〔論　文〕

光転送と仮想マシンを用いた教育用組込みシステムの開発と実践

千田 陽介[*1]

Educational Embedded System Using Optical Transmission of Virtual Machine Code

Yosuke SENTA[*1]

Abstract

For beginners, it is more exciting to use actual moving robots or colorful led substrates in programming practice. If we allow students to take the equipment home after school and play with it, we may be able to teach the students the fun of computer programming. To realize this concept, the equipment must be made cheaper and easy to set up for the programming environment. This study develops an embedded educational system using a PIC microcomputer, a virtual machine running in a PIC chip, and optical transmission of the virtual machine code using blink of the PC display. The details of the educational system are presented herein along with the results of classes using the system.

Key Words : Programming education, Virtual machine, Optical Transmission

1　はじめに

「コンピュータは与えられた手順を愚直に遂行する機械である」という概念を子供たちに伝えようとする様々な取り組みがなされて久しい．与えられた手順とは言うまでもなくプログラムのことであり，IT 社会を支える様々な機器はこのようなしくみの延長線上に成り立っていると肌で感じて欲しいことや，プログラミングという行為自体がパズルを解くようで楽しいと発見して欲しいなど，その思惑は様々である．その理念は社会的ニーズと重なり，今年度 (2020 年度) より小学校プログラミング教育が必修化された．残念ながら新型コロナウイルス (COVID-19) による社会的災禍により幸先の良いスタートとなっておらず，取り組みや効果について十分な情報は入っていない．なお文科省が考えている小学校プログラミング教育カリキュラムでは，プログラミング的思考やコンピュータの活用法について重点を置いており，プログラミングという行為自体には重要視されてない [(1)]．文系理系万遍なく教育するカリキュラムとしてはあるべき姿かもしれない．しかし工学部もしくは工業大学が希望者向けにコンピュータやプログラムの教室を開く場合はやはり実際にプログラミングを体験してもらいたい．

初学者にコンピュータプログラムを教えるには大きく二つの障害がある．一つはプログラムは本来テキストベースの作業だということである．テキストは抽象化した作業を表現するには自由度が高い代わりにスペルミスや文法エラーを多発してしまい，本筋であるプログラムの概念やプログラミングの楽しさを学ぶ前に挫折してしまう可能性が高い．そのような背景から Scratch 言語 [(2)] をはじめとするグラフィカルな教育用言語が考えられている．もう一つの障害は実行結果の出力先に目新しさが必要なことである．単純にディスプレイでキャラクタを動かしたりスピーカから音を出したりすることは，ゲーム等の通常アプリと変わらずデジタルネイティブ世代は強い関心を起こさないように見える．実際の物体に作用することが望ましい．そのためロボットや LED 基板といった実際の物体を用いたプログラミング教育が数多く試みられてきた．制御対象は市販の玩具ロボットをそのままもしくは教育向けに改造したものや [(3–5)]，自作のロボットや LED 基板を用いたもの [(6,7)] など様々な形態がある．

筆者も一日大学生や市民講座等で実際のロボットや LED 基板を用いた初学者対象のプログラム教室を開くことがある．教室で使用した教材は終了後，記念に持ち帰ってもらうことが多い．自宅でも引き続きプログラミングを楽しんで欲しいと考えるからである．ここで問題となるのがプログラム開発環境とプログラム転送手段である．このうち開発環境とは組込み教材上のマイコンプログラムを開発するもので通常は PC を使う．近年は一般家庭に PC が普及したため PC の有無についての心配は無くなったが，親や兄姉などの PC を借りる場合，大がかりなソフトウェアやドライバのインストールは嫌がられる傾向にある．バイナリファイル一つコピーするだけで動くことが望ましい．一方プログラム転送手段とは組込み教材上のマイ

[*1] 情報ネットワーク工学科
令和２年11月12日受理

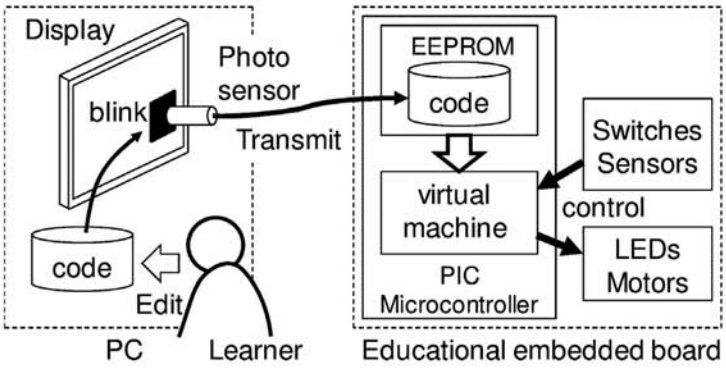


Fig. 1: Concept of this study[12]

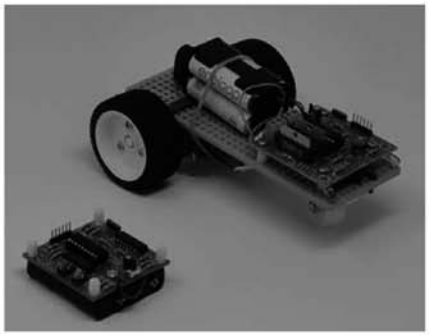


Fig. 2: Prototype of Educational Emvedded System[13]

コンに制御コードを転送するもので, 書き込み機 (ライタ) を使うことが多い. しかしライタ本体は比較的高価で持ち帰る教材に含めることは難しい. 別途購入にするとほとんどの者は自宅学習をしないであろう.

そこで筆者は**図 1** のように PC ディスプレイの明滅によってコードを転送するライタ不要のシステムを構築した. 転送されたコードはマイコンの EEPROM 領域に格納され, マイコン内の仮想マシンによって解釈実行される. 本稿はこのシステムについて, 過去公表した内容 [12,13] も交え改めて整理したものである. そのため掲載した一部の図表は既存のものを流用している.

2　組込み教材

2.1　ハードウェア構成

図 2 に図 1 の構成を実現した教材を示す. 図中左が三色 LED の光り方を制御するボード, 同右がセンサを用いて障害物を回避する移動ロボットである. これらのコンセプトは仙台市を中心に活躍している市民グループ: メカトロで遊ぶ会の教材, いろは姫や梵天丸 [8] から強くインスパイアを受けており, ハードウェア的機能はほとんど同じである.

