大学1年次コンピュータリテラシ科目でのアセンブリ言語プログラミング体験

久野　靖[1]　江木啓訓[1]　赤澤紀子[1]　竹内純人[1]　笹倉理子[1]　木本真紀子[1]

概要：電気通信大学では 1 年次の前期に「コンピュータリテラシ」を必修で開講している。2017 年度から科目の運営をリフォームし、さまざまな実習を多く行ってもらうように工夫している。その中で「コンピュータの動作原理」を学ぶ回の実習材料として仮想的な (簡潔な命令セットを持つ) コンピュータのシミュレータをアセンブリ言語で記述するものを取り入れた。本発表ではその経験について報告する。

Experience of Assembly Language Programming in College 1st-Grade Computer Literacy Classes

Kuno Yasushi[1]　Egi Hironori[1]　Akazawa Noriko[1]　Takeuchi Sumito[1]　Sasakura Michiko[1]
Kimoto Makiko[1]

1. はじめに

電気通信大学は東京都調布市にある理工系の単科大学である。その 1 年次情報基礎教育では、前期に「コンピュータリテラシ」後期に「基礎プログラミングおよび演習」を必修科目として開講している。

「コンピュータリテラシ」は「コンピュータの基本的な構成と Unix という OS の基本を学び、情報倫理、情報セキュリティについて理解すること、および、実際にコンピュータを道具として使いこなせるようになること」を達成目標としており、Unix のファイル操作、Emacs/LaTeX による文書作成などに重点があることが特徴である。

2016 年度までの科目運営では、ファイル、Emacs、LaTeX の操作などに多くの時間が割かれていた。そして、コンピュータの基本原理のようなことは、LMS 上に資料として掲載されていて教員が説明するだけであり、実習は含まれていなかった。しかし実習など手を動かす部分がないと、学生が内容を十分理解するのは難しいと思われた。

また、実習が内容として含まれている部分についても、説明に時間を要するため、結果として授業時間には実習がほとんどできず、課外にやるように指示しても実際にはうまく実習ができていないことが多く見受けられた。

2017 年度からは、テキストを通読できる冊子形式で用意した上で [2]、解説ビデオも提供して予習を義務づけ、授業時間には実習を中心に行なうように運営方法を改めた。[*1]

これに従い、コンピュータの基本原理の回は実際に CPU が動作する様子を (模擬的にではあるが) 体験してもらうため、アセンブリ言語でプログラムを入力し、それをその場で仮想的なマシン「小さなコンピュータ」の機械語に変換して模擬実行するシミュレータの実習を取り入れた。本稿ではその概要ならびに実施結果について報告する。

2. コンピュータリテラシ科目の構成と概要

「コンピュータリテラシ」は前述のように 1 年前期の授業であり、2017 年度の受講人数は 780 名である。1 年次の必修科目であるが、再履修学生や既習得単位認定者がいるため、受講人数は学年人数と一致しない。13 クラスに分かれて開講しており、12 クラスが昼間 (59〜65 名)、1 クラス

[1] 電気通信大学, 182-8585 東京都調布市調布ケ丘 1-5-1
University of Electro-Communications, 1-5-1, Chofu-Gaoka, Chofu, Tokyo 182-8585 Japan

[*1] 扱う内容や題材は主に筆者がこれまでに開発してきた授業やテキスト [5][4] を土台にしている。後期についても同様の予定である。[3]

表 1　2017 年度の各回内容

回	内容
1	コンピュータの利用と認証
2	インターネットの原理
3	ネットワークと安全性
4	コンピュータの動作原理
5	ファイルシステムとファイル操作
6	コンピュータシステムと OS
7	フィルタとシェルスクリプト
8	ソフトウェア開発とテストケース
9	テキストファイル/Emacs の詳細
10	マークアップによるテキスト整形
11	グラフィクス/図と表
12	アカデミックリテラシ (総合課題)
13	HTML/CSS による Web ページ記述
14	Web と情報アーキテクチャ
15	Web サイトの設計/製作 (総合課題)

が夜間課程 (35 名) である。

授業はすべて本学教職員が 1 名ないし 2 名で担当し、大学院生の TA が 2 名つく。授業運営には情報基盤センターの仮想マシン上に設置している Moodle を使用しており、学生は授業時には演習室から、課外には学内や学外の PC やスマフォ・タブレットから Moodle に接続して利用する。このほか Unix の実習はセンターの Linux システムに学内・学外の PC から接続しておこなう。

