進数などという面倒な表現方法を学習するのでしょうか。それは、コンピュータのハードウェアと密接な関係があります。現在のコンピュータは、電流の ON/OFF という 2 つの状態でデータを表現します。この ON/OFF が、ちょうど 2 進数の 0 と 1 に対応しているのです。COMET II のコンピュータの内部では、21 という数値は、$0000000000010101_{(2)}$ という形で記憶されます。通常は、0 が OFF、l が ON に対応します。

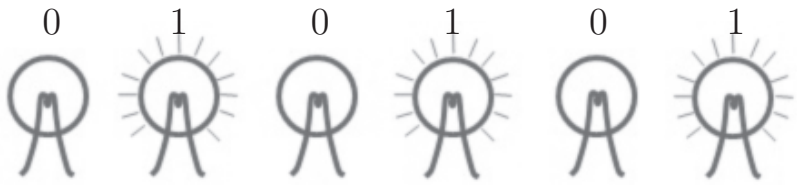


図 2.1: 電流の流れとビットの ON/OFF

2 進数の 1 桁がちょうどこの ON/OFF に相当しますが、この情報を保持した桁のことをビット (bit) と呼びます。ON/OFF に相当する 1 ビットでは、0 と 1 の 2 つのデータを表現できます。2 ビットでは、00、01、10、11 の 4 つのデータを表現できます。3 ビットでは、000、001、010、011、100、101、110、111 の 8 つのデータを表現できます。一般に、n ビットでは、2^n 個のデータを表現できます。8 ビットのまとまりを**1 バイト** (byte) と呼びます。1 バイトは $2^8 = 256$ 通りのデータを表現できます。COMET II は、2 バイトすなわち 16 ビットまでのデータを保持できますので、$2^{16} = 65536$ 通りのデータを表現することができます。なお、前述のように、COMET II では、2 進数値を 16 ビットで表現しますが、その各ビットには次のようなビット番号がついています (符号については、2.6 節 (22 ページ) で説明します)。

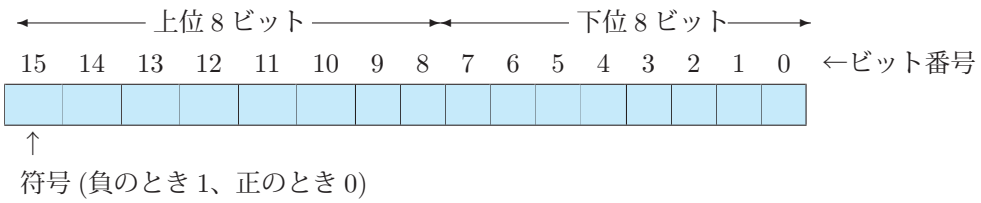


図 2.2: COMET II におけるビット構成

ビット番号 0 のビットを**最下位ビット**、ビット番号 15 のビットを**最上位ビット**と呼ぶことがあります。

2.4 8 進法

2 進数値は、コンピュータ内の電源の ON、OFF にダイレクトに対応するので、コンピュータの内部のデータの様子を、そのまま表現できることがお分かりいただけたかと思います。しかし、2 進数では、使用する数字が 0 と 1 だけの 2 つしかないので、どうしても数の桁数が多くな

ります。そこで実際には、8 進数、16 進数を使います。CASL II では 8 進数を表現することができませんが、本書では 8 進数も説明することにします。8 進数では、0 から 7 までの数値を使ってデータを表現します。8 進数では、$0_{(8)}$、$1_{(8)}$、$2_{(8)}$、$3_{(8)}$、$4_{(8)}$、$5_{(8)}$、$6_{(8)}$、$7_{(8)}$ と数えていき、$7_{(8)}$ の次に繰り上がりが起きて、$10_{(8)}$ となります。そして、$11_{(8)}$、$12_{(8)}$、…、$17_{(8)}$ と数えていき、$17_{(8)}$ の次に $20_{(8)}$ となります。それでは、$123_{(8)}$ という数値は、10 進法ではどのような値になるのでしょう。$123_{(8)}$ は、$1\times8^2+2\times8^1+3\times8^0=1\times64+2\times8+3\times1=64+16+3=83$ となり、10 進数では 83 ということになります。表 2.7(22 ページ) に、10 進数、8 進数、16 進数の 0 から 63 までの一覧表を示すので参考にしてください。

8 進数と 2 進数の聞には、とても単純な関係があります。8 進数の各桁は、ちょうどそれに対応する 2 進数を下位から順に 3 桁ずつ区切った値に対応します (3 桁に満たない場合は 0 を補います)。たとえば、$1110011001010_{(2)}$ は $16312_{(8)}$ となります。

```
   ┌── 3 桁に満たないので 0 を補う
  001   110   011   001   010   (2)
   1     6     3     1     2    (8)
```

2 進数値を 3 桁ごとに区切ると、その中には $2^3=8$ 個のデータがあり、これがちょうど 8 進数の 0〜7 の数値に対応しています。そのため、2 進数を 3 桁ごとに区切ると、ちょうど、8 進の各桁の重みに一致するので、このような単純な関係になるのです。

この関係から、8 進数の 0 から 7 までのビットパターンを覚えておけば、前述の $16312_{(8)}$ という 8 進数がどのようなビットパターンを形成するかはすぐに思い浮かべられると思います (演習問題 2-4 参照)。