図 2 に示した教材の回路構成を**図 3** に示す. どちらの回路も Microchip 社の PIC16F1827 マイコンを核として構成されている. PIC マイコンは入手性がよく比較的安価なため, 当学では実験装置やハードウェア演習の教材として活用されており学生にとって非常に身近なマイコンである. 学生が研究活動等で PIC マイコンを活用するにあたり, 割り込みなどハード寄りの組込みプログラムの知識や技術を習得しなければならない. そこで当初は, そのような用途にも適用可能なよう教科書的な回路構成としてみた. 例えば三色 LED 基板 (a) において PIC マイコンの RB6 ポートを High にすれば三色 LED-A の赤成分が光る. また移動ロボット (b) において RA6 ポートを High, RA7 ポートを Low にすれば片側のモータ (タイヤ) が正転する. このように各ポートの High/Low を適切に出力するだけで LED の色やロボットの動きを制御することができ, 初学者でも容易にプログラムすることができた.

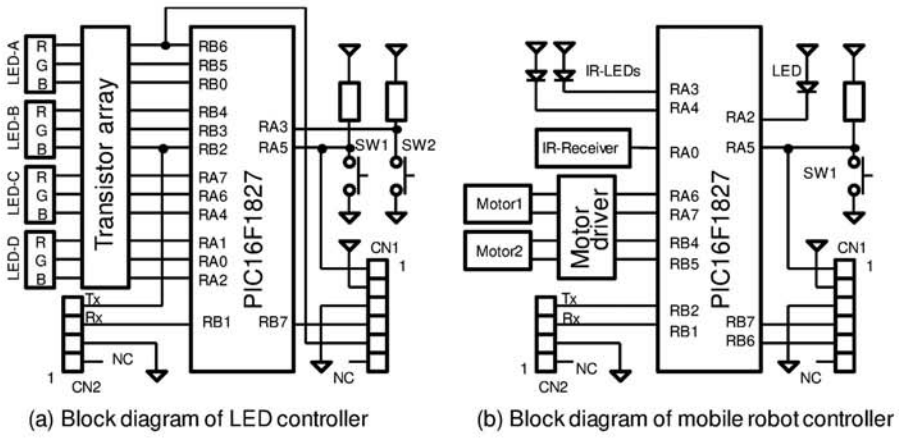


Fig. 3: Architecture of prototype of educational emvedded system[12]

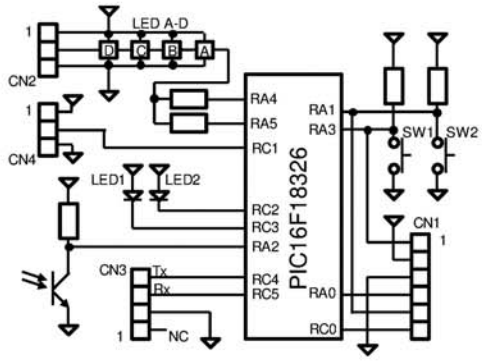


Fig. 4: Block diagram of new LED Controller

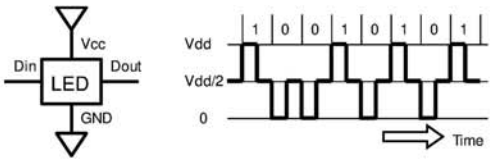


Fig. 5: Architecture of Intelligent control RGB LED

しかし基板が作られてから三年間程, 組込みプログラムの教材として使われることはほとんどなかった. 一方で, 特に三色 LED 教材は親子科学教室や高大連携等で活躍し 50 個以上製作配布したが, 部品点数と半田箇所の多さから準備が大変であった. そこで組込みプログラムの教材には使われないと割り切り, 作りやすさと部品の安さに重点を置いた基板を作成した. 回路構成を**図 4** に示す. 図から分かるようにこの回路は通常の LED でなくマイコン内蔵の LED (OptoSupply: OST4ML5B32A) を用いている. この LED は**図 5** 左のように電源の他に入力 Din と出力 Dout 二つの端子を持っている. この Din には同図右のように Vdd, Vdd/2, 0V の組み合わせで 0 と 1 を表現したビット列を入力する. 入力されたビット列は赤, 緑, 青の輝度として各 8bit, 合計 24bit で解析される. 24bit 以上の入力を与えた場合, 解析に用いられなかった残りのビットは Dout から出力されるので図 4 左上のようにカスケードに繋げることで多数の LED を一本の信号で制御することが可能となる. 2 つのポート (図 4 では RA4 と RA5) を操作すればこのような信号を生成することはできるが, タイミングがシビアなため組込みプログラム初学者が C 言語で実装するには難しいであろう. 本基板は 4 章で述べる仮想マシンを用いることを前提に設計されており, その場合 LED 制御部分は隠蔽されるので問題とならない.

図 4 において CN1~ 4 を設けているが, これは将来当基板を何かの実験に使うかもしれないと設けたもので通常使用しない. またマイコン内蔵三色 LED 以外に普通の LED (LED1, LED2) も存在するが, これは次章で述べる光転送時のステータスを提示するためのものである. **表 1** に新旧の LED ボードの比較図を示す. 表から分かるようにハンダ点数, 部品数, 原価共にかなり改善され作りやすくなっていることが分かる.

2.2 制御コード格納

組込み教材では PC で開発したプログラムを教材内のマイコンに書き込む手段が課題である. プログラムは電池交換等, 電源が遮断されても保持されるよう不揮発性メモリに格納する. 不揮発性メモリにはネイティブコード (ファームウェア: F/W) を格納するプログラムメモリとデータを格納する EEPROM の二種類がある. 通常これらへのデータ格納はライタを用いる. PIC マイコンの場合, Microchip 純正のライタ (PICKit3/PICKit4) は 5,000 円以上する. PIC ライタは自作例も多く教材基板内に機能を入れ込むことはできる. しかし近年の PC は USB 以外の汎用ポートを持つものがほとんどなく, ライタを作るにあたってどうしても数百円はする USB インタフェイス IC を必要とする. 材料費を下げるにも限界がある. 先述したいろは姫や梵天丸の場合も別売で専用の書き込み機を擁していたが安価ではなく, よほど興味を持った者以外は教室終了後持ち帰っても活用されない問題を抱えていた.