Moodle 上の教材は標準的な回で次のようになっている。

- その回のテキスト PDF。なお、テキストは全回をたばねた冊子の PDF も提供し、さらにそれを印刷したものを生協で頒布している。
- 予習用の動画。動画本体は YouTube に置き、そこへのリンクを設置している。
- 予習確認問題。Moodle の小テスト (quiz) 機能を使用し、授業冒頭の 10 分間で実施する。予習内容の理解を評価する (成績には入らない)。
- 報告課題 (activitiy-report)。Moodle の課題 (assginment) 機能を使用し、その授業時の各自の活動を簡単に報告し、アンケートに回答する (成績には入らない)。
- 提出課題 (assignment)。Moodle の課題機能を使用し、各回とも指定した課題をレポートとして提出する (アンケート回答も併せて提出する)。提出課題は成績評価に使用する。
- 実習用ページやその他の付録ページ。各回の実習に必要なものがあれば適宜提供する。またシステムの使用方法などを学生が参照できるように記述したものも付録として提供している。

初回は冒頭が新入生テストとなること、総合課題回 (後述) は予習確認がないことなど回により多少の違いがある。このほか、クラス単位の質疑応答用フォーラム、アナウンス用フォーラム、全クラスアナウンス用のフォーラムなども使用している。

授業は毎週 90 分で、標準的な回では冒頭 10 分で予習確認、最後の 10 分で報告課題の記入をおこない、間の 70 分が授業本体となる。各回の授業内容を表 1 に示す。総合課題回は通常よりテキスト内容が少なく、提出課題 (レポート) がグループワークを要求するものとなっていて、授業時にグループワークで課題を行なう。

成績評価は提出課題と期末試験を 50:50 としている。提出課題は各回テキストにある課題 (実習により何かを調べたり検討して結果を報告するもの、小課題 9 個程度を掲載) から 1 個以上を選んで提出することを求めている。期限は次回授業の前日 23:59 までとしている。

提出課題の評価は担当教員がおこなうが、基本的に指定された内容を満足していれば「B(3 点)」とし、すべて B のとき 50 点となるようにしている。要件を満たさないことがあれば C(2 点)、とくに評価できることがあれば A(4 点) で、A は各クラス各回とも 0〜数件とするように依頼している (総合課題回はやや配点が異なる)。これは評価作業の負担を軽くすることと、学生にとってもとくに頑張らなくても普通にやれば満点をつけますよというメッセージとすることを意図している。

試験はまだ実施していないが、Moodle を用いて実施し、試験内容は各回の予習確認問題のうち試験範囲に含まれるものの類題とする予定である。試験範囲はテキストの全内容の 50%程度の節にその旨を表示している。これは学生の負担を減らすことと、テキストの中でも操作の説明のような部分や高度な部分 (興味を持つ学生向け) は試験に含めないようにしたことによる。

3. アセンブリ言語プログラミング体験

本科目の内容には「コンピュータの構造と動作原理」が以前から含まれているが、筆者らが授業の様子を見た範囲では、一般的な CPU やメモリの働きなどを説明されても、多くの学生には具体的なイメージや理解を得ることは難しいように感じられていた。[*2]

このため、2017 年度からこの内容に対応して、単純で理解しやすい仮想的な命令セット/命令形式を持つ「小さなコンピュータ」を設計し、Web ブラウザ上で動作する (図 A·1) シミュレータを用いて実習することで、コンピュータの動作原理を理解することを目指した。この内容は＃ 4 の中に含まれる。

＃ 4 の位置づけは表 1 にあるように、冒頭 3 回の導入 (必須事項や安全教育) が終わったあと、ハードウェアから OS、アプリケーション、サービスのように順次階層をさか

[*2] 試験問題にこの内容を問う設問が含まれていなかったため、実際にどうであるかを示す客観的なデータは残っていない。

のぼって内容を網羅していく初回に相当する。アセンブリ言語プログラミングを必要とする内容は以後の回にはないので、あくまでも CPU の原理を知る題材とプログラミングに接するきっかけとなることが目的である。

機械語を 16 進などで入力するのはハードルが高すぎると思われたため、アセンブリ言語も併せて設計し、アセンブリ言語ソースを入力欄に打ち込んで RUN ボタンを押すとアセンブルされて (エラーがなければ) 直ちに実行される、という形にした。

このことはまた、本科目以前にプログラミングの経験がない学生にとっては、このアセンブリ言語によるプログラミングが最初のプログラミング経験となる、ということを意味する。

先に述べたように、後期にはプログラミングの理解と習得を目標とした科目があるわけだが、我々としてはそこに至る前に前期の科目でも少しだけプログラミングを経験してもらうことが、後期の科目の導入としても有効であると考えている。

ただ、今日ではプログラミングはほぼ常に高水準言語で行なうものであり、アセンブリ言語が実用に使われる場面は極めて少ない。そのような言語を体験してもらうことについては、次のように考えた。