表 2.4: 8 進数のビットパターン

桁	ビットパターン
0	000
1	001
2	010
3	011
4	100
5	101
6	110
7	111

演習問題 *2-4* ：*8* 進法

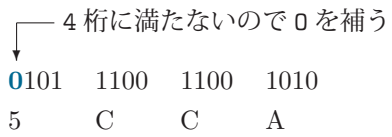
次の数値を 10 進数と 2 進数に変換してください。

(1)　$13_{(8)}$
(2)　$57_{(8)}$
(3)　$1234_{(8)}$

2.5 16 進法

さて、**16 進法**の場合も 2 進数や 8 進数の考え方と同様に考えます。16 進数では、16 個の数字を使います。そのため、0 から 9 までの 10 個の数字と A、B、C、D、E、F までの 6 個の数字の合計 16 の数字を使います[2]。

表 2.5 に、16 進数の 0〜9、A〜F の各数値に対応する 2 進数、10 進数を示します。16 進数では、$0_{(16)}$、$1_{(16)}$、$2_{(16)}$、$3_{(16)}$、$4_{(16)}$、$5_{(16)}$、$6_{(16)}$、$7_{(16)}$、$8_{(16)}$、$9_{(16)}$ と数えていき、この次に、$A_{(16)}$ となります。これは繰り上がりが起きたのではありません。9 の次に対応する数字として (10 進数では 10 ですが)、A という文字を数字として用いるのです。そして、A の次の数は $B_{(16)}$ となります。B は、10 進数では 11 に相当します。そして、$C_{(16)}$、$D_{(16)}$、$E_{(16)}$、$F_{(16)}$ まで数えていき、この次に繰り上がりが発生し、$10_{(16)}$ になります。そして、$11_{(16)}$、$12_{(16)}$、…、$1F_{(16)}$ と数えていき、$1F_{(16)}$ の次が $20_{(16)}$ になります。それでは、$123_{(16)}$ という数値は、10 進法ではどのような値になるのでしょう。$123_{(16)}$ は、$1\times16^2+2\times16^1+3\times16^0 = 1\times256+2\times16+3\times1 = 256+32+3 = 291$ となり、10 進数では 291 ということになります。表 2.7 に、10 進数、8 進数、16 進数の 0 から 63 までの一覧表を示すので参考にしてください。16 進数と 2 進数の各桁の間には、とても単純な関係があります。16 進数の各桁は、2 進数を 4 桁ずつ区切った数になります (4 桁に満たない場合は 0 を補います)。



上記のように、$101110011001010_{(2)}$ は、$5CCA_{(16)}$ となります。

表 2.5: 16 進数字の表現

16 進数	10 進数	2 進数	16 進数	10 進数	2 進数
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	A	10	1010
3	3	0011	B	11	1011
4	4	0100	C	12	1100
5	5	0101	D	13	1101
6	6	0110	E	14	1110
7	7	0111	F	15	1111

16 進数は機械語の世界では特によく使います。16 進の数値と 2 進の数値の対応づけは、完全に頭の中に入れておく必要があります。たとえば、$F_{(16)}$ は $1111_{(2)}$ を意味し、$A_{(16)}$ は $1010_{(2)}$ を、$8_{(16)}$ は $1000_{(2)}$ を意味します。このことを頭の中に入れておけば、16 ビットの数値の場合、$FFFF_{(16)}$ はすべてのビットが 1、$AAAA_{(16)}$ であれば 10 のビットの繰り返し、$8000_{(16)}$ であれば最上位ビット (ビット番号 15) のみが 1 であとは全て 0 であることが直ちに分かります。このように、慣れてくると 16 進数を見ただけで、その数値のビットパターンを連想することができます。

[2]16 進数字 A〜F に対して、小文字の a〜f を使うこともあります。しかし、CASL II は大文字しか扱わないので、本書では、16 進数を表現する際は、A〜F のみを使うことにします。

表 2.6: 代表的な 16 ビット数値

16 進数	2 進数
0000	0000000000000000
0001	0000000000000001
FFFF	1111111111111111
F000	1111000000000000
8000	1000000000000000
8888	1000100010001000
1111	0001000100010001
AAAA	1010101010101010

表 2.7: 10 進数、8 進数、16 進数の表

10 進	8 進	16 進	10 進	8 進	16 進	10 進	8 進	16 進	10 進	8 進	16 進
0	0	0	16	20	10	32	40	20	48	60	30
1	1	1	17	21	11	33	41	21	49	61	31
2	2	2	18	22	12	34	42	22	50	62	32
3	3	3	19	23	13	35	43	23	51	63	33
4	4	4	20	24	14	36	44	24	52	64	34
5	5	5	21	25	15	37	45	25	53	65	35
6	6	6	22	26	16	38	46	26	54	66	36
7	7	7	23	27	17	39	47	27	55	67	37
8	10	8	24	30	18	40	50	28	56	70	38
9	11	9	25	31	19	41	51	29	57	71	39
10	12	A	26	32	1A	42	52	2A	58	72	3A
11	13	B	27	33	1B	43	53	2B	59	73	3B
12	14	C	28	34	1C	44	54	2C	60	74	3C
13	15	D	29	35	1D	45	55	2D	61	75	3D
14	16	E	30	36	1E	46	56	2E	62	76	3E
15	17	F	31	37	1F	47	57	2F	63	77	3F