このような問題を解決するため玩具ロボット KIROBO[11] では, PC に標準的に搭載されているイヤホンジャックからの音声信号でプログラム転送を行っていた. 転送機能周辺の回路は数個のトランジスタ, 抵抗, コンデンサで構成され安価な作りとなっている. 受信データは通常のライタのように直接不揮発性メモリに書き込むことはできないが, 事前に音声信号をデータ化する F/W を書き込み実行することで間接的に書き込むことはできる. これは非常によく考えられたしくみではあ

Tbl. 1: Comparsion of new and old led board

	First edition	Current edition
PIC Controller	16F1827	16F18325
Constitution	two circuit boards and cable	one board
Number of electronic components	37	26
Number of soldering points	134	76
Approximate material cost (Japanese yen)	780	650

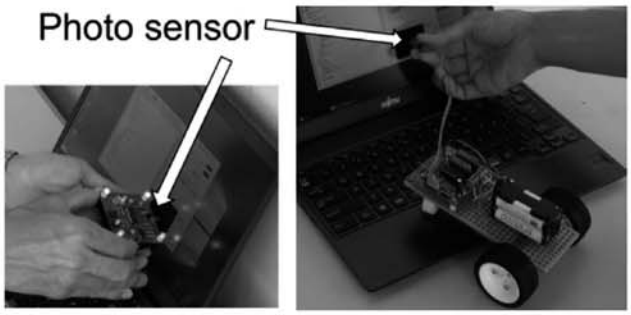


Fig. 6: Code transfer using blinking display[13]

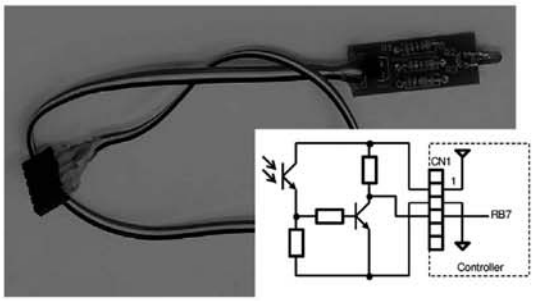


Fig. 7: Optical sensor Module[12]

るもののイヤホンジャックは本来の用途であるヘッドフォンと兼用となるため毎回音声レベルを調整しなければならない短所がある.

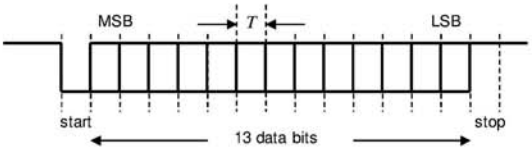
そこで筆者はディスプレイの明滅を用いてデータ転送する方法を考えた. ディスプレイは PC に必須な装置のためその有無を悩む必要はない. また転送プログラム稼働時だけディスプレイに表示すればよいので, 他のアプリケーションや周辺機器とのバッティングや調整作業の必要もない. ディスプレイの明滅を取得するセンサモジュールの部品代は 100 円に満たないため教材に含めることもできる. KIROBO と同じく直接データを書き込むことはできないが, 事前に適切な F/W を入れておくことで間接的にプログラムメモリや EEPROM に保存することはできる.

3　光転送

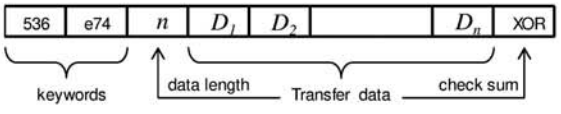
3.1　通信プロトコル

ディスプレイの明滅を用いて光転送を行っている様子を**図 6** に示す. 図左は図 4 で示した新しい基板であり, 基板上に光センサ (NJL7502L) が搭載されている. 一方図右は図 3 で示した旧世代基板のやり方であり, 基板上の CN1 に**図 7** のような光センサモジュールを接続する. データの転送は**図 8** (a) のような調歩同期方式 (start:1, stop:1, data:13, non-parity) を用いた. 調歩同期方式は RS232C 等で使われているシリアル通信の一種で, パリティが無い場合まず信号線が High から Low になることでデータの送信開始を示し (start bit), 次にあらかじめ定めたビット数分 High/Low を変化させてデータを表現し (data bits), 最後に High に戻してデータ終了を示す (stop bit). ここで 1bit 分にかける時間 T (この逆数をボーレートと呼ぶ) はあらかじめ送受信側で取り決めておかなければならない. **図 9** に調歩同期方式のフローチャートを示す. このうち (a) は PC で動作する送信側のフローチャートで, 時間 T 毎に start/data/stop bit を送付するものである. 一方 (b) は PIC マイコン上で動作する受信側のフローチャートで, まず光センサからの入力 (明滅状態) が High から Low (当図では 1 から 0) へ変化することを監視し, そこから $T/2+TN$ 毎にポートの状態を取得し ($N=0\sim14$) データを復元している. ここで $T/2$ を用いてビットが中間でポート状態を取得しているのは, 送受信別のハードウェアで動いており T には必ず誤差があるためである. 送受信それぞれの T を T_s, T_r とすると stop bit まで正しく転送されるためには, $14T_s < 29T_r/2 < 15T_s$ という関係が成り立っていなければならない. これは 3.4% までの誤差なら許容できることを意味し, 安価な PIC マイコンでも十分実現可能である.