- 仮想的な CPU の構造や動作をできる限り簡潔にすることで、プログラムがどのように動作するかというイメージを高水準言語よりも持ちやすくできる。さらに、CPU の動作と直接対応することから、CPU の動作を理解する上でも効果的である。
- 記述が 1 行単位なので、複雑な構文を扱ったりその間違いに対処する必要がない。
- 簡単な問題であっても、コード記述の自由度や多様性が極めて大きい。コード記述に多様性があることを身をもって体験し、また自分で選択肢から選んでコードを組み立てる体験をすることは、後で高水準言語に進んだ場合にも大きく役立つと考える。

上記の考えから、「小さなコンピュータ」ではレジスタ Acc だけを計算に用いることにした。ただし、データの並びを扱いたいので、アドレス修飾をするレジスタ Idx を追加している。命令一覧を表 A·1 に示した。[*3][*4]

入出力は複雑になるため扱わず、処理するデータはすべてアセンブリ言語上で定数として与え、結果も適当なラベルの位置に格納させて停止後にメモリ内容表示を見て確認することとした。この方が簡潔であり、また load 命令や store 命令に対する理解が深まる効果もある。

[*3] 値をメモリから取り出す命令と直値で取り出す命令は本来は違う命令だが、直値ビットで区分を表し、アセンブラ上ではオペランドが数値か名前 (=ラベル) かをこの区分に対応させている。

[*4] http://www.edu.cc.uec.ac.jp/~ka002689/smallcomp.html

4. 学習内容と授業の様子

学習内容を記述したテキストを付録につけた。この回はこの内容の前に 2 進法・2 の補数表現、CPU の構成と動作 (メモリ、レジスタ、演算回路) の説明があり、また後に CPU 性能の向上の話題があるが、予習を前提としていることから、各教員には多くの時間をアセンブリ言語プログラミングの実習にあてるように依頼した。

なお、内容のうち前半は条件分岐まで、後半はループと Idx レジスタであるが、試験範囲は前半までであり、教員にも学生にそのことを告知の上、まず前半の実習を十分やり、後半は比較的軽く扱うように依頼している。

これは、本科目全体として学生により背景や理解度が大きく異なるので、全員が同じ問題をやるのではなく、興味・関心があり易しい問題にあきたらない学生がより高度な問題に取り組めるようにしているためである。この回の課題は付録には 6 小問が掲載されているが、この後に「命令の実行速度を測る (シミュレータ、本物の CPU)」「ここまでに使ってみていない命令を使う」というチャレンジ問題 3 問があり、「9 小問から 1 問以上」の提出を課している。

授業の様子を見ると、ほとんどの学生は予習では意味を理解せず、シミュレータの画面を開かせても何のことかわからず説明を聞いている状態だった。[*5]

しかし最初の例題 (X と Y を足して Z に格納し停止) を説明してからその通り実行させ、これに基づいて「1a: 5 つの数を合計する」「1b: 1 つの数を 5 倍する (実際には足し算で計算)」の演習をするように促すと、ほぼ全員が様々に工夫しながら取りむようになった。何をしてよいか分からない学生には TA と教員でヒントを出すなどして対応した。

さらに、2 数のうちの最大を求める例題の説明を聞いてから (実際には説明を聞かずにテキストを呼んで取り組む学生も多数)「1c: 3 数の最大を求める」段階では、多くの学生が真剣に悩んでいる様子がうかがえた。この課題に対する質問が授業時には最も多かったように感じる。

後半のループと Idx については、ほとんど質問はなく、ごく一部の学生だけが取り組んでいたが、これも予定通りであった。

5. レポートによる評価

学生の学習状況を評価するため、提出課題の分析をおこなった。期限 (次回授業前日一杯) までに提出された＃ 4 の提出課題 694 件を分析対象とした。1 つ前の＃ 3 の提出数は 692 件であり、とくにこの回の提出が少ないということはなかった。

すべての課題についているアンケートは、この課題においては表 2 の通りである。主として 2 番目の設問に対する

[*5] ごく一部、予習で課題を全部やったため授業時間が退屈だったという感想も見られた。

表 2　提出課題＃ 4 のアンケート

Q1. コンピュータの動作原理やアセンブリ言語によるプログラムについてどれくらい知っていましたか。新たに知ったことで面白かったことは何ですか。
Q2.「小さなコンピュータ」でプログラムが組めるようになりましたか。
Q3. リフレクション (今回の課題で分かったこと)・感想・要望をどうぞ。

