情報処理学会論文誌　教育とコンピュータ　Vol.4　No.2　23–36 (June 2018)

論　文

# コンピュータサイエンス入門教育の題材としてのアセンブリ言語プログラミング

久野 靖[1,a)]　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]

受付日 2017年9月18日，再受付日 2017年12月18日，
採録日 2018年3月3日

**概要**：コンピュータサイエンス入門教育の中で，コンピュータの動作のしくみを学ぶ部分は，講義だけで実感を持ってもらうことは難しい．電気通信大学の 1 年次教育では，半期（15 回）の授業のうち 1 回（90 分）をこの内容にあてているが，筆者らは 2017 年度から，そこに題材としてアセンブリ言語プログラミングを導入した．アンケートによる主観評価の結果，多くの学生にコンピュータの仕組みを理解してもらうことができ，またプログラミングに興味・関心を持ってもらうことができたと分かった．また，期末試験による客観評価でも，約 67%の学生がプログラミング作成問題で 1 問以上正解ないし部分正解を得ていた．

**キーワード**：コンピュータの原理，プログラミング入門，アセンブリ言語プログラミング，アセンブリ言語シミュレータ，短冊型テスト

# Assembly Language Programming as a Contents of Introductory CS Education

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

Received: September 18, 2017, Revised: December 18, 2017,
Accepted: March 3, 2018

***Abstract:*** Among introductory computer science education, topics on CPU operation principles are difficult to be understood with solid sense. In University of Electro-Communications' 1st-grade introductory CS subject (15 weeks), we assign 1 week (90 min.) to those topics. In school year 2017, we have started to use assembly language programming as a learning material. Our enquiry shows that many students understood how CPU operates, and also gained interests in programming. Additionaly, results of end-of-the-term tests show that about 67% of the students actually could answer program-coding problems.

***Keywords:*** principle of CPU operation, introduction to programming, assembly language programming, assembly language-level simulator, split-paper testing

## 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，

[1] 電気通信大学
University of Electro-Communications, Chofu, Tokyo 182–8585, Japan
[a)] y-kuno@uec.ac.jp

学生が内容を十分理解するのは難しいと思われた．

ファイル，Emacs，LaTeX の操作などの部分については，LMS 上の資料では知識の説明に加えて具体的な実習の指示や実習課題が示されていて，カリキュラム上は実習を行うことになっていた．しかしそのような箇所であっても，知識の説明部分に講義時間を多く要したため，結果として授業時間内には実習がほとんど行えないことが多くあった．そのような場合，教員は課外に実習を行うよう指示していたが，経験したことがない操作を時間外に個人でやることはハードルが高く，結果として実習がなされていないことも多くあった．

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うえで [4]，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．本稿はその概要ならびに実施結果について報告するものである．

以下，2 章で関連研究について整理した後，3 章で科目の構成と概要および今回使用したアセンブリ言語とシミュレータについて，4 章で学習内容と授業の様子について説明する．続いて，5 章で授業時アンケートによる主観評価および期末試験における客観評価の結果を示し，6 章で議論とまとめを行う．

## 2. 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，古くから行われて来た（文献 [9], [11] 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ムを用いて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きている（文献 [1], [2] など）．これらもハードウェアの多様な要素を学ぶため，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，複雑なものとなっている．

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．

東京農工大学工学部数理情報工学科では 1980 年代に，アセンブリ言語やハードウェアの教育というより，コンピュータで実現可能なことの限界を理解させ，コンピュータシステムに対する知的好奇心を持たせることを目的として，EDSAC[*2] の機械語を用いた教育を行っていた [3], [10]．この実践が本文で報告している試みと最も目的が近い．

しかし農工大の場合も，専門学科の教育としてハードウェアからソフトウェアまで全体を視野に入れ，多くの時間をかけて実施したものであるので，その点では本稿で提案している内容とは大幅に異なっている．

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は「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，短時間での習得ならびにプログラミング体験を可能にし，結果としてコンピュータの原理理解とプログラミングへの興味という 2 つの目標を達成させようするものである．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．

## 3. 科目の構成とアセンブリ言語の教育内容

### 3.1 前期基礎科目の構成と概要

前期の情報基礎科目「コンピュータリテラシ」は，2017 年度の受講人数が 780 名であった．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，12 クラスが昼間（59〜65 名），1 クラスが夜間課程（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 LMS として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 から接続して行う．

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．予習内容の理解を自己評価させることが目的である（成績には含めない）．
- 報告課題（activitiy-report）．Moodle の課題（assignment）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（成績には含めない）．
- 提出課題（assignment）．Moodle の課題機能を使用

[*1] 扱う内容や題材は主に筆者の 1 人がこれまでに開発してきた授業やテキスト [7], [8] を土台にしている．後期についても同様に予定している [6]．

[*2] 世界で最初に稼働したプログラム内蔵方式のコンピュータ．多くの良質な機械語プログラムが知られている．

表 1　2017 年度の各回内容
**Table 1**　The curriculum of school year 2017.

| 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） |

し，各回とも指定した課題をレポートとして提出する（アンケート回答もあわせて提出する）．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行い，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行うが，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，学生に対して，それほど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．

表 2　新入生テストの設問「プログラミングを学んだか」の回答
**Table 2**　Answer to the query “have you learned programming?”.

| 選択肢 | 件数 | 百分率 |
|---|---|---|
| 高校までの授業で学び理解している | 33 | 4.5 |
| 高校までの授業で学んでいないが理解している | 19 | 2.6 |
| 高校までの授業で学んだが理解していない | 110 | 15.2 |
| 高校までの授業で学んでいないし理解していない | 545 | 75.1 |
| 無解答 | 19 | 2.6 |
| 合計 | 726 | 100.0 |

試験は授業と同様にクラス単位で Moodle を用いて実施し，問題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した（不正対策のため実施時間帯ごとに問題を違えた）．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た．これは学生の負担を減らすことと，テキスト中でも操作説明の部分や高度な部分（興味を持つ学生向け）は試験に含めないためである．