光センサは照明の変化等様々なノイズにより予期せぬタイミングで High/Low が切り替わり, データ送付開始 (start bit) と判断する. 図 9 (b) から見て取れるように start 及び stop bit の判定である程度棄却できるが, それでも誤ったデータが受信される可能性はある. そこで図 8 (b) のようにキーワード, データ長, データ本体, チェックサムで構成された通信プロトコルを用いて堅牢性を高めた. 受信プログラムは 2word のキーワードがマッチしないとデータ受信を開始せず, 全データの排他的論理和によるチェックサムが合わなければ受信失敗と判断する. ここでキーワード `0x536, 0xe74` はランレングスが適度に分散している (1bit:8, 2bit:6, 3bit:2), 13bit 目が 0 である (stop bit と逆) という条件の中から適当な数として, 筆者



(a) Datalink layer



(b) Application layer

Fig. 8: Data transfer protocol[13]

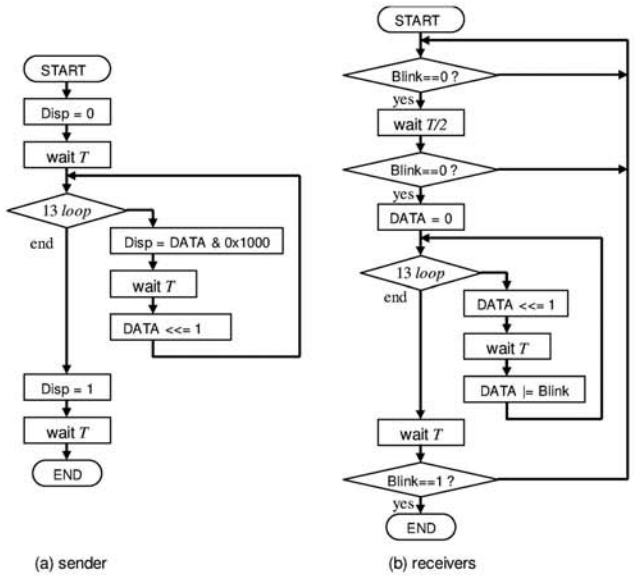


Fig. 9: Flowchart of synchronous communication

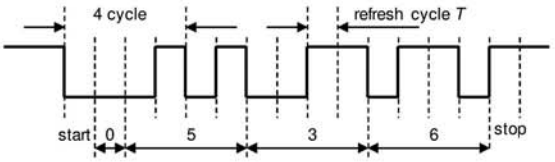


Fig. 10: Baudrate detection method using the first 4bit of keyword[13]

の姓の子音 'S', 'n', 't' の ASCII コードから選んだものである. 受信したデータは PIC の EEPROM 内に 8bit 2byte 毎に 1word 13bit のデータとして格納する. 今回用いた PIC の EEPROM は 256byte であるため, 最大 128word のデータを格納することができる. なお組込み教材がデータを受信し EEPROM に蓄えるか, EEPROM 内のデータを活用して仮想マシンを動かすかは電源投入時のタクトスイッチ (SW1) の状態で切り替えている. タクトスイッチを押したまま電源を投入するとデータ受信モードになる.

3.2 通信速度自動制御

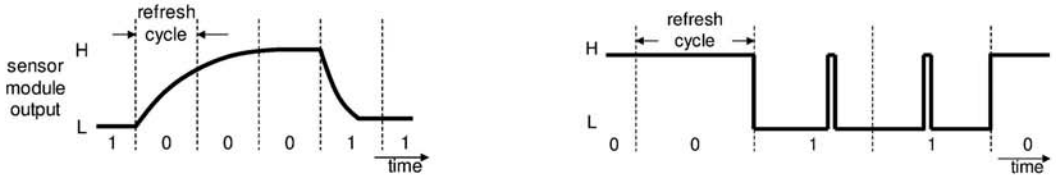
調歩同期方式でデータを送るには一定間隔 (T) でディスプレイの明暗を切り替える必要がある. このタイミングはディスプレイ描画の切り替え (垂直同期: vsync) に合わせて行わなければ画面に筋が入るなど綺麗に切り替わらない. 逆に言えば vysnc に合わせて明暗を切り替えればハードウェアの性能が持つ最高の速度でデータを転送することができる. なお vsync のタイミングは Windows PC では DirectX の機能を用いて知ることができる. vsync の周期はディスプレイのリフレッシュレートの逆数である. 通常の PC におけるリフレッシュレートは 60Hz, Windows Surface 等一部の機種は 59Hz である. そのため教材作成当初は 59.5Hz をターゲットにしていた. しかし当教材は学習者が持ち帰って自宅 PC で活用してもらうことから, この値を前提とすることはあまり望ましくない. 2020 年現在, 極一部の PC は 30Hz といった別の値を持つためである. そこでキーワード読み込み時に vsync の周期 T の推定を行い, それに合わせて明暗判定タイミングを調整する機能を作り, 以後の配布教材には組み込んだ.

4.2 節で述べる教材用開発環境 (送信プログラム) は WEB ページ[14,15] から最新版がダウンロードできる. 初期の教室で配布されたリフレッシュレート 59.5Hz 固定とした古い F/W が書き込まれている教材でも, 最新の開発環境が利用できるためには図 8 で示した通信プロトコルは変えたくない. 幸い, キーワードの最初の値は `0x536` であり, **図 10** のように最初の 1 から 0 の変化に start bit から 4bit 分の時間が経過する. この時間を計測し 4 で割ることで, 推定周期 T_e を得ることができる. 以後は推定した T_e を用いて $T_e/2 + T_eN$ 毎にセンサの状態を読み取ればよい. ここで 4 や 2 の除算はシフト命令で実現することができるため, 教材上の非力な PIC マイコンでも負荷なく演算することができる.

3.3 通信安定化

当教材は様々な PC 環境で動くことを想定して作られ実際に動作している. その中で転送が上手く行かない機種が見受けられた. そのような機種では実際どのようなセンサ信号が出ているのかオシロスコープで確認してみた. 典型的な信号の形状を**図 11** に示す.

図 (a) はディスプレイの応答が悪い機種の信号である. ソフトウェアでは vsync に合わせて明暗を切り替えているのにディスプレイの表示が追いついてない. そこで送信側で vsync 毎に明暗を切り替えるのでなく, 2 回毎, 3 回毎に切り替えることができるようにした. 数回毎に明暗を切り替えることで, 通信速度は落ちるが確実にセンサ信号が切り替わり, 情報を伝えることができる. 受信側は前節で述べた方法で vsync の推定周期 T_e を得ることができる. この T_e が実際のディスプレイの周期 T の 2 倍, 3 倍の値となるだけであり特別なことを行う必要ない. 一方送信側は vsync の発生イベントを数え, GUI で指



(a) Delayed　　(b) Spike noise

Fig. 11: Example of sensor module output for communication failure

定された既定回数が起きてから明暗を切り替えるようにすればよい.