表 3　達成度の分類

分類段階	件数 (百分率)
0. 未回答/達成度不明	10 (1.4%)
1. できなかった/分からなかった	14 (2.0%)
2. 難しい/少ししかできなかった	42 (6.1%)
3. 基本は分かった/基本的なものはできる	258 (37.2%)
4. 分かった/できるようになった	147 (21.2%)
5. 面白い/楽しい/興味を持てた	223 (32.1%)

回答に基づき (ただしアンケートの回答以外の箇所に書かれた自由記述も参考として)、回答者のプログラミングに関する自己評価と楽しさ・面白さを表 3 に示す 6 段階で分類し集計した。これらのうち 1〜2 は negative な記述、3〜5 は positive な記述だといえる。

ここで最上位段階 5 が面白さ・楽しさであるのは、今回の学習の目的がプログラミングを経験してもらうことであり、単にできたことよりも、面白い・楽しいと考えてもえたことの方が、この先の学習という目的から見れば価値があると考えたためである。*6

全体として 3〜5 の positive な回答の合計は 90.5%であり、プログラミングを体験してもらい後期に備えるという当初の目的は達成されていると考えている。

また、レポートにおいて解答されている設問を分類し集計したものを図 1 左に示す (1 問以上を選択して解答するので、合計数は学生数より多い)。いずれも、レポートにプログラムが記載されているか否かの区分も示している。

付録にあるように、設問 1-1、1-2、1-3、2-2、2-3 がアセンブリ言語でプログラムを作成する課題である。設問 3 は付録に掲載していないが、3-1(シミュレーターの 1 命令あたり実行時間を計測してみる)、3-2(シミュレータのまだ使ったことのない命令を動かしてみる)、3-3(本物の CPU の 1 命令あたり実行時間を何らかの方法で計測してみる) の 3 問から成る。これらは advanced な課題であり、解答件数も少ないため、1 つにまとめて示した。

これを見ると、平易な問題である 1-1 と 1-2 が当然ながら多いが、複数の条件分岐を必要とする 1-3(3 つの値の最大) もかなり多く取り組まれていることが分かる (全学生の 1/3 程度)。そして、より複雑なループを必要とする問題である 2-2、2-3 もそれなりに取り組まれている。

さらに、同じ集計を表 3 の分類で 0..2(未解答と negative — 件数が少いので 1 つにまとめた)、3、4、5 それぞれについて分けて行なったものを図 1 右に示す。0..2 ではさすがに 1-1 と 1-2 が大半だが、3(基本的なものはできる) ではやや複雑な 1-3 がそれなりに取り組まれている。そして、4(分かった/できる) では高度な問題の取り組みがはっきりと多い。5(興味深い/面白い) は、4 ほど高度な問題の比率が多くないが、1-3 が 1-1、1-2 と遜色ないくらい解答されており、またレポートにプログラムが掲載されている比率が他群よりも高い。