演習問題 *2-5* ：16 進法

次の 16 進数を 2 進数に変換してください。

(1)　$5CCA_{(16)}$
(2)　$FDDA_{(16)}$
(3)　$1234_{(16)}$
(4)　$4444_{(16)}$
(5)　$CAFE_{(16)}$

2.6　負のデータの扱い

2.6.1　負のデータと 2 の補数表現

16 ビットのデータは、2^{16} 通りの表現が可能ですから、65536 通りのデータを表現することができます。これを 10 進数の数値に対応させると、0 から 65535 までの数値を表現することができます。しかし、数値には負の数も存在します。COMET II では、負の数を表現するのに、**2 の補数**という方法を使います。

ポイント 2-3 ： 負の数は 2 の補数で表現

COMET II では、負の数を表現するのに、2 の補数を用いる。

16 ビットで説明すると、桁が多くなって説明するのが大変なので、とりあえず 3 ビットで説明します。図 2.3 のビット表現を想定します。

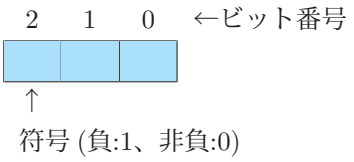


図 2.3: 3 ビットの構成

3 ビットのデータは 2^3 通りの表現が可能ですから、8 通りのデータを表現することができます。10 進数値に対応させると、3 ビットでは 0 から 7 までの数値を表現することができます。しかし、0 から 7 までの数値は、すべて 0 以上の数値です。これは、ちょうど表 2.8 の「符号無し表現」の部分に相当します。

表 2.8: 3 ビットのビットパターンとそれが表現する数

2 進数	符号無し表現	符号付き表現
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

一般にコンピュータでは、2 進数のビットパターンの数値の扱いを、「符号無しデータ」と「符号付きデータ」の 2 通りの仕方で処理します。ここで大切なことは、同じビットパターンのデータに対してコンピュータが 2 つの異なる解釈をするということです。

符号無しデータでは、2 進のビットパターン $(b_2b_1b_0)$ をそのまま、$b_2 \times 2^2 + b_1 \times 2^1 + b_0 \times 2^0$ という形で表現します。しかし、符号無しデータの場合、最上位ビット (3 ビットの場合には、ビット番号 2、16 ビットの場合にはビット番号 15) を符号ビットとみなして、最上位ビットに 1 が立っているとその数値を負の数とみなします。表 2.8 では、ビット番号 2 が 1 となっている数は、符号付き表現では負の数となっています。

ポイント 2-4 ： 符号付きデータの表現

符号付きデータでは、最上位ビット (ビット番号 15) を符号ビットとみなし、それが 1 ならば負、0 ならば非負のデータとしている。

2 の補数という方法で負の数を表現する場合、3 ビットでは -2^2 から 2^{2-1} までの数が表現できます。この場合、$100_{(2)}$ が一番小さな値 (-4) になり、$111_{(2)}$ が -1 になります。

一般に、n ビットで負の数を表現する場合、-2^{n-1}〜$2^{n-1}-1$ までの数を表現でき、$100...00_{(2)}$(全部で n ビット) が -2^{n-1} を、$111...11_{(2)}$(全部で n ビット) が -1 を表現します。

COMET II の数値は 16 ビットなので、-2^{15}〜$2^{15}-1$、すなわち、−32768〜32767 までを表現でき、$1000000000000000_{(2)}$ が −32768 となり、$1111111111111111_{(2)}$ が -1 となります。

表 2.9: 16 ビットのビットパターンとそれが表現する数

2 進数	符号無し表現	符号付き表現
0000000000000000	0	0
0000000000000001	1	1
0000000000000010	2	2
0000000000000011	3	3
≀	≀	≀
0111111111111111	32767	32767
1000000000000000	32768	-32768
1000000000000001	32769	-32767
≀	≀	≀
1111111111111100	65532	-4
1111111111111101	65533	-3
1111111111111110	65534	-2
1111111111111111	65535	-1

さて、ここで大切なことは、前述したように、コンピュータの内部では、1 つのビットパターンに対して、符号付きデータと符号無しデータの 2 つのデータというそれぞれ異なる 2 つの解釈があるということです。2 進数 $1000000000000001_{(2)}$ は、符号無しデータとみなした場合 32769 ですが、符号付きデータとみなした場合には、-32767 となります。COMET II の各命令は、データを「符号付きデータとみなして計算するもの」と「符号無しデータとみなして計算するもの」とに分けられています。たとえば、データの加算に対して、符号付きデータで計算する場合には ADDA という命令を用いますが、符号無しデータで計算する場合には ADDL という命令を用います。

ポイント 2-5 ： COMET II の命令は符号付きデータと符号無しデータで異なる

符号付きデータを加算する時は ADDA 命令、符号無しデータを加算する時は ADDL 命令を用いる

5.4 節「フラグレジスタと条件判断」(102 ページ) で詳しく説明しますが、COMET II には、演算結果が負の値になった場合、「演算結果が負になりましたよ」という印が付くサインフラグ (SF) というものがあります。