初回の冒頭に行う新入生テストは，学生のこれまでの情報科の学習内容を確認するテストと，それぞれのテスト項目に相当する内容を高校までで学んだか，理解しているかを問うアンケートからなる．その中に含まれる，本稿に関係する「プログラミングを学んだか，理解しているか」の回答集計を**表 2** に示す．初回受講生に実施しているため，回答総数は後で分析する第 4 回と異なっている．プログラミングを理解していると答えた学生が全体の 7%程度，学んだが理解していないと答えた学生が 15%程度いたことが分かる．

### 3.2　科目実施に関する基本方針

本科目では，学生が各自で考えて課題に取り組み，自分なりに設問に取り組んでもらうことを基本方針としている．また，学生のレベルはコンピュータが不得手な学生から得意な学生まで非常に幅広いが，必修科目であり，（既習得単位認定者を除けば）全員が受講し単位を取らなければ卒業できない．

これらのことを考慮し，「課題 1 個以上」という緩い条件を設定し，それぞれの内容が得意でない学生もやさしい課題が選べ，さらに興味を持った学生は自発的に多く取り組めるようにすることで，学生の多様性に対応した．

そして，レポート課題も協力して行うことを勧め，ただしレポート自体は必ず各自で書くことを義務づけた（レポートのコピーにはペナルティがあることを明示）．一方でレポートの内容については「要件を満たしていれば（課題 1 個でも）B（通常点）」とすることで，レベルを問うのではなく自分なりに課題をやることを求めていることを明示した．

一般には，より高度な課題，より多くの問題をこなした学生の方が点数が高くなる方が公平であり普通である，という考え方が多そうである．しかし，もしそのような方針をとったとすれば，コンピュータが得意な学生は喜々として点数を上積みするであろうが，そうではない多くの学生は否定的な感情を持ち，ドロップアウトも増えるものと予想される．ここで重視したのは，得意でない学生がやる気をなくさないことであり，そのため試験範囲を狭く設定したり，やさしい問題だけでもきちんと解いてもらえればよいとしたものである．

一方で，余裕がある学生にはなるべく多く問題を解いて実力をつけてもらいたいことも確かである．そのことを，筆者の 1 人は授業の演習サポートに入って巡回しているときや，ツイッターで授業について情報発信しているときに伝え，「学ぶことは結局自分のためになることである」と説得してきた．

ただし，700 名規模の授業で，筆者らが参加できるのは授業の一部であり，またツイッターも実際には見ている学生はさほど多くないので，そのような伝達手段の効果は限られていた[*3]．結果としては，非常に簡素なレポートからページ数が多く力作なレポートまで多様なレポートが提出されている．

また，コピーによるレポートは筆者らが知る限り発見されていない（課題の内容自体はもともと協力してもよいこととしている）．これは，コピーを行って「C」「D」となるリスクをとるよりも内容が簡素でも自分の言葉で書いて「B」をとっておけばよいと学生に判断してもらえたからであると考えている．そして，課題内容を他人に教えてもらったとしても，それを自分の言葉でレポートにするなら，まったく無知では何も書けないので，一定の学びは起きているものと考えている．

アンケートについてもレポートと一体で提出するようにしているため，すべて記名となる．アンケートの中には（本稿で取り上げている回以外についてであるが）批判的なものも含まれているので，記名であるから取り繕った内容しか書かないということはないと考えている．

以下で示すデータはすべて，上記の状況において得たものである．

現在では試験の結果返却まで含めて本科目は終了しているが，学生の多様性に対応し，得意でない学生でもやる気をなくさないようにという指針は適切であったと考えている．得意な学生が「たくさん課題を行ってもレポート評価が同じなのは納得がいかない」とコメントすることはあったが，得意な学生は試験でその能力を発揮してもらえたと考える．また，課題を 1 個以上任意個提出という方針についても，実際には複数の課題をやって提出する学生が多かった（第 4 回については以下で詳しく報告する）．

### 3.3　アセンブリ言語プログラミング実習

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，筆者らが授業時に観察した範囲では，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そうであった[*4]．

このため，2017 年度からこの内容（第 4 回）について，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．

機械語を 16 進表記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，アセンブリ言語もあわせて設計し，アセンブリ言語ソースを入力欄に打ち込み RUN ボタンを押すとアセンブルされて（エラーがなければ）ただちに実行される，という形にした．

第 4 回の位置づけは表 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．

これは，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．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つねに高水準言語で行うものであり，アセンブリ言語が実用に使われる場面はきわめて少ない．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 の動作を理解するうえでも効果的である．
- 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したりする必要がなく，学習が短時間で完結する．
- 簡単な問題も，コード記述の自由度・多様性が大きい．コード記述に多様性があることを体験し，多くの選択

[*3] 台風による休講の補講について LMS とツイッターで告知しようとしたが補講の参加者が 0 で延期になったということがあった．

[*4] 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．

肢を考えてコードを組み立てる体験をすることは，高水準言語に進んだ場合にも役立つと考える．

上記の考えから，「小さなコンピュータ」は計算にレジスタ Acc のみを用いる設計とした．ただし，データの並びを扱いたいので，アドレス修飾レジスタ Idx を追加している．命令一覧を**表 A·1** に示した[*5,*6]．

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．この方が入出力命令を用いるよりも簡潔であり，また load 命令や store 命令に対する理解が深まる効果もある．

## 4. 学習内容と授業の様子

テキストの本稿で取り上げている範囲を付録に掲載した．この回は本内容の前に 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 性能向上の話題があるが，予習を前提としていることから，担当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．

内容の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うえ，まず前半の実習を十分行い，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・関心がありやさ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 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分からず説明を聞いている状態だった[*7]．

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とおり実行させ，これに基づいて「1-1：5 つの数を合計する」「1-2：1 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組むようになった．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た後（実際には説明を聞かずにテキストを読んで取り組む学生も多数）「1-3：3 数の最大を求める」に取り組む段階では，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．