*6　「できる」「基本的なことはできる」等の記述と「面白かった」等が併記されている場合は 5 に分類した。

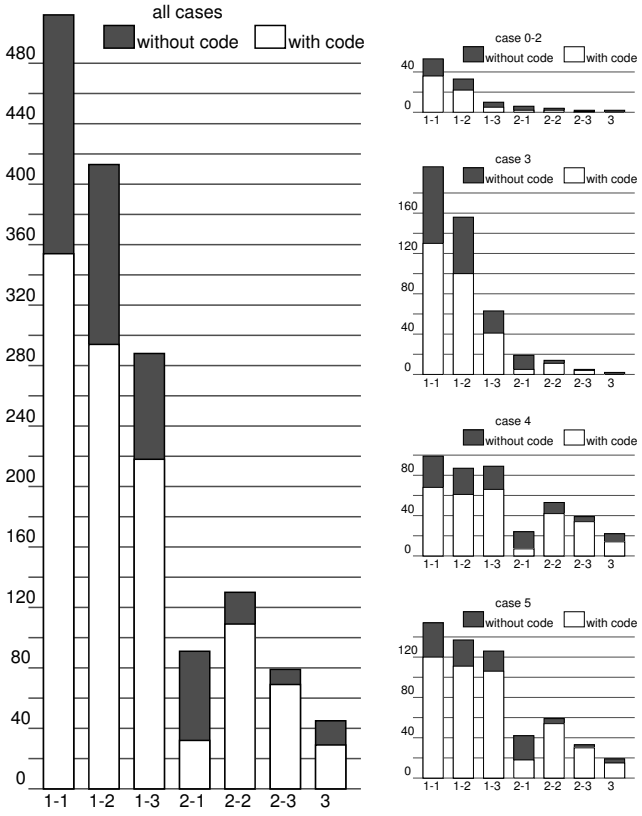

図 1　解答された問題

表 4　件数の多かったコメント

件数	コメント
103	プログラミングは難しいと思った
94	さらに学んで行きたい/もっとできるようになりたい
58	コンピュータの原理が分かった
47	コンピュータ/プログラムの動くようすが面白かった
36	各命令は単純である/単純なものを組み合わせて複雑なものを作れる
21	プログラミングはパズルのようであると感じた
16	高水準言語とは違うと思った/高水準言語の良さが分かった
9	コンピュータは厳密である/正確に書かないと動かないと分かった
7	プログラムが複雑になると設計が重要と分かった/計画が必要だと分かった
7	簡単な計算も多数の命令を必要とすることが分かった
7	試行錯誤するのは楽しい/間違い探しは楽しい
7	同じ結果でも多様な書き方ができると分かった
6	短くしよう/最適化しようと工夫するのが楽しい
6	0 と 1 で全てを表しているのが分かった

で類似内容のものをまとめて累計したものを表 4 に示す。3 番目以降では「コンピュータの原理が分かった」「動くようすが面白い」「単純な命令から複雑な動作が作られる」などコンピュータの基本原理に対応するコメントが多くなっているが、「パズルのようで面白い」「試行錯誤が楽しい」「工夫が楽しい」など、楽しさの内容を記したものも多くなっている。

一方、1 番目と 2 番目は「難しい」「もっと学びたい」(いずれも 100 件前後) であり、「分かった」学生からも「分からない」学生からも出されるような内容である。未解答と negative を合わせても 66 名なので、これらも半数以上は「分かった」学生からのコメントであると分かる。

筆者らの経験では、プログラミング関係では「難しい」は決してマイナスの記述ではなく、難しいものに取り組むから面白い、やりがいがある、という使われ方も多くあると考える。

6. 関連研究

アセンブリ言語によるプログラミングの教育において CPU のシミュレータを用いることは古くから行なわれて来た ([7][6] など)。しかしこれらにおいて、その目的は実際にアセンブリ言語で仕事ができる技術者を育成することであり、そのため対象となる命令セットアーキテクチャは (実在の CPU であっても仮想的なものであっても) ある程度の機能を備えた複雑なものであった。

アセンブリ言語プログラミングというよりは、コンピュータのハードウェアについて理解するためのシミュレータも複数作られて来ている ([1] など)。これらもハードウェアの様々な要素を学ぶ必要があるため、やはりその命令セットアーキテクチャには多くの要素が含まれている。

このため、上記いずれの場合においても、アセンブリ言語によるプログラミングの学習が 1 時間で済み、学生がすぐにプログラムを組むということはほぼ考えられない。

これに対し、本稿で提案している教育内容はコンピュータの構成について「CPU とメモリ」程度の簡単なモデルの説明にとどめ、またレジスタの数も最低限とすることで、短時間での習得とそれに続くプログラミング体験を可能にしている (実際にそれが有効に機能していることは前節までで説明した)。このような指針による教材とカリキュラムは筆者らが調べた範囲では、他には見当たらなかった。

7. 議論とまとめ

本稿で報告した授業は原稿執筆時点でまだ進行中であり、試験も実施していないので、positive に解答した学生が実際にどれくらいアセンブリ言語プログラミングの内容を身につけているかは分からないところである。

しかしそれでも、多くの学生が前向きに解答し、コメントにもコンピュータの原理やプログラミングの面白さなどに関するものが多く記されている様子から見て、本内容はある程度当初の目的 (コンピュータの原理を学びつつプログラミングを体験しその楽しさを知ってもらうことで後期につなげる) を達成できているものと考える。

学生からのコメントへの対応であるが、1 回限りの内容であり、以後アセンブリ言語を使うことはないので、その意味でのフォローアップはない。ただし、この後に高水準言語の回があり、また後期にはプログラミングを系統的に学ぶ科目があることから、関心を持った学生への対応としては十分あると考えている。

反省点としては、導入時にもう少し丁寧な説明を用意することで、negative な解答 (8%) を減らせなかったかということがある。これについては今年が初年度であり、来年度以降の教材や進め方を手直しして行きたい。

謝辞

筑波大学ビジネスサイエンス系の大木敦雄氏には、「小さなコンピュータ」の開発や初期の教材としての活用において意見やアドバイスを頂いた。また電気通信大学「コンピュータリテラシ」担当の先生がたにも、多くのコメントやアドバイスを頂き、また実際の科目運営を担っていただいている。ここに感謝します。