一方図 (b) は輝度を PWM で調整している機種の信号で, 最大輝度で白を表示してもデューティ比が完全に 1 にならないものの例である. 定期的に 100us 程度の黒が入り, センサが明暗を判別するタイミングと重なると受信データが化け, チェックサムではじかれる. ここで受信データが化けない (1 を 1 と間違えず読み取る) 確率は, リフレッシュレート 60Hz の場合 $1 - 0.0001/(1/60) = 99.4\%$ となりめったに起きないように思える. しかし仮に 13bit 40words のデータを通信する場合, その半分が 1 だと仮定すると通信が成功する確率は $0.994^{13\times40\times0.5} = 21\%$ となり 5 回に 4 回は失敗してしまうことが分かる. そこで図 9 (b) における明滅状態を読み取る時刻に, 一回だけセンサを確認するのでなく, 常に 248us 間隔でセンサを確認し過去数回の状態から 0/1 を判定するようにした. 248us (PIC マイコンのタイマ割込みの関係上このような値となっている) はリフレッシュサイクルに対して十分小さく, PWM ノイズに対して十分大きいためノイズを消すには十分な間隔である.

またディスプレイ近くに照度センサを内蔵し自動輝度調整できるノート PC もある. この機能が有効だと輝度が動的に変わり転送が失敗することが多い. Windows の設定で自動輝度調整は off にしておいた方がよいことも付記しておく.

4　仮想マシン

4.1　命令形態

光転送で得たデータは EEPROM に保存される. PIC では EEPROM のデータをネイティブな命令として実行することができない. そこで PIC 内に仮想マシンを構築した. 仮想マシンは EEPROM 上のデータを 1word ずつ読み取り, 仮想マシンのコードとして実行する. 命令の設計思想は, プログラミングの楽しさと, 与えられた命令を順次愚直に実行するというコンピュータの概念を理解してもらうことを第一とした. この場合, LED やモータの制御命令と簡単な条件分岐やループ命令があればよく, 変数や算術演算などの機能はなくてもかまわない.

しかし当教材を自宅に持ち帰った者の中には, より複雑な処理を行わせたいと思う者がいるかもしれない. そのため仮想マシンは現実のデジタルコンピュータと類似した構成とし, 変数や算術演算等を用いた複雑なプログラムも作れるよう工夫した. **図 12** に仮想マシンのアーキテクチャを示す. 各種演算が可能な `a`, `b`, 一次記憶や I/O となる汎用用途の `r1` ~ `r4`, 比較命令等の結果 (フラグ) を格納する `z`, 一番内側のループカウンタを示す `i` といった 8 つのレジスタと, ループ及びサブルーチンコールを各 5 段まで可能とするためのスタック領域 LC, RA 及びそれらのポインタ LS, AS, さらにプログラムカウンタ PC の 21 個の内部メモリを持っている. **表 2** にこの仮想マシンの基本命令を示す. 全ての命令は 13bit の固定長である. ビット数は光転送にかかる時間を短くするには少なく命令を増やすには多く必要で, そのバランスを取った結果である. 命令のうち `loop` と `lend` は `loop` 命令のオペランド I で示した回数のループ処理を実現するもので組で用いられる. それぞれの

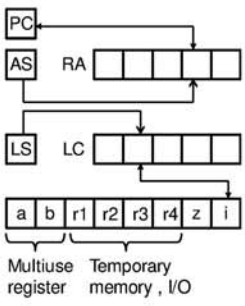


Fig. 12: Architecture of virtual machine[12]

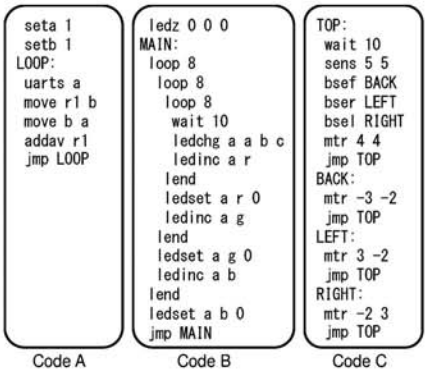


Fig. 13: Sample code of virtual machine[12]

Tbl. 2: Basic command of Virtual Machine

Mnemonic	Description	Opcode
`seta` I	`A = I`	`0-0000-iiii-iiii`
`setb` I	`B = I`	`0-0001-iiii-iiii`
`adda` I	`A += I`	`0-0010-iiii-iiii`
`addb` I	`B += I`	`0-0011-iiii-iiii`
`anda` I	`A &= I`	`0-0100-iiii-iiii`
`andb` I	`B &= I`	`0-0101-iiii-iiii`
`ora` I	`A &= I`	`0-0110-iiii-iiii`
`orb` I	`B &= I`	`0-0111-iiii-iiii`
`cmpa` I	Compare A and I	`0-1000-iiii-iiii`
`cmpb` I	Compare B and I	`0-1001-iiii-iiii`
`loop` I	Start I times loop	`0-1010-iiii-iiii`
`wait` I	Wait I×20 ms	`0-1011-iiii-iiii`
`jmp` A	Jump	`0-1100-0aaa-aaaa`
`call` A	Call subroutine	`0-1100-1aaa-aaaa`
`lend`	Loop end	`0-1101-0aaa-aaaa`
`break`	Break loop	`0-1101-1aaa-aaaa`
`beq` A	Branch if equal	`0-1110-0aaa-aaaa`
`bne` A	Branch if not equal	`0-1110-1aaa-aaaa`
`bgt` A	Branch on greater than	`0-1111-0aaa-aaaa`
`blt` A	Branch on lower than	`0-1111-1aaa-aaaa`
EXPANSION AREA		`(0x1000 ~ 0x1eff)`
`move` R_x R_y	`[Rx] = [Ry]`	`1-1111-00xx-xyyy`
`rora` N	`A >>= N`	`1-1111-0100-0nnn`
`rorb` N	`B >>= N`	`1-1111-0100-1nnn`
`rola` N	`A <<= N`	`1-1111-0101-0nnn`
`rolb` N	`B <<= N`	`1-1111-0101-1nnn`
`rord` N	`AB >>= N`	`1-1111-0110-0nnn`
`rold` N	`AB <<= N`	`1-1111-0110-1nnn`
`addav` R_x	`A += [Rx]`	`1-1111-1000-0xxx`
`addbv` R_x	`B += [Rx]`	`1-1111-1000-1xxx`
`subav` R_x	`A -= [Rx]`	`1-1111-1001-0xxx`
`subbv` R_x	`B -= [Rx]`	`1-1111-1001-1xxx`
`andav` R_x	`A &= [Rx]`	`1-1111-1010-0xxx`
`andbv` R_x	`B &= [Rx]`	`1-1111-1010-1xxx`
`orav` R_x	`A |= [Rx]`	`1-1111-1011-0xxx`
`orbv` R_x	`B |= [Rx]`	`1-1111-1011-1xxx`
`xorav` R_x	`A ^= [Rx]`	`1-1111-1100-0xxx`
`xorbv` R_x	`B ^= [Rx]`	`1-1111-1100-1xxx`
`cmpav` R_x	Compare A and `[Rx]`	`1-1111-1101-0xxx`
`cmpbv` R_x	Compare B and `[Rx]`	`1-1111-1101-1xxx`
`uarts` R_x	Send `[Rx]` to UART	`1-1111-1110-0xxx`
`uartr` R_x	Receive UART to `[Rx]`	`1-1111-1110-1xxx`
`waitv` R_x	Wait `[Rx]` × 20 ms	`1-1111-1111-0xxx`
`ltop`	Clear loop nest	`1-1111-1111-1100`
`skipuart`	Skip next if no UART data	`1-1111-1111-1101`
`ret`	Return from subroutine	`1-1111-1111-1110`
`err`	Make an error state	`1-1111-1111-1111`

