# チームワーク志向の学生実験について

鈴木 貢[†] 南 宣正[†,☆☆] 前田 洋一[†,☆]
河野 健二[†] 楯岡 孝道[†]
阿部 公輝[†] 渡邊 坦[†]

プロセッサ(P)とコンパイラ(C), ネットワーク(N)という要素を統合的に取り扱う学生実験(CNP実験)を学部3年生を対象に企画し実施している. この実験では, 各6人のチームで2人ずつが上記それぞれを担当しながら, P担当者がLAN付きコンピュータシステムを市販のFPGA評価ボード上に実現し, C担当者がそのためのクロスコンパイラを作成し, N担当者がそれを使って普通のUnixと通信可能なUDP/IPのプロトコルスタックと適当な検証プログラムを記述するという作業を平行して進め, 最後に3つの要素を合体させて動かす. この実験の成立には, 実験時間やデバイスの容量等 いくつもの制約をクリアしなければならなかった. ここでは, 実験を設計・実現するにあたって顕在化したそれらの制約をいかにクリアしたかを, コンパイラ部分の担当者の立場で報告する.

## 1. はじめに

プロセッサ, コンパイラ, それにコンピュータネットワークは現代の計算機システムを構成する重要な要素であり, 我々も関連する授業やそれらに対応する学生実験を実施してきた. これらは講義と連携して計画的に実施される一方で, 内容がいささか陳腐化し学生に退屈感を与えてきつつあったことも否めない.

そこで我々は,現代的な計算機システムを構成する重要な要素としてプロセッサ(アーキテクチャ), コンパイラ, ネットワークという3つの要素を選び, それらの要素を統合的に取り扱う,学部3年生を対象とする学生実験(CNP実験) を企画し, 2001年度から実施を開始した. Cはコンパイラ, Nはネットワーク, Pはプロセッサを意味する. ここでは,次の項目を達成することを目標とした.

(1) 計算機システム全体構成を体得できる内容であること.
(2) 充分スリルや達成感を感じられる程度の難しさをもった内容である一方, 限られた実験時間(合計3時間×12日)で実施できる程度に単純かつ容易であること.
(3) 取り扱う題材が充分実用的かつ今日的であること.
(4) 学生に与える素材には,大きなブラックボックスがないこと.

結果的に,実験のストーリーとゴールを次のように決めた. すなわち,学生が作成したネットワークプロトコルスタック(TinyIP)を, 学生が作成したコンパイラ(TinyC)でコンパイルし, 学生が作成した計算機システム(MinIPSシステム)で実行するというものである. TinyIPはUDP/IPの実装であり, 実験では標準的なワークステーションと通信する.

この種の大規模な学生実験を実施する場合,東京大学理学部情報科学科のプロセッサ実験[1] のように半年(週3回×12週)程度を実験時間に割り当て, プロセッサの命令セットの設計や基板の配線からコンパイラの設計や製作に至るまで, 学生がゼロから全てを考え, 実施するという例もある. この方針は,学生がシステム全体について非常に深い知識を獲得し得る反面, 学生に比較的長期に渡る集中力の持続を要求し, 最終的にゴールまで到達できる学生が非常に少なくなると考えられる.

この実験の企画にあたって何よりも重視したのは, 学生が本物のシステムを作ったという達成感を得て, それが後の研究や開発において精神的・技術的な支えになることと, システムの構築におけるチームワークの有用性を体得することである. そのためには, 学生の創意工夫を引き出しながら,それなりに努力したチームならば最終ゴールにまで到達できるように実験を企画しなければならない. そして, 学生が必ずゴールに達することができるという確信をもって, 実験に取り組めるよう, 教官側がしっかりとしたプロトタイプを用意していることが重要である.

† 電気通信大学大学情報工学科
Dept. of Computer Science of The University of Electoro Communications
☆ 現在, 日立製作所
Presently with Hitachi,Ltd.
☆☆ 現在, 富士フィルムソフトウエア(株)
Presently with FUJIFILM SOFTWARE Co.,Ltd.