参考文献

[1] 今井 慈郎, 富田 眞治, 古川 善吾, 井面 仁志, 白木渡, 石川浩, 大和田 昭邦, 計算機システム教育のためのビジュアルシミュレータ VisuSim, 情 報処理学会研究報告コンピュータと教育 (CE), 2000-CE-059, pp. 77-84, March 2001.
[2] 久野　靖, コンピュータリテラシ 2017 (学内冊子), 電気通信大学共通教育部情報部会, 2017.
[3] 久野　靖, Ruby による情報科学入門, 近代科学社, 2008.
[4] 久野　靖, 改訂 2 版 UNIX による計算機科学入門, 丸善, 2004.
[5] 久野　靖, UNIX の基礎概念, アスキー, 1995.
[6] 野津 直貴, 池内 雄馬, 竹澤 真弘, 内田 智史, 情報処理技術者教育のた めのハードウェアシミュレータの提案, 情報処理学会研究報告コンピュー タと教育 (CE), 2008-CE-093, pp. 133-139, February 2008.
[7] Kenneth Vollmar, Pete Sanderson, MARS: an education-oriented MIPS assembly language simulator, ACM SIGCSE'06, pp. 239-243, March 2006.

付　　録

A.1 テキスト: 小さなコンピュータ

A.1.1 小さなコンピュータのシミュレータ

本ものの CPU の命令は複雑なので、ここでは簡単化した (架空の)「小さなコンピュータ」を想定して、その命令を動かしてみましょう。この小さなコンピュータは JavaScript 言語で記述されていて、ブラウザ上で動作します (図 A·1)。とりあえず使ってみましょう。

ここで「プログラム (program)」と記された欄に、1 行

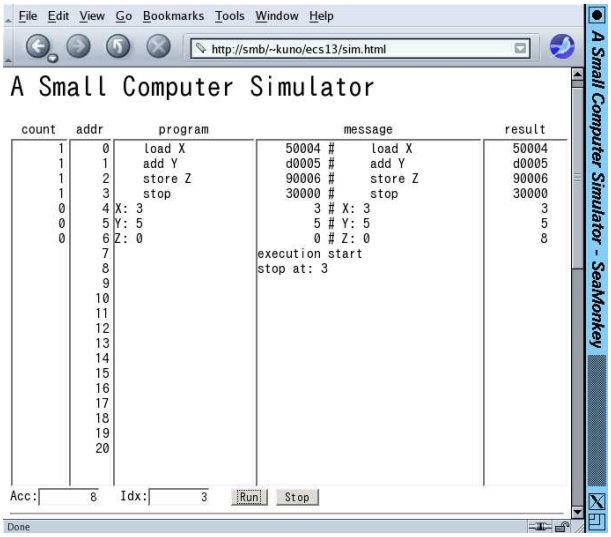

図 **A·1** 「小さなコンピュータ」の画面

に 1 つずつ、命令を書いて行きます。たとえば、次の 7 行のコードを打ち込んでから「実行 (run ボタンをクリック)」してみましょう。なお、**コード** (code) というのは、「プログラムないしその断片」を表す一般的な用語です。

```
    load X
    add Y
    store Z
    stop
X: 3
Y: 5
Z: 0
```

ここで「load」は、数値をメモリの指定場所 (この場合は X という名前のついている番地) からアキュムレータ (Acc) というレジスタに取り出して来る命令 (図 A·2 上)、「add」は、メモリを指定場所 (この場合や Y という名前のついている番地) から取り出し、それを Acc の内容に足し込む命令、「store」は、Acc の内容をメモリの指定場所 (この場合は Z という名前のついている番地) に格納する命令です (図 A·2 下)。これらの命令はいずれも「どこからどこへ」のように 2 つの場所を本来は指定する必要がありますが、その一方は Acc になっているので場所を 1 つ指定すれば済むのです。

そして最後の「stop」は、その名前通りプログラムの実行を停止する命令です。この命令が無いと、コンピュータは次の番地の内容を命令だと思って実行してしまいます (上の例だと次には「3」「5」等が入っていて、これらを実行して行き、正しく止まりません)。

プログラムの後には、X、Y、Z という名前のついた場所を用意し、それぞれに 3、5、0 という値を入れておきます。そうすると、プログラムを実行した結果、X と Y の値を足した結果 (8 ですね) が Z の場所に格納され、確かに足し算

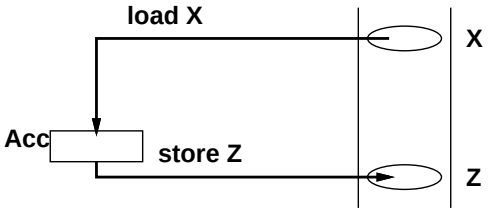

図 **A·2** ロード命令とストア命令

ができていることが分かります。

実行開始時には、**プログラムカウンタ** (program counter、PC) というもう 1 つのレジスタ (画面には見えていない) に 0 を入れてから開始します。CPU は PC が指す番地 (最初は 0 番地) から命令を取り出し、PC をその命令の次の番地に変更します。最初の命令を実行し終ったらまた PC の指している番地から命令を取り出し、PC を次の番地にします。これを繰り返して実行が進みます。