ADDA を用いて、データを符号付きデータであるとみなして計算した結果が $1000000000000001_{(2)}$ となった場合には、−32767 と見なされるので、SF には「負になった」という意味の 1 がセットされ、ADDL を用いてデータを符号付無しデータとみなして計算した結果が $1000000000000001_{(2)}$ となった場合には、32769 と見なされるので、SF には「非負になった」という意味の 0 がセットされます。

演習問題 *2-6-1* ：符号無し表現と符号付き表現

次の 4 ビットの 2 進数を 10 進数の符号無し表現と符号付き表現 (ビット番号 3 を符号ビットとする) で示してください。

(1) $1111_{(2)}$
(2) $1010_{(2)}$
(3) $1110_{(2)}$
(4) $0001_{(2)}$
(5) $0101_{(2)}$

2.6.2 負の数 (2 の補数) への変換

さて、負の数の表現方法は、前項で説明したので理解できたと思います。そこで、本項では、正の数を負の数に変換したり、その逆を行う方法について説明します。

ある数の 2 の補数 (いわゆる、負の数) を求めるには、次のようにします。たとえば、3 の 2 の補数を求めて、−3 にしてみましょう。

その手順は、とても単純で、次の 2 つの手順を行うだけです。

(1) すべてのビットを反転する
(2) 1 を加える

3 の場合には、次のようになります。

3	$0000000000000011_{(2)}$	
	↓ (ビット反転)	
	$1111111111111100_{(2)}$	
+	$1_{(2)}$	← 1 を加える
−3	$1111111111111101_{(2)}$	

ビット反転して 1 を加えたところ、$1111111111111101_{(2)}$ が求められました。表 2.9 をみると、この数値が −3 になっていることが確認できますね。それでは、この数値に対してもう一度まったく同じ操作をしてみましょう。

−3	$1111111111111101_{(2)}$	
	↓ (ビット反転)	
	$0000000000000010_{(2)}$	
+	$1_{(2)}$	← 1 を加える
3	$0000000000000011_{(2)}$	

なんと、ちゃんと 3 に戻っていますね。つまり、2 の補数を求めるというのは、符号を反転させることに等しく、また、このようにとても単純な操作で出来てしまうのです。

ポイント 2-6 ： 2 の補数を求める手順

(1) すべてのビットを反転する
(2) 1 を加える

演習問題 *2-6-2* ：2 の補数

次の数値の 2 の補数を 16 ビットの 2 進数で求めなさい。

(1)　5
(2)　10
(3)　−5

2.6.3　2 の補数と減算の仕組み

COMET II には減算の命令として、SUBA(符号付きデータ) と SUBL(符号無しデータ) の 2 つがあります。しかし、整数の減算を行うという演算回路は実際には用意されていないこともあるのです。それは、減算という操作が、「2 の補数を加える」という操作でできてしまうからなのです。

11 − 6 という計算について調べてみましょう。この 2 つの数値は 16 ビットの 2 進数では次の通りです。

11　：　$0000000000001011_{(2)}$
6　：　$0000000000000110_{(2)}$

さて、6 の 2 の補数を求めて、−6 に変換して、これを 11 に加えてみましょう。まず、6 の 2 の補数を求めます。

6	$0000000000000110_{(2)}$	
	↓ (ビット反転)	
	$1111111111111001_{(2)}$	
＋	$1_{(2)}$	← 1 を加える
−6	$1111111111111010_{(2)}$	

−6 は、$1111111111111010_{(2)}$ となりました。この値を 11 に加えて 11 − 6 を求めてみましょう。次の例のように答えのビット数は 17 ビットになりますが、COMET II には格納する部分が 16 ビット分しかありませんので、オーバーフローした部分は切り捨てられます。

	11	：	$0000000000001011_{(2)}$
＋	−6	：	$1111111111111010_{(2)}$
			$10000000000000101_{(2)}$
			↑
			切り捨て
	5		$0000000000000101_{(2)}$

答えは、$101_{(2)}$、すなわち 5 になりました。これは、11 − 6 の答えです。つまり、2 の補数を使えば、負の数をそのまま加算することによって、正しく演算が行われるのです。

それでは、2 – 6 を計算してみましょう。–6 は $1111111111111010_{(2)}$ であることが分かっていますから、次の計算を行えば良いことが分かります。

$$\begin{array}{rrcl} & 2 & : & 0000000000000010_{(2)} \\ + & -6 & : & 1111111111111010_{(2)} \\ \hline & -4 & & 1111111111111100_{(2)} \end{array}$$

今回の演算では、オーバーフローした部分はありません。$1111111111111100_{(2)}$ は、負の数としてみると –4 であり 2 – 6 が計算されたことがわかります。

演習問題 *2-6-3* ：2 の補数の計算

次の計算を 16 桁の 2 進数のレベルで 2 の補数を用いて行いなさい。

(1) 5 – 3
(2) 2 – 10
(3) 14 – 5

***Coffee Break 2-1* ： 小数点以下の 2 進表現**

COMET II では扱えませんが、実際のコンピュータでは、実数を扱うことができます。科学技術計算では、実数データも 2 進数で表現されます。しかし、実数データを 2 進数で表現すると、我々の日常感覚とは異なる現象が発生するので注意が必要です。