そこで学生は6人ずつのチームに分かれ, 各チームは各々2人づつがC実験(コンパイラの作成),N実験(ネットワークプロトコルの作成), P実験(プロセッサの作成)を実施するようにした. そして,P実験とN実験ではプロトタイプに設けた穴を補うことを必修課題とし, C実験では付属の例題を翻訳できる処理系に, 設定された機能拡張を施すことを必修課題とする. そして最終的に, これらの必修課題を全てパスすれば, ゴール, すなわち作ったシステムがLinuxと通信できるようにした. 各必修課題は, 担当部分の構成を体得できるように考慮されている一方で, 高い能力を有する学生に対しては, 創意工夫を引き出すべく, プロセッサの命令セットの拡充や必修課題ではネットワークプロトコルから省かれている機能の拡充, コンパイラへのさらなる機能追加を自主課題として実施するように指導し, それらを実施した学生に対してはより高い評価を与えた.

最初の10回は計算機システムのシミュレータを活用して並行して担当分の開発を進め, 最後の2回で3つを統合する. この実験の運営やプロトタイプ全体については文献[2]を, P実験の詳細は文献[3]を, C実験の詳細は文献[4]を参照して欲しい.

この規模の実験は, プロトタイプの作成も含め, 実験の準備自体も一人では困難である. それぞれを3人で分担し, 担当者間で仕様の擦り合わせを行いながら2年の準備期間をかけて用意した. その際, 実験時間や使用するFPGAのキャパシティー, 学生の能力といった多くの制約をクリアしなければならなかった. 本報告では, 中間管理職的な役割を持つコンパイラ担当者の立場から, それらをどう克服したかを振り返る.

## 2. プロトタイプの設計と実装

実験を構成する3つの要素のプロトタイプの設計と実装について述べる.

### 2.1 P実験のプロトタイプ

プロセッサを含むコンピュータシステム(MinIPSシステム) の概略を**図1**に示す. 本報告では, コンパイラに関連する部分に絞って説明する.

#### 2.1.1 プロセッサ

システムの中核をなすプロセッサとしては**表1** のような命令セットをもつMinIPSアーキテクチャ[5]を用いた. これはMIPSアーキテクチャ[6]の命令セットアーキテクチャのサブセットである. 省かれた命令は, 主に浮動小数点関係と乗除算命令である. また, 割り込み制御レジスタの仕様が変更されている.

MinIPSは, 上記の命令セットを5段(IF, ID, Ex, Mem, WB) パイプライン構成のプロセッサとして実現されている. フォワーディングユニットのおかげでコンパイラやアセンブラによるレジスタの定義-使用依存の対策は不要であるが, ロードと分岐の遅延はそれぞれ1なので, コンパイラ等での対策が必要である.

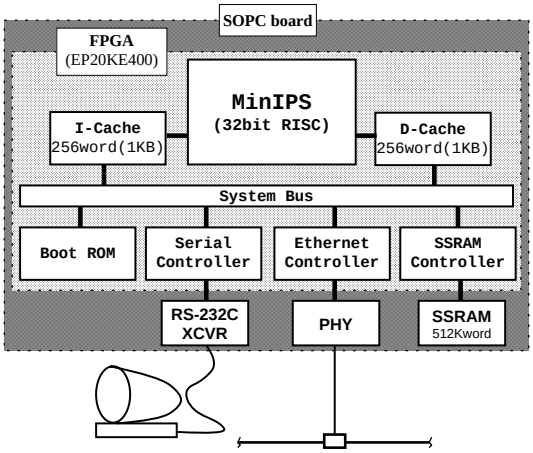


**図1** MinIPS システム

**表1** MinIPS の命令セット

| 区分 | 命令(ニモニック) |
|---|---|
| 算術演算 | add, sub, addi, addu, subu, addiu |
| 論理演算 | and, or, andi, ori |
| シフト演算 | sll, srl, sra, sllv, srlv, srav |
| データ転送 | lb, lw, sb, sw, lui, mfe, mte |
| 比較 | slt, slti, sltu, sltiu |
| 条件分岐 | beq, bne |
| 無条件分岐 | j, jr, jal, rfe |