なお、表示の見かたですが、message の欄はコードを命令に翻訳した様子や実行開始/停止の情報が表示されます。result の欄はプログラムが停止したときのメモリの内容が表示されます。

私たちが普段使っているコンピュータでは、画面やマウスなどを使ってデータをやりとりしますが、この「小さなコンピュータ」ではごく基本的な命令しか用意していないので、このようにメモリに直接データを用意して動かすようにしています。

実は、このように命令やメモリの場所に名前をつけて表すプログラムの書き方を**アセンブリ言語** (assembly language) と呼びます。アセンブリ言語のプログラムは**アセンブラ** (assembler) と呼ばれるプログラムによってビット列、つまり 0 と 1 だけから成るプログラムに変換され、CPU はそれを実行します。この、CPU が直接実行する形のプログラムのことを**機械語** (machine language) のプログラムと呼びます。「小さなコンピュータ」では、実行開始時に message 欄に機械語 (アセンブラの変換出力) を表示するようになっています。

上の例は「足し算」でしたから順番に命令を 4 つ実行すれば終わりでした。しかし実は、プログラムでは「計算した結果によって処理を切り替える」ということが可能であり、これによって複雑な処理が行えます。そのために、命令の中に「分岐 (ジャンプ) 命令」「条件分岐命令」があります。具体的には、通常の命令はその命令を実行し終わると「次の」命令に進むのに対し、分岐命令は番地を指定し、次はその番地の命令の実行に進むようにさせます。そして条件分岐命令は、「Acc が 0 でないならば」のように条件を指定して、その条件が成り立っている時だけ分岐します (成り立っていなければ、次の命令に進む)。

たとえば、今度は X と Y のうち「より大きい値を」求めてそれを Z に入れることを考えます。そのためには、X

から Y を引いてみて、マイナスなら Y の方が大きいと分かります。この考えに基づいてプログラムを作ってみましょう。

```
      load X
      store Z
      sub Y
      ifp Skip
      load Y
      store Z
Skip: stop
X:  3
Y:  5
Z:  0
```

「sub」は引き算の命令です。このプログラムではまず、X を Acc に取り出し、とりえあず Z に入れます。次に、Acc の内容から Y を引き算します。ここで、もしプラスなら X の方が大きくて OK ですから、Skip という場所に「飛びます」(ifp は Acc の内容がプラスなら指定した場所に分岐する条件分岐命令です)。プラスでないなら、もう 1 回 Y の値を Acc に持って来て、Z に格納します。いずれにせよ Skip の所に合流して、そこでプログラムは止まります。このように、条件判断して自動的に処理を切り替えることで、コンピュータは複雑な処理が行えているのです。

演習 1　「小さなコンピュータ」でここまでに出て来た例題をそのまま動かしてみなさい。うまくできたら、以下の処理を実行するプログラムを作成してみなさい。

a. 5 つの値を A、B、C、D、E というラベルの番地に入れておき、その合計を Z というラベルの番地に入れて止まる。データ例: 1, 2, 3, 4, 5 → 結果 15 (16 進では F)。

b. A というラベルのついた番地に入れておいた数値を 5 倍し、結果を Z というラベルのついた番地に入れて止まる。データ例: 4 → 20 (16 進では 14)。

c. 3 つの値を A、B、C というラベルのついた番地にそれぞれ入れておき、その 3 つの最大値を求めて、Z というラベルのついた番地に入れて止まる。データ例: 3, 6, 2 → 結果 6。

いずれにおいても、プログラムがどのように動作するか説明すること。

A.1.2　ループのあるプログラム

先の課題では、5 つの値の合計とか、5 倍とかなので、その数だけ足し算命令を使えば済んでいました。

では、合計に戻って、もっと沢山の数を合計したければどうしましょうか? A、B、C…のように沢山変数を並べていてはプログラムが長くなって大変そうです。そこで、Acc のほかにもう 1 つ、インデックスレジスタ (Idx) というものが用意されています。そして、load 命令の拡張版