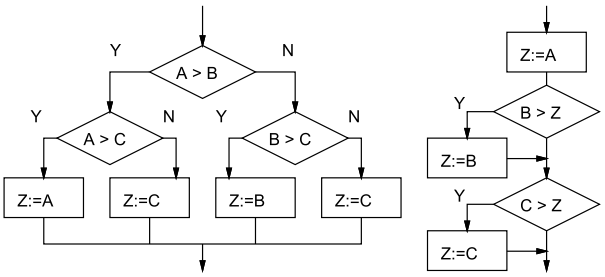


**図 1**「3 数の最大」の 2 つのアルゴリズム
**Fig. 1** Two argorithms for “largest of the three numbers”.

「悩んでいる」内容については，観察しただけでは十分正確には分からないが，「アルゴリズムを考える」部分も「考えたアルゴリズムをアセンブリ言語の上で記述する」部分も両方あり，どちらかといえば後者が多い印象であった．

とくに，**図 1** 左のアルゴリズムを実装しようとした学生は，分岐やラベルが多いため苦労していた．想定解は図 1 右のもので，これは「2 数の最大」の例題の後ろにもう 1 つ判定を追加すればできる．アルゴリズムで迷っている学生にはこちらを示唆するとすんなり理解していた．ただしその場合でも，ラベルまで含めてそのままコピーしてラベルの 2 重定義になるなど，アセンブリ言語の記述に慣れないことでうまくいかない例は多く見られた．

後半のループと Idx については，質問は少なく，一部の学生だけが取り組んでいたが，これも予定どおりであった．

## 5. 評価

### 5.1 レポートによる評価

学生の学習状況を評価するため，提出課題の分析を行った．期限（次回授業前日一杯）までに提出された第 4 回の提出課題 694 件を分析対象とした．1 つ前（第 3 回）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．

すべての課題でアンケートを実施したが，この回においては質問内容は**表 3** のとおりであった．主として 2 番目の設問に対する回答に基づき（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照して），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 4** に示す 6 段階で分類し集計した．これらのうち 0〜2 は（無解答もそう解釈するとして）negative な回答，3〜5 は positive な回答だといえる．

ここで種別 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・楽しいと考えてもらえたことの方が，この先の学習という目的から見れば価値がある

*5 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．
*6 https://joho.g-edu.uec.ac.jp/joho/cl2017/PRAC/sim.html
*7 こく一部，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 3　第 4 回提出課題のアンケート
**Table 3**　Query in the 4th assignment report.

| |
|---|
| 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 4　第 4 回提出課題アンケート Q2 回答に基づく達成度の分類
**Table 4**　Achievement levels based on Q2 in 4th assignment report.

| 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%) |

と考えたためである[*8]．

ただし後述するように，試験による客観評価では種別 3，4，5 の間で結果に統計的に有意な違いは現れていない．この種別の区分はあくまでも，主観評価においてどのような回答があったかを分類する意図である．

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 2** 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（with code と表示），いない（without code と表示）の区別も示した．

レポートの課題文ではコードを記載するように明示してあるものの，学生はプログラミングの課題のようなものに対するレポートを書くことが初めてであるため，コードを記載しなくてよいと勘違いして提出している例が一定数見られた．

コードの記載のないレポートは典型的には，どのような考え方で，どのようなコードを書いた，という説明と，やってみてどうだったかという考察の比較的簡素な文章のみからなる．そのようなレポートが良いレポートであるとはいえないが，同程度の簡素な文章でプログラムが記載されているものや，プログラムは記載されているが文章がほとんどないものなどと比べて大幅に要件を欠いているともいえないので，レポートとして受理し，採点で B または C をつけている．

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 問からなる．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．

[*8] 「できる」「基本的なことはできる」などの記述と「面白かった」などが併記されている場合は 5 に分類した．

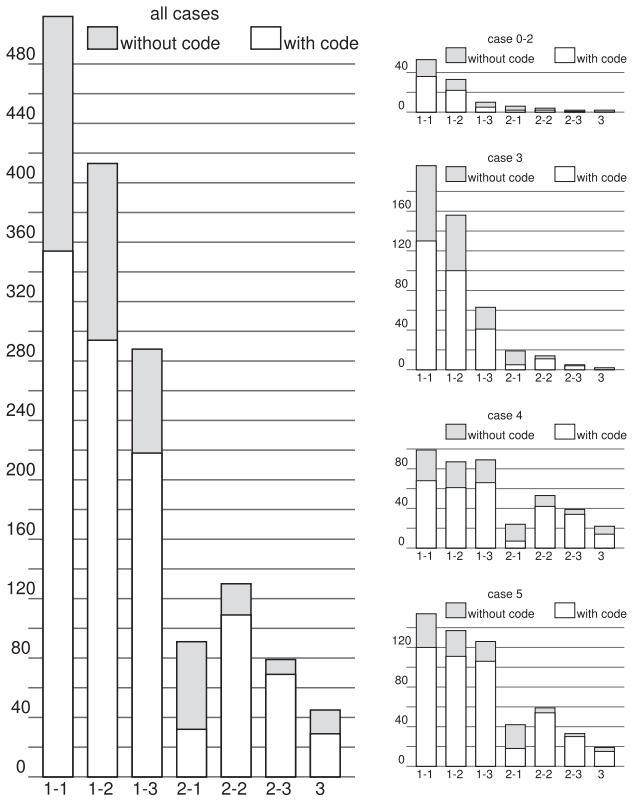


**図 2**　解答された問題（設問 1-x，2-x は付録中に掲載）
**Fig. 2**　Histgram of answered problems (problems 1-x and 2-x are included in the appendix).