#### 2.1.2 MinIPSシステム

コンピュータシステムの実装には, Altera社の評価用ボードSOPC (System in a Programmable Chip) ボード[7]を利用した. このボードはFPGA(40万ゲート相当+20KBの内部メモリ)や, メモリ(SDRAM 64MB+SSRAM 2MB)や各種I/O(10/100Base Ethernet+シリアル+etc)がワンボードに集約されている. 搭載されているI/Oはコントローラの形ではなく, EthernetのPHYチップや, RC-232Cのレベルコンバータ等と, それぞれの規格の端子のみなので, コントローラの機能をFPGAに実現する必要があるが, 全て自前で用意した. I/Oやメモリの制御に商用のIP(Interectual Property)を一切使用していないので, コンピュータシステムのハードウエアの設計でも全てがホワイトボックスである. シリアルコントローラは文献[6]のMIPSエミュレータの仕様にあわせた. Ethernetコントローラは独自の仕様をもち, FIFOを使っ

てデータのやりとりを行う．両者とも極めて単純な構成であるが, 送受信の割り込みをサポートするなど, 充分実用的なものである.

システム起動時に初期化やブートローディングを行うプログラムは, FPGA の内臓メモリに実装し, FGPA にコンピュータシステムを流し込む時に設定される．このプログラムは, メモリの設定と指定番地への飛び込みの機能しか有しないので, これを用いて より高度な操作内容をもつモニタプログラムをロードする.

### 2.2 N 実験のプロトタイプ

Internet Protocol(IP) をネットワーク層で, User Datagram Protocol(UDP) をトランスポート層に用いたプロトコルスタックを用意した．これでは階層化モデルを用いており, Ethernet パケットの受信には割り込みを用いている．その構造は, 新機能の追加が容易で, ハードウエア依存部を `hardware.c` に集約した設計になっている．パケットのフラグメント化, Address Resolution Protocol (ARP), Internet Control Message(ICMP) を省いているが, Linux 等に実装された標準的な IP スタックと通信ができる.

`TinyIP` は ANSI C と `TinyC` のそれぞれで実装されている．ANSI C 版は約 800 行のステートメントと約 400 行のコメントから, 一部機能を省いた `TinyC` 版は約 500 行のステートメントから, それぞれ構成される．ANSI C 版は Linux 上で動作し, プロトコルスタックの理解を助ける．`TinyC` 版は Linux および エミュレータ上で動作確認し, 最終的に実機で動作させる．この `TinyC` 版を Linux の gcc でコンパイルして動作確認するという事項が, `TinyC` に付加する機能に対して一つの制約となり, ANSI C には無い機能を付け加えることができなくなってしまった.

### 2.3 C 実験のプロトタイプ

#### 2.3.1 オリジナルの TinyC コンパイラ

オリジナルの `TinyC`[8] は, 渡邊によって開発された非常に小さな C 言語のサブセットで, 講義科目 (言語処理系論) で例題として使われいる．その仕様は, el($\alpha$)[9] の経験等を生かしながら注意深く設定された.

`TinyC` コンパイラの概略は次の通り: (1)C++で記述された 1 パス構成のコンパイラで, 手書きの再帰下降構文解析器をもち, (2) **表 2**, **図 6** のような C 言語のサブセットを入力とし, MIPS のアセンブリコードを出力し, (3) 基本ブロック単位で 変数のレジスタへの割り当てを行う．（しかし複雑な式の処理に対するレジスタの溢れ処理を行っていない）

ソースプログラムの構成を **表 3** に示す．機種依存部は `mips.C` と `mips.h` に集約されている．また, 機種依存部と構文解析器は抽象レジスタマシン (`ARM`, Abstruct Register Machine) 層で分離されている．これにより, `TinyC` は RISC 方式の命令セットアーキテクチャであれば, 比較的容易にリターゲットできる.