Tbl. 3: LED control command of Virtual Machine

Mnemonic	Description	Opcode
`leda` I_r I_g I_b	Set LED-A as `[Ir,Ig,Ib]`	`1-000r-rrgg-gbbb`
`ledb` I_r I_g I_b	Set LED-B as `[Ir,Ig,Ib]`	`1-001r-rrgg-gbbb`
`ledc` I_r I_g I_b	Set LED-C as `[Ir,Ig,Ib]`	`1-010r-rrgg-gbbb`
`ledd` I_r I_g I_b	Set LED-D as `[Ir,Ig,Ib]`	`1-011r-rrgg-gbbb`
`bbtp1` A	Branch if btn1 pushed	`1-1000-0aaa-aaaa`
`bbtn1` A	Branch if btn1 unpushed	`1-1000-1aaa-aaaa`
`bbtp2` A	Branch if btn2 pushed	`1-1001-0aaa-aaaa`
`bbtn2` A	Branch if btn2 unpushed	`1-1001-1aaa-aaaa`
`ledinc` L C	Increment `[C]` of `[L]`	`1-1010-0000-LLcc`
`leddec` L C	Decrement `[C]` of `[L]`	`1-1010-0001-LLcc`
`ledcld` L	Load `[L]` to r1~ r3	`1-1010-0010-00LL`
`ledcst` L	Store `[L]` from r1~ r3	`1-1010-0010-01LL`
`ledset` L C I	Set `[C]` of `[L]` as `I`	`1-1010-1LLc-ciii`
`ledchg` ℓ_a ℓ_b ℓ_c ℓ_d	Swap LED color	`1-1011-AABB-CCDD`
`ledz` I_r I_g I_b	Set all LED as `[Ir,Ig,Ib]`	`1-110r-rrgg-gbbb`

Tbl. 4: Robot control command of Virtual Machine

Mnemonic	Description	Opcode
`mtr` I_r I_l	Set motor power `[Ir,Il]`	`1-0000-rrrr-llll`
`addmtr` I_r I_l	add motor power `[Ir,Il]`	`1-0001-rrrr-llll`
`bbtp` A	Branch if btn pushed	`1-0010-0aaa-aaaa`
`bbtn` A	Branch if btn unpushed	`1-0010-1aaa-aaaa`
`sens` I_r I_l	Sense obstacle by `[Ir,Il]` power	`1-0011-rrrr-llll`
`bsne` A	Branch if obstacle not exists	`1-0100-0aaa-aaaa`
`bsef` A	Branch if obstacle exists front	`1-0100-1aaa-aaaa`
`bser` A	Branch if obstacle exists right	`1-0101-0aaa-aaaa`
`bsel` A	Branch if obstacle exists left	`1-0101-1aaa-aaaa`
`ledon`	light up LED	`1-1110-0000-0000`
`ledoff`	light down LED	`1-1110-0000-0001`
`mtrld`	Load motor power to r_1, r_2	`1-1110-0000-0010`
`mtrst`	Store motor power from r_1, r_2	`1-1110-0000-0011`

内部処理を C 言語風に書くと “`LC[++LS]=`I`;`”, “`if (LC[LS]--) PC=`A`; else LS--;`” となる. `lend` 処理内の飛び先 A は, 対となる `loop` 命令の次のアドレスである. アドレス情報がオペコードにあるもののオペランドが無いのはアセンブラが該当アドレスを計算できるためである. 同様にループから強制的に抜け出す `break` 命令も, 内部では “`LS--; PC=`A`;`” といった処理をしており, 飛び先 A を示す `lend` 命令の次のアドレスもアセンブラが計算できるためオペランドは省略されている.

表のようにコンピュータとして基本的な命令は網羅しており, メモリさえ許せばほとんどの演算処理が可能なことが見て取れる. また `0x1000 ~ 1eff` までの 3840 個の空間は拡張空間であり, ここに**表 3** のような LED 基板用の命令, もしくは**表 4** のようなロボット制御用の命令を展開する. **図 13** にこの仮想マシンのサンプルコードを示す. サンプル A は基本コードのみを使ったもので, フィボナッチ数列を UART から出力するものである (ただしレジスタおよび UART 出力は 8bit なため 13 番目までしか正しい値にならない). レジスタ `a`, `b` 及び `r1` を用いて, はじめに `a`, `b` に 1 を代入した後, C 言語風に書くと “`r1=b; b=a; a+=r1;`” といった処理を繰り返している. 次にサンプル B は LED 基板で光を回転させつつ 512 色順に出力するものである. はじめにすべての LED を消灯した後, 三重のループに入り一番深いループで 200msec の停止, 表示色の回転, 赤の輝度を 1 段階ずつ 8 段上げていき, 外側のループで緑および青の輝度を同様に上げている. 最後のサンプル C は移動ロボットが障害物を回避しつつ進むものである. 障害物センサの状態を読み取り, 前方に障害物があれば左旋回しつつ後退, 右に障害物があれば左旋回, 左に障害物があれば右旋回, 何もなければそのまま前進という処理を繰り返している.