まず、2 進数で浮動小数点数を表すことを考えましょう。$0.b_1b_2b_2\cdots$ という表現は次のように解釈されます。

$$0.b_1b_2b_2\cdots = b_1 \times 2^{-1} + b_2 \times 2^{-2} + b_3 \times 2^{-3} + \cdots$$

具体的な事例について見ていきましょう。まず、10 進数の 0.5 ですが、これは 2 進数では $0.1_{(2)}$ と表現されます。

$$0.5 = 1 \times 2^{-1} = 1 \times \tfrac{1}{2} = 0.1_{(2)}$$

次に、10 進数の 0.25 は 2 進数では $0.01_{(2)}$、0.125 は $0.001_{(2)}$ と表現されます。

$$0.25 = 1 \times 2^{-2} = 1 \times \tfrac{1}{4} = 0.01_{(2)}, \quad 0.125 = 1 \times 2^{-3} = 1 \times \tfrac{1}{8} = 0.001_{(2)}$$

10 進数の 0.75 は 0.5 + 0.25 なので、次の 2 進数になります。

$$0.75 = 0.5 + 0.25 = 2^{-1} + 2^{-2} = \tfrac{1}{2} + \tfrac{1}{4} = 0.11_{(2)}$$

ここで示したように 2^{-n} で表現される値あるいはその合計値は 2 進数で正確に表現できます。しかし、これに該当しない数値、たとえば、0.1、0.2、0.3 といった普通の数値は 2 進数で表現すると、なんと循環小数になってしまうのです。たとえば、0.1 は次の 2 進数になりますが、実はこれが大問題なのです。*Coffee Break 2-2*(29 ページ) をご覧ください。

$$0.1 = 0.0001100110011001100110011001100110011001100110011\cdots_{(2)}$$

2.7　各進数表現の変換

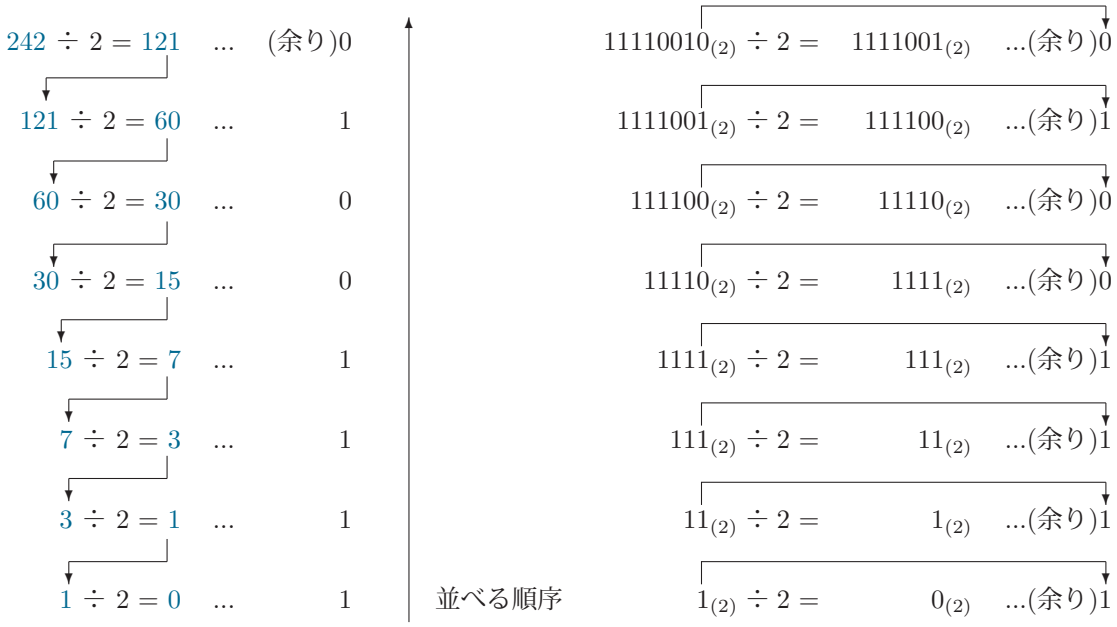
実際にアセンブラ言語でプログラムを作成する際には、10 進数を 2 進数や 16 進数に変換したり、あるいはその逆の変換を行う必要があります。ここでは、その方法について説明したいと思います。

2.7.1　10 進数の n 進数への変換

10 進数を n 進数に変換するには、その数を n で割って余りを並べていきます。

(1) 10 進数→ 2 進数

10 進数 d を 2 進数に変換することを考えてみましょう。$d = 242$ を 2 進数に変換する場合、次のように 2 で割って余りを並べていきます。



そして、上記のように余りを逆順に並べると、$242_{(10)} = 11110010_{(2)}$ となります。

この仕組みについて説明しましょう。実は、$11110010_{(2)} = 242_{(10)}$ という数を分析すると、次のことがわかります。

$11110010_{(2)}$ を 2 で割ることを考えます。この数を A とすると、次のように表現できます。

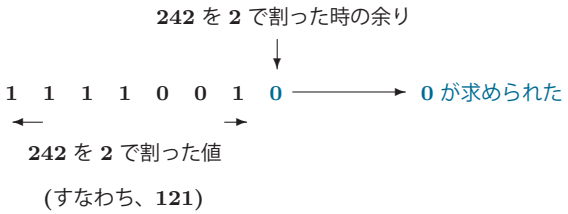
$$A = 1 \times 2^7 + 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0$$