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 超）．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 2 割弱の学生が提出している．

さらに，同じ集計を表 4 の分類で 0〜2（negative は件数が少いのでまとめた），3，4，5 それぞれについて分けて行ったものを図 2 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．

さらに，アンケートや課題中のコメントに書かれた主要な記述で類似内容のもの（6 件以上あったもの）を**表 5** に

表 5　件数の多かったコメント
**Table 5**　Representative comments.

| 件数 | コメント（参照記号） |
|---|---|
| 103 | プログラミングは難しいと思った (a) |
| 94 | さらに学んで行きたい/もっとできるようになりたい (b) |
| 58 | コンピュータの原理が分かった (c) |
| 47 | コンピュータ/プログラムの動くようすが面白かった (d) |
| 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 (e) |
| 21 | プログラミングはパズルのようであると感じた (f) |
| 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 (g) |
| 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 (h) |
| 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 (i) |
| 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 (j) |
| 7 | 試行錯誤するのは楽しい/間違い探しは楽しい (k) |
| 7 | 同じ結果でも多様な書き方ができると分かった (l) |
| 6 | 短くしよう/最適化しようと工夫するのが楽しい (m) |
| 6 | 0 と 1 ですべてを表しているのが分かった (n) |

表 6　主要コメントを書いた学生の種別内訳
**Table 6**　Archivement levels of the students corresponding to each of the representative comments.

| コメント \ 種別 | 0 | 1 | 2 | 3 | 4 | 5 | total |
|---|---|---|---|---|---|---|---|
| a. 難しい | 3 | 2 | 17 | 47 | 11 | 23 | 103 |
| b. もっと学びたい | 0 | 0 | 4 | 37 | 10 | 43 | 94 |
| c. 原理が分かった | 0 | 0 | 1 | 23 | 18 | 16 | 58 |
| d. ようすが面白い | 0 | 0 | 3 | 31 | 8 | 5 | 47 |
| e. 各命令は単純 | 0 | 2 | 1 | 12 | 9 | 12 | 36 |
| f. パズルのよう | 0 | 0 | 1 | 7 | 1 | 12 | 21 |
| g. 高水準言語と違う | 0 | 0 | 1 | 4 | 7 | 4 | 16 |
| h. 厳密である | 0 | 0 | 1 | 2 | 2 | 4 | 9 |
| i. 設計が重要 | 0 | 0 | 0 | 2 | 4 | 1 | 7 |
| j. 多数の命令が必要 | 0 | 0 | 0 | 4 | 2 | 1 | 7 |
| k. 試行錯誤は楽しい | 0 | 0 | 0 | 2 | 2 | 3 | 7 |
| l. 多様な書き方 | 0 | 0 | 1 | 2 | 2 | 2 | 7 |
| m. 最適化が楽しい | 0 | 0 | 0 | 0 | 3 | 3 | 6 |
| n. 0/1 で表している | 0 | 0 | 0 | 4 | 1 | 1 | 6 |

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．

各コメントをつけた学生の表 4 の種別内訳を**表 6** に示す．これを見ると，「難しい」は種別が netagive（0〜2）の学生もいるが，positive の学生にも多い．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った．今回の場合も同じことがあてはまると考える．

それ以外はほぼ positive の学生によるものであるが，「もっと学びたい」「動くようすが面白い」「各命令は単純と分かった」の回答は negative の学生にも見られた．これは，negative な意見の学生でも何も学んでいないわけではないことを示しているといえる．

### 5.2　試験による評価

本科目の試験は学期末に，授業に用いたと同じ Moodle システムを利用してコンピュータ上で実施した．2017 年度の試験受験者合数は 730 名であった（追試験は除く）．以下では前節までとの整合性のため，上記の中で第 4 回の課題を提出していた 677 名ぶんを分析対象とする（第 4 回の課題を提出したが期末試験に欠席した学生が 17 名いた）．

試験時間は 80 分であり，設問数は 42 問である．不正防止のため，時間帯の異なるクラスでは問題が異なるようにしているが，各問とも授業時の復習確認問題（全クラス同一）の類題であり，したがって互いにも類題となっている．

問題には次の 2 種類がある．

- 5-2 問題 — 5 つの文章（30〜50 文字程度）が提示され，そのうちから正しいものを 2 つ選ぶ．34 問を出題．
- 短冊問題 — 選択肢の集合が提示され，それらから選択して並べることで回答を構成する，短冊方式による問題．8 問を出題．

短冊方式は，筆者のうち 1 人が情報入試研究会において開発に関与した試験方式であり [5]，もともとはプログラム作成能力を評価することを目的として開発された．回答の自由度が高いため，単純な選択肢式では評価しにくい「問われたものを組み立てる」力が測れることが期待される．

**図 3** に，短冊問題の出題ページを示す．ここで表示されている問題は**表 7** の Q15 の類題に相当する予習確認テストである（解答は中途）．回答者は右の選択肢フィールドから左のフィールドに選択肢をドラッグ&ドロップして解答を組み立てるようになっている（実際の試験では記号列を Moodle の解答欄にコピーして提出）．

実際の Q15，Q16 の出題内容は，次のような形のものとなっている（$m$，$n$，条件などをセットによって違えることで類題を作成している）．

Q15　番地 X に格納されている値の $m$ 倍と番地 Y に格納されている値の $n$ 倍の和（差）を計算し，番地 Z に格納して停止する．

Q16　番地 X に格納されている値の $m$ 倍と番地 Y に格納されている値の $n$ 倍の大小関係（大/小/以上/以下のいずれか）に応じて，1 または 2 を番地 Z に格納して停止する．