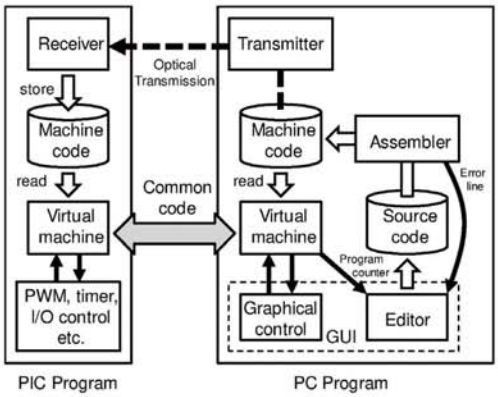


Fig. 14: Common code of virtual machine[12]

Fig. 15: Integrated development environment of educational emvedded system

4.2　シミュレータ

PIC マイコンの F/W は C 言語で記述しメーカ (Microchip 社) が提供しているコンパイラ XC8[10] を用いてコンパイルした. 標準的な C 言語で記述したためタイマや I/O ポート等マイコン固有の部分を除けば Visual C++ 等の PC 用コンパイラでもコンパイルすることができる. そのため仮想マシンのコアとなるコードの解釈実行部分を PC 上で走らせれば, 本物の基板と同じように動作するシミュレータを構築することができる. PC はユーザインタフェイスが豊富なため一時停止やステップ実行といったデバッグやレジスタの値や LED の状態, モータへの出力値を GUI で操作確認することもできる. ステップ実行を分かりやすく表現するためには, 今コードのどこを実行しているか画面表示する必要がある. すなわちアセンブルされた仮想マシンコードだけでなく, アセンブル前のソースコードもシミュレータに取り込まなくてはならない. そこでシミュレータそのものの他にソースコードの編集機能, アセンブラ機能, ディスプレイ明滅によるコード転送機能等を備えた統合開発環境を構築した. **図 14** に PIC 上の仮想マイコンと統合開発環境上の仮想マイコンの関係を示す. 図のように二者の仮想マシンのコードは同一となっており, まず統合開発環境上の仮想マシンで動作確認してから光転送で組込み教材にコードを送って動かすことが出来るしくみとなっている.

Microsoft Visual C++ 2008 Express Edition で作成した統合開発環境の外見を**図 15** に示す. 生成したコードは Windows の実行形式 (exe ファイル) 一つで動作し, 別途ドライバやライブラリをインストールする必要はない. このうち (A) は LED 教材向けの開発環境である. 図中左上部のボタンでソースファイルの管理とシミュレータの操作, その下でソース編集, 右上で LED とボタンの模擬を実現し, 右下は光転送部分となっている. また右中央は情報提示部分でステップ実行時は今の命令のアドレス, 各 LED の色, 各レジスタの値等のデバッグ情報を表示する. 一方コード編集中はドロップダウンリストで表 2, 3 に示した全命令が選択可能となり, 選択時はその命令のチュートリアルも表示する. ドロップダウンリストで選択された命令は編集画面に差し込むこともできる. これによりキーボードの苦手な子供でも比較的容易にプログラミングすることが可能である. ここで色の指定命令 (`leda` 等) のパラメータは RGB の輝度を 0~ 7 の 8 段階で指定する. この部分を Windows の標準機能である色選択ダイアログ (ColorDialog) を用いて GUI で操作できればいいという意見があった. しかしトライアンドエラーで気に入った色の RGB 値を探すという行為自体が楽しいのではないかと考えあえて実装していない. 一方 (B) はロボット教材向けの開発環境である. LED 教材に対し異なるのは右上の部分のみで, モータの出力を表示したり, 左右の障害物センサが反応したかどうか入力できたりするようになっている. 最終的に障害物を自由に配置できる仮想フィールド上でロボットの軌跡を表示していくシミュレータが作れれば面白いと感じているが実現には至ってない.

5　実験授業

本教材は久留米市主催の小学生対象の親子科学教室 (**図 16**), 当学学生対象のものづくり実践プロジェクト (**図 17**), 高校生対象の高大連携や一日大学生, などで活用されている. このうち親子科学教室は久留米市の生涯学習センタ (えーるピア久留米) が毎年 7 ~ 8 月に行う公開講座で (2020 年は中止), 小学生とその保護者の中から希望者を募って抽選で行われるものである. 当教材を用いた教室は 3~ 6 年生対象に 12 組で公募をかけ開かれた.

教室は 2018 年, 2019 年の 7 月末に開催し, 2 時間程で以下のようなカリキュラムとした.

1. 図 15 (A) で示したシミュレータを用いて, 全 LED が赤緑青順に点滅するプログラムの入力と実行

Fig. 16: Photograph of open class using the LED system[13]

Fig. 17: Photograph of project class using the robot system

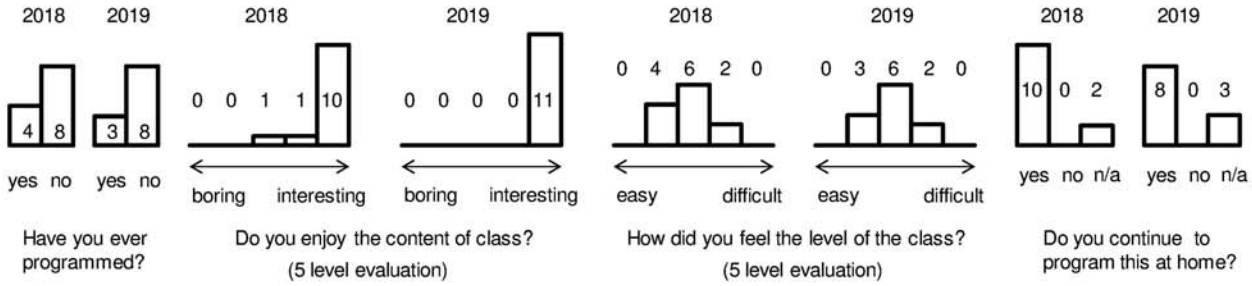


Fig. 18: Questionnaire results of classes using the emvedded system in 2018 and 2019