この値を 2 で割ると、次のようになります。

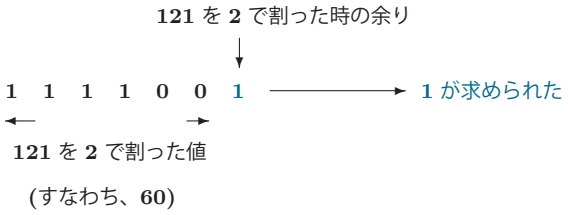
$$1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

この値は、A を 2 で割った値 $1111001_{(2)} = 121_{(10)}$ になっていますね。そして、A の最下位桁は、自分自身を 2 で割った時の余りになります。

つまり次のことが分かります。



これで、最下位桁が求められました (青い部分)。さらに $242_{(10)}$ を 2 で割った値 $1111001_{(2)} = 121_{(10)}$ について同じ処理をします。



これで、2 桁目が求められました。この手順を繰り返せば、すべての 2 進数が求められます。

ポイント 2-7 ： 10 進数→ 2 進数変換方法

10 進数を 2 進数にするには、2 で割って余りを逆順に並べる

演習問題 *2-7-1(1)* ：10 進数→ 2 進数

次の 10 進数を 2 進数に変換しなさい。

(1) 159
(2) 2152
(3) 10000

***Coffee Break 2-2* ： 0.1 の表現**

Coffee Break 2-1(27 ページ) で紹介したように、0.1 を 2 進数で表現すると、次のように循環小数になってしまいます。

$$0.1 = 0.0001100110011001100110011001100110011\cdots_{(2)}$$

0.1 だけではなく、0.2 も 0.3 も 2 進数で表現すると循環小数になってしまいます。

$$0.2 = 0.0011001100110011001100110011001100110\cdots_{(2)}$$

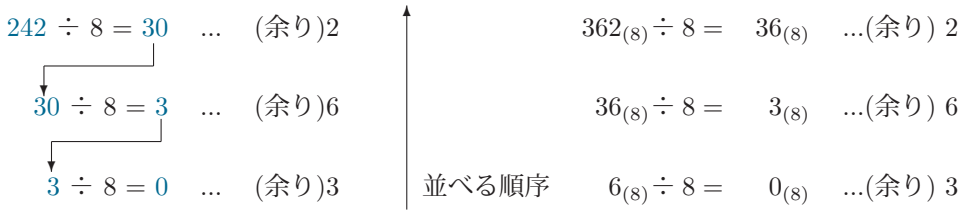
$$0.3 = 0.0100110011001100110011001100110011001\cdots_{(2)}$$

実は、科学技術計算で用いられる C 言語などでは、浮動小数点データはすべて 2 進の形式で表現されます。そのため、プログラム上で 0.1 と書いても、実際には 0.1 に近い近似値で表現されます。つまり、C 言語では、0.1 を 10 回加えても 1.0 にはならないのです。なお、0.1 を 2 進数に変換する方法は、*Coffee Break 4-1*(65 ページ) をご覧ください。

CASL II では、浮動小数点を扱いませんが、この事実はコンピュータを学習する方は覚えておくとよいでしょう。浮動小数点を扱う場合には、このような計算誤差に十分注意して作成する必要があります。

(2) 10 進数→ 8 進数

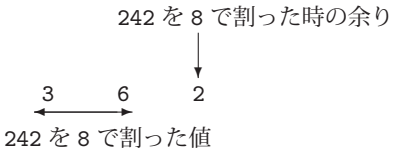
10 進数を 8 進数に変換する手順も同じで、8 で割って余りを求めます。



余りを逆順に並べて、$242_{(10)} = 362_{(8)}$ となります。

この仕組みについて説明しましょう。実は、$362_{(8)} = 242_{(10)}$ という数を分析すると、次のことがわかります。

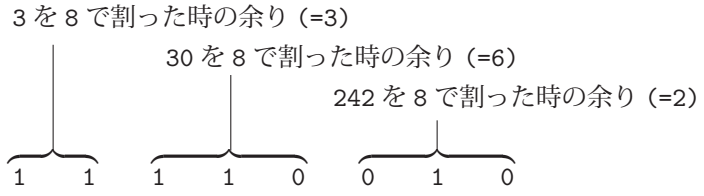
$362_{(8)}$ を 8 で割ることを考えます。これは定義によれば、$362_{(8)} = 3 \times 8^2 + 6 \times 8^1 + 2 \times 8^0$ ですね。これを 8 で割ると、$3 \times 8^1 + 6 \times 8^0$ となります。これは、$36_{(8)} = 30_{(10)}$ で、$362_{(8)}$ を 8 で割った値です。そして、$362_{(8)}$ の最下位桁である $2_{(8)}$ は、元の数 $362_{(8)}$ を 8 で割った時の余りになります。つまり、362 という 8 進数の各桁は次の関係になっているのです。