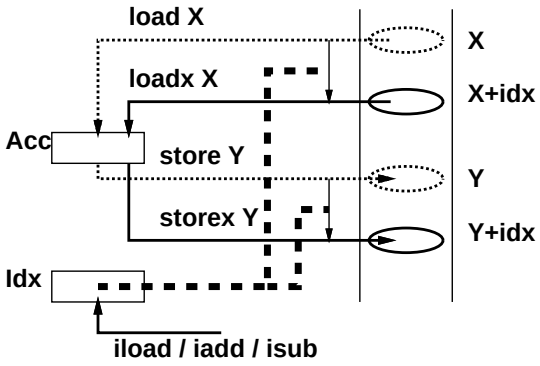

図 **A·3**　Idx を使うロード命令とストア命令

表 **A·1**　「小さなコンピュータ」命令一覧

名前	コード	命令の動作
nop	00,01	何もしない
stop	02,03	プログラムの実行を停止
load	04,05	Acc に値を持って来る
loadx	06,07	〃 (値/番地に Idx を足す)
store	08,09	Acc の値を格納する
storex	0a,0b	〃 (番地に Idx を足す)
add	0c,0d	Acc に値を足す
sub	0e,0f	Acc から値を引く
iload	10,11	Idx に値を持って来る
iadd	12,13	Idx に値を足す
isub	14,15	Idx から値を引く
ifz	16,17	Acc ＝ 0 なら分岐
ifnz	18,19	Acc ≠ 0 なら分岐
ifp	1a,1b	Acc ＞ 0 なら分岐
ifn	1c,1d	Acc ＜ 0 なら分岐
jump	1e,1f	無条件に分岐
neg	20,21	Acc の符号を反転

loadx では「指定した番地より Idx の値だけ先の場所」からデータを取り出すことができます (図 A·3)。これを使って、並んだデータを順番に取り出し、合計して行けばよいのです。プログラムを見てみましょう。

```
      iload 0
Loop: loadx Data
      ifz End
      add Sum
      store Sum
      iadd 1
      jump Loop
End:  stop
Sum:  0
Data: 1
      2
      5
      0
```

Data というラベルの後に数行ぶんのデータがありますが、これを順番に持って来て合計するわけです。

「iload」命令は Idx に指定した値 (この場合は 0) をロードします。次に、loadx 命令で Acc に値を持って来ますが、最初は Idx は 0 なので、ちょうど Data の場所の値が持って来られます。もしその値が 0 なら、これは終わりの印なので (合計を取りたいのに 0 をデータに入れる必要はないでしょうから)、End へ分岐します。そうでなければ、Acc の値と Sum を加え、その結果を Sum に入れます。続いて、「iadd」命令で Idx を 1 増やしてから、「jump」命令で無条件に Loop へ戻ります。すると、次は Data の次の場所から値が取り出せるわけです。これを繰り返して次々に値を足して行き、0 が現れたら End へ来て止まりますが、そのときには Sum には総計が入っています。

このプログラムの「肝」は、手前 (上) 方向への分岐命令を使って一群の命令列を「繰り返し」実行させることで、短いプログラムでも沢山の処理を行わせられる、というところにあります。これが、今日のコンピュータの重要な原理だと言えます。最後に、この「小さなコンピュータ」が持っている命令の一覧を掲載しておきます (表 A·1)。[*7][*8]

演習 2　「小さなコンピュータ」で上に説明した N 個のデータの合計プログラムを動かし、さらにデータの個数を増やしたり減らしたりして、動作を確認しなさい。うまくいったら、以下の課題から 1 つ以上をやりなさい。

a. 「小さなコンピュータ」では左端の count 欄にそれぞれの命令を実行した回数が表示されます。合計するデータの個数 N と、データを取り出す loadx 命令の実行回数の関係を調べ、なぜそうなるのかを検討しなさい。

b. 正負とりまえぜて複数の整数を与えておき (0 が終わりの印)、それらの数値の「絶対値の合計」を求めるプログラムを作りなさい。データ例:「1 -2 3 -4 5 0」→結果例: 「15」

c. 2 つの整数 (いずれも 0 以上) を与えておき、その 2 つの積 (掛けた結果) を求めるプログラムを作りなさい。データ例:「X: 3, Y: 5」→結果例:「15」。ヒント: この問題では Idx レジスタは使う必要がありません。つまり iload や lodax 命令は使う必要がありません。

b と c については、プログラムがどのように動作しているか説明すること。

[*7] 条件分岐命令が沢山ありますが、その覚え方は次のようになります。ifz〜「if zero」、ifnz〜「if not zero」、ifp〜「if positive」、ifn〜「if negative」

[*8] 「コード」はその命令を表現するビット列です。すべて 2 つずつコードがありますが、大半の命令はどちらでも動作は同じです。値を取り出す命令 (add、sub、mul、load) のみ、「命令の後に指定した数値を取り出す」「命令の後に指定した名前をつけたメモリの場所から取り出す」の 2 通りに分かれています。