2. 打ち込んだプログラムを光転送を用いて組込み教材に転送・実行できることの体験
3. 1. のプログラムの解説 (`ledz`, `wait`, `jmp`)
4. 光の三原色とそれを調整する方法について, 各 LED を個別に制御する命令 (`leda`, `ledb`, `ledc`, `ledd`) を交えて説明
5. 各自好きな色やパターンで教材を光らせるプログラム体験
6. 色の入れ替え命令 (`ledchg`) と繰り返し命令 (`loop~ lend`) の説明
7. 各自何かテーマを決め, そのイメージに合う光り方をする作品を制作
8. 作品発表

このように教室で使った命令は 9 つのみであった. それでも様々なバリエーションの明滅パターンを持った作品が作られた. 教室終了後「このボタンは何のためあるのか」といった質問も受けた. 基板には 2 個のタクトスイッチが付いているのに教室ではその片方をデータ受信モードの切り替えに使っただけだったためである. 実はもっと沢山の命令があってその中にはボタンが押されたか判断するものもある, それらは WEB ページ[14] で紹介していると伝えた.

教室を開催する前は小学生がキーボードで英文字を入力できるか不安であった. しかし保護者が同席していたためか先述したドロップダウンリストで命令を選択したためか特に問題となることはなかった. その代わり大文字と小文字の対応に苦慮する者は数例見かけた. 当シミュレータのアセンブラは大文字小文字区別しないいわゆるケースインセンティブな体系としている. 可読性の観点から命令は小文字, ラベルは大文字という表記ルールで資料を記述していたが, それが逆に障害となっていた. この問題は今年度 (2020 年度) からプログラミングだけでなく英語も小学校で必修化されることから次第に解決するのではと期待している.

教室終了後行ったアンケートの結果を**図 18** に示す. 母集団が「科学教室の参加を希望した組」と偏りがあり, 「主催者が行った質問」なので忖度も混じっていると考えられ, これを一般的な評価と考えるのは危険ではある. しかし, 少なくとも否定的な意見はなく, またほとんどの方が自宅でも続けて行ってみたいという感想を持ったことから一定の効果はあったと考えている.

6　おわりに

PIC マイコン内に仮想マシンを構築し, 仮想マシン用コードを PC のディスプレイの明滅で転送する組込み教材を作成した. この構成はライタを必要としないため, 教室終了後に用いた教材を持ち帰り自宅で学習を続けることができる. 試作した教材は LED の色や輝度を制御する基板と 2 つのモータと障害物センサを備えた小型移動ロボットの二種類である. これらは同じ命令体系の仮想マシンを基礎とし, 各教材固有の拡張を行っている. 今後さらに別の教材への応用も行っていきたい.

一方で現在の横文字テキストベースのプログラミングスタイルから脱却する方向も検討している. 例えば `jmp` や `cmpa` などの命令について, ある程度英語やプログラムに慣れた者であれば自然と “jump” や “compare (from) A” と読み替え, 内容を類推することができるが, 小学生には難しいかもしれない. 例えば “へとぶ”, “を A とくらべる” など日本語表記にする方法がある. これはアセンブラプログラムにおいてニモニックの文字列比較部分を書き換えるだけでよく実装は簡単である. しかし実際に運営するとなると日本語 (ひらがな) をキーボードから入力させなくてはならず逆にスムーズにいかない可能性がある. タイルプログラミング等のグラフィカルな開発環境への方向も考えられる. 個々の命令に対応したタイルをマウス等で並べるような GUI を作ることは工数はかかるが技術的問題はない. 近年は家庭内の情報機器としてスマートフォンやタブレットが主流となってきている. タイルプログラミングは GUI 操作がメインでありこれらの端末と相性がよい. また光転送はディスプレイさえあれば成立することもタブレットのような機器向きと言える. ただしそのようなグラフィクスプログラミングスタイルが教材としてふさわしいかどうかについては検討する必要がある.

当教材は一日大学生や高大連携, 子供向け公開講座などで活用している. 教室開催後のアンケート結果は概ね良好である. 今後も当教材を用いて, 科学やプログラミングの楽しさについて啓蒙できればと考えている. なお, 本稿で述べたシミュレータは WEB ページ [14,15] よりダウンロード可能である.

当教材を用いた親子科学教室を実施するにあたり, 当学当科の江藤信一教授に助力を頂いた. ここに謝意を表する. また本研究は 2019 年度久留米工業大学学長裁量経費の助成を受けたものである.

文　献

(1) 小学校プログラミング教育の手引き (第三版), 文部科学省, 2020
(2) Scratch, `https://scratch.mit.edu`
(3) 酒井, 長谷川, Sphero を用いた小学校プログラミング学習単元の開発, 日本科学教育学会研究会研究報告, 31-8, pp.117–122, 2017
(4) 塚原, 松嵜, プログラミングを取り入れたモデリング授業の実践報告, 日本科学教育学会研究報告, 31-6, 2016
(5) 木室, 千田ほか, 簡易言語とファームウェア改造による移動ロボット制御授業, 信学技報告, ET106–166, pp.5–10, 2006
(6) 水谷, 岩本, 教育用ロボット梵天丸を用いた小・中学生のためのプログラミング教育, 信学技報告, ET106–166, pp.43–48, 2006
(7) 野口, 梶原, プログラミング教育のためのロボット教材, 情報教育シンポジウム論文集, 2017(20),145-151, 2017
(8) 岩本, 水谷, 小中学校における制御教育− 図画工作から始める科学技術教育− , 計測と制御, 46-9, pp.283-286, 2007
(9) PICkit3, `http://www.microchip.com/PICkit3`
(10) MPLAB XC Compilers, `https://www.microchip.com/mplab/compilers`
(11) ELEKIT WEB World, `https://www.elekit.co.jp/product /promo/kirobo/about.html`
(12) 千田, 光転送と仮想マシンを用いた教育用組込みシステム, 信学技術報告, ET2018–12, pp.1–6, 2018
(13) 千田, ディスプレイを用いた簡易データ転送, SICE 九州支部講演会, pp.69–70, 2019
(14) 虹色ランタンシステム, `http://sentalab.kurume-it.ac.jp/research/ColorLED/index.html`
(15) プログラミングロボットシステム, `http://sentalab.kurume-it.ac.jp/research/IRRobot/index.html`