本科目では短冊方式をプログラミングに加えてそれ以外

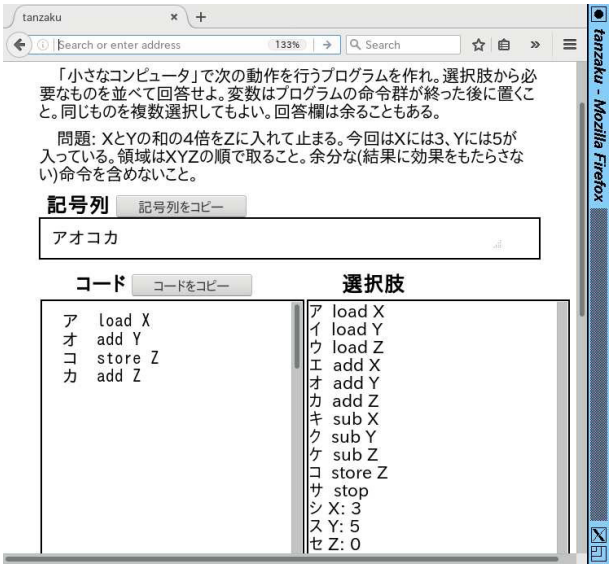


図 3　短冊問題の出題ページ

**Fig. 3**　Image of the split-paper test web page.

表 7　短冊問題のあらまし

**Table 7**　Overview of split-paper test problems in the end-term test.

| 問題 | あらまし | 無解答 |
|---|---|---|
| Q15 | 入力 X と Y の値に応じた数値を計算して Z に格納し停止するアセンブリ言語プログラムを構築する. | 14.9% |
| Q16 | 入力 X と Y に応じて計算した値の大小に応じて Z に指定した値を格納し停止するアセンブリ言語プログラムを構築する. | 33.8% |
| Q25 | `tr`, `sort`, `uniq` などを用いてシェルコマンドのパイプラインを構築する. | 41.1% |
| Q27 | 入力 x と y の値に応じて数値または YES/NO などの文字列を表示するような JavaScript プログラムを構築する. | 29.7% |
| Q36 | 指定した模様や色を持つ 6 ピクセル ×6 ピクセルの PB テキスト M 形式画像ファイルを構築する. | 14.2% |
| Q37 | 指定した見出し, 本文, 図表を含んだ 10 行程度の LaTeX のソースファイルを構築する. | 21.8% |
| Q39 | HTML で指定した内容（リンクや強調を含む）の段落を構築する. | 17.8% |
| Q40 | HTML で指定した形・内容の table 要素を構築する. | 20.7% |

の多様な内容の評価に使うことを意図して，表 7 の 8 問を出題した（問題番号は 5-2 問題と通算した出題番号）．試験時間に比して設問数が多いため，配点が同じで（告知済み）解答時間を要する短冊問題の無解答率（解答欄に何も記入していない比率）は 5-2 問題の無解答率（1 問だけ 3.4%，残りはすべて 2.2%以下）より大きかった．短冊問題の中でアセンブリ言語プログラミング問題（Q15，Q16）の無解答率が他の設問と比べてとくに大きい/小さいことはないといえる．

前述のように，短冊問題の配点が 5-2 問題と同じであること，短冊問題の方が時間がかかることは学生には周知されており，多くの学生は「時間があったら」短冊問題に取り組むという方針をとったものと想像される．以下であげる解答データはこの前提に基づき検討する．

表 8　期末試験アセンブリ言語問題の集計（上段：件数，下段：百分率）

**Table 8**　Results of the assembly language problems in the end-term test (upper: frequency, lower: percentage).

| 問題 \ 種別 | 0-2 | 3 | 4 | 5 | 全体 |
|---|---|---|---|---|---|
| Q15：正解 | 18 | 109 | 69 | 112 | 308 |
| | 28.12 | 43.43 | 47.26 | 51.85 | 45.49 |
| Q15：部分点 | 10 | 47 | 31 | 39 | 127 |
| | 15.62 | 18.73 | 21.23 | 18.06 | 18.76 |
| Q15：不正解 | 20 | 55 | 27 | 39 | 141 |
| | 31.25 | 21.91 | 18.49 | 18.06 | 20.83 |
| Q15：無解答 | 16 | 40 | 19 | 26 | 101 |
| | 25.00 | 15.94 | 13.01 | 12.04 | 14.92 |
| Q16：正解 | 4 | 23 | 18 | 16 | 61 |
| | 6.25 | 9.16 | 12.33 | 7.41 | 9.01 |
| Q16：部分点 | 3 | 35 | 23 | 34 | 95 |
| | 4.69 | 13.94 | 15.75 | 15.74 | 14.03 |
| Q16：不正解 | 21 | 105 | 66 | 106 | 298 |
| | 32.81 | 41.83 | 45.21 | 49.07 | 44.02 |
| Q16：無解答 | 36 | 88 | 39 | 60 | 223 |
| | 56.25 | 35.06 | 26.71 | 27.78 | 32.94 |

**表 8** に，期末試験アセンブリ言語短冊問題の 2 問の結果を，表 4 の種別ごとに示す（negative の 3 つはまとめている）．短冊問題の採点は，複数ありうる正解のいずれかに一致した場合を「正解」，解答列の 1 カ所の修正（挿入，削除，隣接項目の交換）によりいずれかの正解と一致する場合を「部分点」としている．

これによると，Q15（枝分かれのないアセンブリ言語プログラム）では全体で 6 割以上が正解ないし部分点となっている．時間不足で無解答となった学生（positive の分類 4 や 5 でも 1 割以上）を除外すれば比率はもう少し高くなる．やや難しい，Q16（条件分岐のあるアセンブリ言語プログラム）でも，2 割（無解答を除けば 3 割）が正解ないし部分点となっていた．

**表 9** に，正解を 2 点，部分点を 1 点としてアセンブリ言語問題 2 問の得点を計算した集計を分類ごとに示す（negative の 3 分類はまとめている）．0〜2（negative）でも 45%は 1 つ以上で部分点をとっており，全体では 67.5%がアセンブリ言語問題で得点していることが分かる．時間不足による無解答を考慮するなら，時間があればこれらの比率はさらに高くなることが期待される．

種別間で平均の差に意味があるかどうかを，Welch の t 検定によりチェックした結果を**表 10** に示す．結果として，negative は残りの 3 種別（いずれも positive）と有意な差があったが，positive 相互間では差は認められない．すなわち，少なくとも今回の試験結果の分析で見た範囲では，positive の 3 種別には優劣の差はなかった．これらの種別

表 9　アセンブリ言語問題合計点（上段：件数，下段：百分率）
**Table 9**　Scores of the assembly language problems in the end-term test (upper: frequency, lower: percentage).