これで、最下位数が求められました。ところで、これを 2 進数で見ると次のようになります。2 進数を 8 で割るということは、その数値の各桁を 3 つ右へ移動させるのと同じ事です。この操作をシフト演算と呼びます。シフト演算については 6.3.5 項「シフト演算命令」(124 ページ) で詳しく説明します。

	242		30	2
11110	$010_{(2)}$	÷ 8 =	$11110_{(2)}$	...(余り)$010_{(2)}$
	30		3	6
11	$110_{(2)}$	÷ 8 =	$11_{(2)}$	...(余り)$110_{(2)}$
	3		0	3
	$11_{(2)}$	÷ 8 =	$0_{(2)}$	...(余り)$011_{(2)}$

242 を 8 進数 $362_{(8)}$ にした時のビット全体は、次のようになります。



実は、これは 2 進数を 3 ビットづつ区切るとなぜ 8 進数になるかの説明になっています。要するに、各桁は 8 で割った余りになっているので、このような単純な対応づけができるのです。

ポイント 2-8 ： 10 進数→ 8 進数変換方法

10 進数を 8 進数にするには、8 で割って余りを逆順に並べる

演習問題 *2-7-1(2)* ： 10 進数→ 8 進数

次の 10 進数を 8 進数に変換しなさい。

(1) 159

(2) 2152

(3) 10000

(3) 10 進数→ 16 進数

10 進数を 16 進数に変換するには、16 で割って余りを逆順に並べます。余りが 10〜15 のときは、A〜F の数字に対応させます。

$242 \div 16 = 15$ ... (余り)2

$15 \div 16 = 0$ ... (余り)15 → F　　並べる順序

余りを逆順に並べて、$242_{(10)} = \mathrm{F2}_{(16)}$ となります。

ポイント 2-9 ： 10 進数→ 16 進数変換方法

10 進数を 16 進数にするには、16 で割って余りを逆順に並べる

演習問題 *2-7-1(3)* ： 10 進数→ 16 進数

次の 10 進数を 16 進数に変換しなさい。

(1) 159

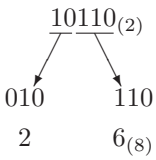
(2) 2152

(3) 10000

2.7.2 2 進数の 8 進数、16 進数への変換

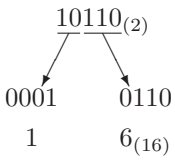
2 進数が求められている場合、それを用いて 8 進数、16 進数を簡単に求めることができます。2.4 節、2.5 節で説明したように、$10110_{(2)}$ を 8 進数に変換する場合には、3 ビットごとに分け 10 110 とし (3 桁に満たない場合には 0 を補います)、16 進数に変換する場合には、4 ビットごとに分け 1 0110 とします (4 桁に満たない場合は 0 を補います)。

2 進数→ 8 進数



$10110_{(2)} = 26_{(8)}$

2 進数→ 16 進数



$10110_{(2)} = 16_{(16)}$

ポイント 2-10 ： 2 進数→ 8 進数・16 進数変換方法

2 進数を 8 進数にするには 3 桁ごとに区切る。2 進数を 16 進数にするには 4 桁ごとに区切る

演習問題 *2-7-2* ： 2 進数→ 8 進数・16 進数

次の 2 進数を 8 進数、16 進数に変換しなさい。

(1)　$1011101_{(2)}$
(2)　$1111000011110000 1111_{(2)}$
(3)　$10101101011010101_{(2)}$

2.7.3　8 進数値、16 進数値から 2 進数値への変換

8 進数値、16 進数値から 2 進数値への変換も簡単です。各桁をそれに対応する 2 進数に変換すればよいのです。

$123_{(8)}$ と $123_{(16)}$ は次のように 2 進数に変換されます。

1	2	$3_{(8)}$		1	2	$3_{(16)}$
001	010	$011_{(2)}$		0001	0010	$0011_{(2)}$

したがって、$123_{(8)} = 1010011_{(2)}$、$123_{(16)} = 100100011_{(2)}$ となります。

ポイント 2-11 ： 8 進数・16 進数→ 2 進数変換方法

8 進数、16 進数を 2 進数にするには、各桁を対応する 2 進数に変換する

演習問題 *2-7-3* ： 8 進数・16 進数→ 2 進数

次の数を 2 進数に変換しなさい。

(1)　$156_{(8)}$
(2)　$FFAB_{(16)}$

2.7.4　8 進数・16 進数の相互変換

8 進数値から 16 進数値、あるいはその逆の変換は、いったん 2 進数に変換してから行えばよいでしょう。たとえば、$123_{(8)}$ を 16 進数に変換するには

1	2	$3_{(8)}$	
001	010	$011_{(2)}$	$123_{(8)}$ を 2 進数 $001010011_{(2)}$ に変換する
0	0101	$0011_{(2)}$	2 進数 $001010011_{(2)}$ を 4 ビットで区切る
0	5	$3_{(16)}$	16 進数に変換する

となり、$123_{(8)} = 53_{(16)}$ となることがわかります。

また、16 進数の $123_{(16)}$ を 8 進数に変換するには

```
  1      2      3(16)
0001   0010   0011(2)   123(16) を 2 進数に変換する
 ↓      ↓      ↓      ↓
 000    100    100    011(2)   2 進数を 3 ビットで区切る
  0      4      4      3(8)          8 進数に変換する
```