| 得点 \ 種別 | 0-2 | 3 | 4 | 5 | 全体 |
|---|---|---|---|---|---|
| 4 | 1 | 12 | 12 | 9 | 34 |
| | 1.56 | 4.78 | 8.22 | 4.17 | 5.02 |
| 3 | 5 | 29 | 17 | 31 | 82 |
| | 7.81 | 11.55 | 11.64 | 14.35 | 12.11 |
| 2 | 15 | 85 | 51 | 86 | 237 |
| | 23.44 | 33.86 | 34.93 | 39.81 | 35.01 |
| 1 | 8 | 41 | 27 | 28 | 104 |
| | 12.50 | 16.33 | 18.49 | 12.96 | 15.36 |
| 0 | 35 | 84 | 39 | 62 | 220 |
| | 54.69 | 33.47 | 26.71 | 28.70 | 32.50 |
| 平均点 | 0.89 | 1.38 | 1.56 | 1.52 | 1.42 |
| 件数 | 64 | 251 | 146 | 216 | 677 |

表 10　種別間の平均の差の検定結果
**Table 10**　Results of the statisitcal significance tests for difference of the avarage score among category levels.

| 種別 \ 種別 | 0-2 | 3 | 4 |
|---|---|---|---|
| 3 | 0.49<br>0.17～0.80<br>p = 0.00266 | 上段：平均の差<br>中断：差の 95%信頼区間<br>下段：p 値 | |
| 4 | 0.67<br>0.33～1.01<br>p = 0.00016 | 0.18<br>−0.07～1.38<br>p = 0.150 | |
| 5 | 0.63<br>0.31～0.95<br>p = 0.00014 | 0.14<br>−0.07～0.36<br>p = 0.188 | −0.04<br>−0.29～0.22<br>p = 0.766 |

の違いとパフォーマンスの関係については，今後の検討の課題としたい．

### 5.3　部分点解答の分析

本科目ではアセンブリ言語の試験は短冊問題を使用している．短冊問題では組合せによる選択肢がきわめて多くあることから，通常の穴埋め形式や単純な選択肢式形式の試験と比較して，「偶然に，運良く」得点する可能性はきわめて低いと考えているが，このことを確認するため，実際に部分点となった解答を分析した．

分析は，Q15 の部分点となった解答 127 件，Q16 の部分点となった解答 95 件について検討し，どのような間違いであるかを分類する形で行った．その集計をそれぞれ**表 11**，**表 12** に示す．

誤りの内容分類は，解答から原因が予想できる場合はその原因を記載し，解答だけではなぜその誤りとなったか分からない場合は「1 命令間違い」「2 命令の入れ替わり」などと記載した（※印）．そのようなものの多くは，見落としやドラッグ&ドロップのインタフェースの操作ミスによるものではないかと想像される[*9]．

また，Q16 で「2 命令の入れ替わり」が多くなっているのは，「条件に応じて 1 または 2 を Z に格納する」設問であることに関係していると思われる．具体的には，**図 4** のように片側の条件についての load を行った後，他方のロードをまたぎ越す jump 命令が欠落していて，なおかつ他方のロードの直後から最後の store 命令への余分な jump が

表 11　Q15 の部分正解（127 件）の内訳
**Table 11**　Causes for partial correct answers in Q15 (total 127).

| 件数 | 内容 |
|---|---|
| 27 | 結果を格納する store 命令が不足している |
| 25 | 冗長な load 命令がある（結果は正しい★） |
| 18 | 冗長な store 命令がある（結果は正しい★） |
| 16 | 領域確保命令が不足している☆ |
| 12 | stop 命令が不足している☆ |
| 8 | 間違った load 命令があり，結果が正しくならない |
| 5 | 加算命令が不足していて，結果が正しくならない |
| 5 | 正解から 1 命令間違い※ |
| 3 | 余分な加算命があり，結果が正しくならない |
| 3 | 正解から隣接 2 命令の入れ替わり※ |
| 2 | 領域確保に間違いがある |
| 2 | 領域確保順に間違いがある（結果は正しい★） |
| 1 | 加算命令が間違っていて，結果が正しくならない |

表 12　Q16 の部分正解（95 件）の内訳
**Table 12**　Causes for partial correct answers in Q16 (total 95).

| 件数 | 内容 |
|---|---|
| 26 | 条件分岐命令の条件が違っている |
| 24 | 境界条件（等しい場合）の扱いが違っている |
| 13 | 冗長な jump 命令がある（結果は正しい★） |
| 11 | 正解から隣接 2 命令の入れ替わり※ |
| 5 | load 命令のロードする値が違っている |
| 4 | 正解から 1 命令欠落※ |
| 4 | 条件のための計算が間違っている |
| 3 | 飛び越す jump 命令の欠落 |
| 2 | 正解から 1 命令間違い※ |
| 1 | 正解から 1 命令余分※ |
| 1 | stop 命令が不足している☆ |
| 1 | 領域確保命令が不足している☆ |

```
      (正)              (誤)
      load 1            load 1
      jump L2     L1: load 2
L1: load 2            jump L2
L2: store Z       L2: store Z
```

図 4　Q16 に複数見られた「2 命令交換誤り」の形
**Fig. 4**　Form of “swapped instruction error” found muitple times in Q16.

[*9] 操作ミスした場合でも，その結果を反映したコードが画面上に現れるため，それを見れば間違いに気付くはずであるが，アセンブリ言語プログラミングに習熟していないと間違いに気付けずそのままになる可能性がある．また，制限時間の終わりに近い場合は気付いてもそのまま出す可能性がある．

あるという誤りの複合の形のものがこれに該当している（2つの誤りの複合なのか単純な操作ミスかは解答からは分からない）．

Q15 の 127 件中 45 件，Q16 の 95 件中 13 件はプログラムとしては正しく動作するものであり，問題設問中の制約「冗長な命令を入れないこと」「領域確保命令は XYZ の順とすること」に合致しないため減点となったものであった（★印）[*10]．

このほか，stop 命令が不足しているもの，結果を格納する番地の領域確保命令が不足しているもの（単純にこれらを補えば正しく動作する）が Q15 では 28 件，Q16 では 2 件あった（☆印）．両問の差は，Q16 の方が複雑なコードを求める問題であるため，このような単純な欠落をおかす学生はもっと別の誤りもおかしていることによると考えられる．

このほか，Q15 では結果を格納する store 命令が不足しているという間違いが 27 件と最も多く，そのほかの主な間違いは求める計算結果を得るための命令列の間違いであった．また Q16 については，条件分岐命令の条件間違いと「以上」「より小さい」などの判定において境界条件（等しい場合）が間違っているものが上位 2 件であり，合わせて 30 件であった．

これらを総合して，今回の試験において部分点をとった学生はおおむね，それぞれの設問のレベルのプログラムが書けるといって差し支えないものと，著者らは考えている．

## 6. 議論とまとめ

本稿では，電気通信大学 1 年前期情報基礎科目「コンピュータリテラシ（必修）」において，「コンピュータの構造と動作原理」の内容を学ぶ手段として，簡潔な命令セットを持つ「小さなコンピュータ」のシミュレータを用いたアセンブリ言語プログラミングを導入した結果について報告した．

その目的は，(1) CPU の動作を理解してもらうことと，(2) プログラミングを体験しそれに興味を持ってもらうことである．ただし，体験といってもコードをテキストから写してそのまま動かすのにとどまらず，それをもとにして「提示された課題のプログラムを構成する」ことを重視した．

アセンブリ言語プログラミングを採用した理由としては，CPU の動作と直結しておりコンピュータの原理を理解するのに適していること，言語が単純で学習に要する時間が短いこと，コード記述の多様性が高いという点でプログラミング体験に適していることがあった．

授業は全 15 回のうちの 1 回（90 分）のみで実施し，例題を説明したうえで，5 つの数を合計する課題，1 つの数値を 5 倍する課題，3 つの数値の最大を求める課題などの演習を中心に行った．授業時には，多くの学生が自分で考えてプログラムを作る様子が見られた．

694 件（計 13 クラス）の課題レポートに書かれたアンケートに基づき，達成度に対する主観的評価を集計した結果，negative な解答（未解答，できない，難しいなど）は 10%弱にとどまり，positive な解答（基本は分かった，できるようになった，面白い/楽しい）が 9 割超であった．このような比率となったのは，「自力でプログラムを作って動かす」ことを重視した結果であると考えている．

プログラミングの課題についても，簡単な課題にとどまらず，3 数の最大やループを含む課題を提出した学生が一定数いた（それぞれ全体の 1/3 超，2 割弱）．課題レポート中の自由記述でも，コンピュータの原理が分かったなどの解答が多く見られた．

期末試験において，選択肢を並べ替えてプログラムを構成する短冊型の問題形式で，アセンブリ言語プログラミングの問題を 2 題（分岐のないプログラムと分岐のあるプログラムの作成）出題した．その結果，分岐のないプログラムの正解・部分正解が 6 割超，いずれかの問題 1 つ以上で正解・部分正解となった学生が 7 割弱との結果を得た．

部分正解について，その誤り内容を検討した結果，部分正解できていればその課題プログラムを書く力はおおむねあるものと考える．

これらの結果を総合して，アセンブリ言語プログラミングを導入した目的（CPU 動作の理解，プログラミング体験）はいずれも達成されたものと判断している．

現在では実用のためのプログラミングがほぼ 100%高水準言語で行われることから，大学の初年次情報教育の中でプログラミングを扱う場合も，アセンブリ言語を採用することはほとんどない．しかし，CPU の動作原理を学ぶという目的と組み合わせて扱う場合は，アセンブリ言語も有力な選択肢になりうるものと考える．

ただし，アセンブリ言語によるプログラミングが繁雑で難しいということも一般に認識されていることである．今回これらの問題を回避できた理由としては，(a) 最小限の単純な命令セットを持つ仮想的な CPU を導入したこと，(b) Web ブラウザ上で動作する操作の簡単なシミュレータを実習に使用したこと，(c) 課題を精選したうえで複数提供し，学生が自分のレベルに合った問題を選んで取り組めるようにするとともに，試行錯誤しながら（自分で考えて）プログラムを組むよう促したことが重要だったと考えている．

また，情報専門学科では専門教育の中でアセンブリ言語によるプログラミングを取り上げることが一般的である．本稿で提示した内容は情報系の学生に限定されない一般情報教育として実施しているものであり，より広い範囲の学生にアセンブリ言語プログラミングを体験してもらうことに特徴がある．

[*10] これらの制約は，このような制約がないとすべての正解を列挙するのが困難になるために設けたものである．

そして，電気通信大学の状況について担当教員に尋ねた範囲では，情報専門学科に進学する学生にとっても，専門教育の中でアセンブリ言語プログラミングはハードルの高い内容であり，学習に困難を覚える者も多いとのことであった．本稿の試みはそのような問題を緩和する上でも有用である可能性がある（今年度から開始した内容であるため，実際にどうであるか判明するのはまだ先になる）．

**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いただいた．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いただき，実際の科目運営を担っていただいた．ここに感謝します．

### 参考文献

[1] Branovic, I., Giorigi, R. and Martineli, E.: Web-MIPS: A New Web-Based MIPS Simulation Environment for Computer Architecture Education, *WCAE'04 Proc. 2004 Workshop on Computer Architecture Educaiton*, DOI: 10.1145/1275571.1275596 (2004).
[2] 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 (2001).
[3] 小谷善行，阿刀田央一，中森眞理雄，高橋延匡：情報工学科における実験・演習の一設計例，情報処理学会論文誌，Vol.22, No.5, pp.402–410 (1981).
[4] 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)，入手先 ⟨http://joho.g-edu.uec.ac.jp/joho/cl2017/⟩.
[5] 久野　靖：情報入試研究会試作問題#001 問題解説，情報入試フォーラム 2013 資料集，pp.4–10, 情報入試研究会 (2013).
[6] 久野　靖：Ruby による情報科学入門，近代科学社 (2008).
[7] 久野　靖：改訂 2 版 UNIX による計算機科学入門，丸善 (2004).
[8] 久野　靖：UNIX の基礎概念，アスキー (1995).
[9] 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 (2008).
[10] 清水敬子，阿刀田央一，高橋延匡：情報工学系学科の計算機初期教育における EDSAC の活用の試みと効果，情報処理学会論文誌，Vol.24, No.3, pp.281–292 (1983).
[11] Vollmar, K. and Sanderson, P.: MARS: An education-oriented MIPS assembly language simulator, *ACM SIGCSE'06*, pp.239–243 (Mar. 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）．