となり、$123_{(16)} = 443_{(8)}$ であることが分かります。

ポイント 2-12 ： 8 進数・16 進数変換方法

8 進数、16 進数の相互変換は、2 進数を介する

演習問題 *2-7-4* ： 8 進数・16 進数相互変換

次の 8 進数を 16 進数に、16 進数を 8 進数に変換しなさい。

(1) $1234_{(8)}$
(2) $ABCD_{(16)}$
(3) $7777_{(8)}$

2.7.5 n 進数の 10 進数への変換

n 進数 $m_5m_4m_3m_2m_1m_0$ は、10 進数では次のように表現できます。

$$m_5 \times n^5 + m_4 \times n^4 + m_3 \times n^3 + m_2 \times n^2 + m_1 \times n^1 + m_0 \times n^0$$

2 進数、8 進数、16 進数の場合には、次のように表現できます。

2 進数 ： $m_5 \times 2^5 + m_4 \times 2^4 + m_3 \times 2^3 + m_2 \times 2^2 + m_1 \times 2^1 + m_0 \times 2^0$
8 進数 ： $m_5 \times 8^5 + m_4 \times 8^4 + m_3 \times 8^3 + m_2 \times 8^2 + m_1 \times 8^1 + m_0 \times 8^0$
16 進数 ： $m_5 \times 16^5 + m_4 \times 16^4 + m_3 \times 16^3 + m_2 \times 16^2 + m_1 \times 16^1 + m_0 \times 16^0$

つまり、各桁のこの重み付けを計算すればよいのです。
たとえば、$10110_{(2)}$、$245_{(8)}$、$3EA2_{(16)}$ は次のようになります。

$$\begin{aligned}
10110_{(2)} &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\
&= 1 \times 16 + 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1 \\
&= 16 + 4 + 2 = 22 \\
245_{(8)} &= 2 \times 8^2 + 4 \times 8^1 + 5 \times 8^0 = 128 + 32 + 5 = 165 \\
3EA2_{(16)} &= 3 \times 16^3 + E \times 16^2 + A \times 16^1 + 2 \times 16^0 \\
&= 3 \times 4096 + 14 \times 256 + 10 \times 16 + 2 \times 1 = 12288 + 3584 + 160 + 2 = 16034
\end{aligned}$$

16 進数の場合には、A、B、C、D、E、F の各数字を使うので、これらは、それぞれ 10、11、12、13、14、15 の値に変換して計算します。また、実際に計算をする場合には、上記のように数式の形で書かないで、重み付けの部分だけを書いた方が効率が良いでしょう。

その方法で、$10101_{(2)}$、$245_{(8)}$、$3EA2_{(16)}$ を 10 進数に変換する場合には、次のようにします。

2 進数の場合

...	2048	1024	512	256	128	64	32	16	8	4	2	1	
								1	0	1	0	1	
								16	+	4	+	1	= 21

8 進数の場合

...	32768	4096	512	64		8		1		
				2		4		5		
				128	+	32	+	5	= 165	

16 進数の場合

...	1048576	65536	4096		256		16		1		
			3		E		A		2		
					14		10				
			12288	+	3584	+	160	+	2	= 16034	

ポイント 2-13 ： 2 進数・8 進数・16 進数を 10 進数に変換

2 進数、8 進数、16 進数を 10 進数にするには、各桁の重みを合算する

演習問題 *2-7-5* ： 8 進数・16 進数相互変換

次の数を 10 進数に変換しなさい。

(1) $110101011_{(2)}$　(2) $12345_{(8)}$　(3) $FC_{(16)}$

総合問題 2

[演習 **2-1**] 次の 10 進数値を、指定された進法に変換しなさい。

(a) 2 進数　(1) 5　(2) 10　(3) 45　(4) 650　(5) 1025

(b) 8 進数　(1) 7　(2) 125　(3) 3333　(4) 10123　(5) 100000

(c) 16 進数　(1) 100　(2) 2345　(3) 1025　(4) 64000　(5) 100000

[演習 **2-2**] 次の数値を 10 進数に変換しなさい。

(1) $1011010101_{(2)}$　(2) $726_{(8)}$　(3) $37337261_{(8)}$

(4) $37337261_{(16)}$　(5) $CAFE_{(16)}$　(6) $12BC34F_{(16)}$

[演習 **2-3**] 次の数値に 1 を加えた数をそれと同じ進法で答えなさい。

(1) $1011011_{(2)}$　(2) $101011000_{(2)}$　(3) $372177_{(8)}$　(4) $323712_{(8)}$

(5) $AB012_{(16)}$　(6) $FD01FFE_{(16)}$　(7) $12ABC_{(16)}$　(8) $FFFFFF_{(16)}$

[演習 **2-4**] 次の 2 進法の加算を行いなさい。

(1) $110101_{(2)} + 10111_{(2)}$　(2) $11111_{(2)} + 100_{(2)}$　(3) $10110101_{(2)} + 110110_{(2)}$

[演習 **2-5**] 10 進数 80000 を 2 進数で表現し、その下位 16 ビットだけを抜き取ったとき、その値はいくつですか?