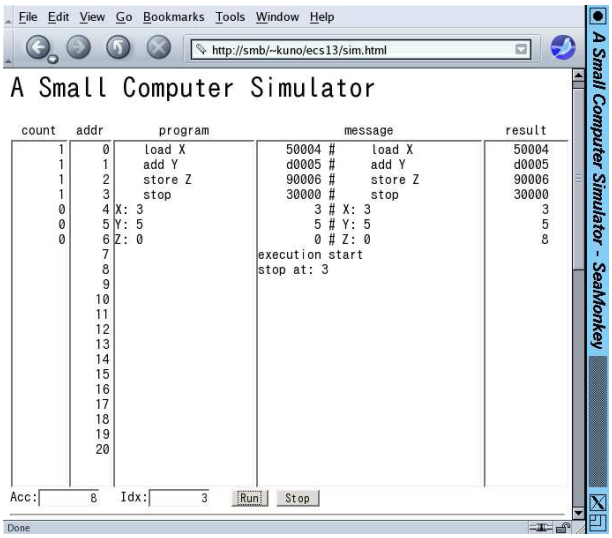


図 A·1　「小さなコンピュータ」の画面
**Fig. A·1**　Screenshot of "small computer" web page.

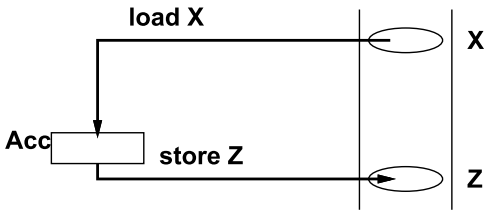


図 A·2　ロード命令とストア命令
**Fig. A·2**　Load and store instructions.

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くる命令（**図 A·2** 上），「add」は，メモリを指定場所（この場合は 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，その名前どおりプログラムの実行を停止する命令です．この命令がないと，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」などが入っていて，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．

私たちがふだん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 だけからな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ときだけ分岐します（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 に持ってきて，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」でここまでに出てき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．

1-1.　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）．

1-2.　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）．

1-3.　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，もっとたくさんの数を合計したければどうしましょうか?　A，B，C··· のようにたくさん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．

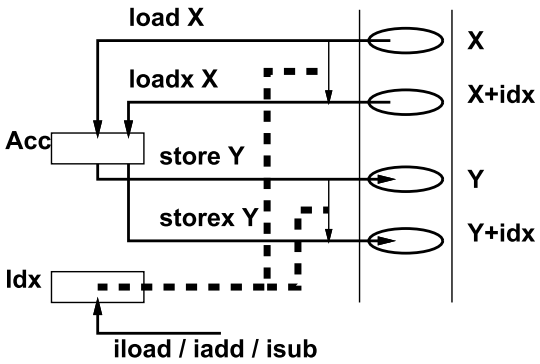


図 A·3　Idx を使うロード命令とストア命令

**Fig. A·3**　Load and store instructions which incorporate Idx register.

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 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き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 に値を持ってきますが，最初は Idx は 0 なので，ちょうど Data の場所の値が持ってこられます．もしその値が 0 なら，これは終わりの印なので（合計をとりたいのに 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いき，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たくさん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いえます．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）[*11,*12]．

表 A·1　「小さなコンピュータ」命令一覧

**Table A·1**　List of instructions for “small computer”.

| 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 |

**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．

2-1.　「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．

2-2.　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」

2-3.　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．

*11 条件分岐命令がたくさんありますが，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」

*12 「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．

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．

（注記）演習 2 設問 2-3 で「Idx レジスタを使う必要がない」と記しているのは，Idx レジスタから値を取り出す（メモリに格納したり Acc に転送したりする）機能はないように設計しているため，Idx を使おうとしても行き詰まるのでこのようなアドバイスを記しているものである．


**久野 靖**　（正会員）

1984 年東京工業大学大学院理工学研究科情報科学専攻博士後期課程単位取得退学．同年同大学理学部情報科学科助手．筑波大学講師，助教授，教授を経て，現在，電気通信大学大学院情報理工学研究科教授．筑波大学名誉教授．理学博士．プログラミング言語，プログラミング教育，情報教育に関心を持つ．本会情報処理教育委員会委員．ACM，IEEE-CS，日本ソフトウェア科学会，日本情報科教育学会各会員．本会シニア会員．

**江木 啓訓**　（正会員）

電気通信大学大学院情報理工学研究科准教授．博士（工学）．2000 年慶應義塾大学環境情報学部卒業．2005 年同大学大学院理工学研究科後期博士課程単位取得退学．同年東京農工大学総合情報メディアセンター助手，2007 年同助教．2013 年神戸大学情報基盤センター准教授．2015 年より現職．教育学習支援システム，教育用情報システム，ヒューマン・コンピュータ・インタラクションの研究に従事．IEEE，日本教育工学会各会員．本会シニア会員．

**赤澤 紀子**　（正会員）

1997 年電気通信大学大学院電気通信学研究科博士前期課程修了．同年日本電気株式会社入社，ソフトウェア開発等に従事し，退社．2015 年電気通信大学情報理工学研究科博士後期課程修了．現在，電気通信大学特任助教．博士（工学）．情報教育，プログラミング教育，教育の情報化に関心を持つ．日本教育工学会，日本教育学会各会員．

**竹内 純人**

電気通信大学教育研究技師部主任学術技師．学内の情報基礎教育および支援活動に従事．

**笹倉 理子**　（正会員）

電気通信大学教育研究技師部学術技師．

**木本 真紀子**

電気通信大学大学院情報理工学研究科教務補佐員．