**表 2** TinyC の言語仕様

| 項目 | 仕様 |
|---|---|
| データタイプ | void と int のスカラと 1 次元配列. 構造体はない. |
| 変数クラス | グローバル変数のみ. |
| 定数 | 10 進整数のみ |
| 式 | 算術式, 比較式 |
| 文 | 代入, 手続き呼び出し, if, while, return, ブロック |

**表 3** TinyC のソースプログラムの構成 (計 2163 行)

| ファイル名 (行数) | 内容 |
|---|---|
| sub.C(451), sub.h(301) | 字句解析器と記号表 |
| tinyC.C(521) | 抽象マシン ARM のコードを生成する構文解析器. main() を含む. |
| arm.C(290), arm.h(185) | ARM の実現. 変数や式の中間値のレジスタへの割り付けを行う. |
| mips.C(360), mips.h(55) | ARM 命令を特定のプロセッサ (オリジナルでは MIPS) の命令に展開する. |

授業との連続性を考慮して, 学生にはオリジナル `TinyC` で乗除算命令を使用している個所をサブルーチンコールに変更しただけのものを渡し, それに 5 節で述べる機能を付加することを必修課題とした．Web や教科書で `TinyC` の完全なプログラムが入手可能であるので, オリジナル `TinyC` に対する穴埋めを課題にはできない.

## 3. 乗除算の実装

MinIPS のお手本になっている MIPS の命令セットには, 整数の乗算も除算も含まれているので, MinIPS にそれらを実装することはごく自然なことである．鈴木は, 前田 (プロトタイプ MinIPS システムの設計者) に符号付乗除算命令の追加を要請したが, ハードウエア的な制約から受け入れられなかった．彼の試行によると, 符号なし整数の 32bit×32bit の乗算器は, FPGA 開発ツールの Mega Function(FGPA メーカーが供給する最適化された IP) でもロジックエレメントの 5%を消費してしまう．乗除算を含まない実装でも約 60%に達してしまっているので, これが 70〜80%に達すれば, 配置配線が不能になるか, 可能だとしても処理時間が学生の実験時間を圧迫するであろう．また, 除算は複数クロックで実施されるので, プロセッサ制御部への演算のストールの考慮が必要になり, プロセッサの構

```
w = 0;
for (31回繰り返す) {
  switch (bの先頭の最上位の2bit) {
    case 00: w = w << 1; break;
    case 00: w = w << 1; break;
    case 01: w = (w + a) << 1; break;
    case 10: w = (w - a) << 1;
  }
  b = b << 1;
}
if (b < 0) w = w - a;
```

**図2** 通常のBooth法の除算

造に大きな見直しが必要になる.

UDP/IPの実装だけなら乗除算は不要か, あるいは必要でもシフト演算の組み合わせで間に合わせることができるであろう. しかし鈴木があえて乗除算にこだわったのは, TinyCのキットに付属しているエラトステネスのふるいで素数を求めるプログラム(オリジナルのTinyCでコンパイル可能)を学生への説明のデモで「つかみ」に使おう決心していたからである.

そこで, コンパイラで乗除算命令を発行している箇所を, 乗除算ルーチンを呼び出すように改造し, 符号付整数の乗除算ルーチンをアセンブラで作成した.

### 3.1 乗算サブルーチン

a×b を計算する場合, 通常のBooth法では, 積を格納する変数をwとして, **図2**のようにbの最上位の2bitを見ながらループを繰り返す.

2bitずつ切り出して比較するのは, アセンブラで書くにはいかにも非能率的である. 鈴木は, ソースレジスタ1の上位2bitの値に従って「01→ソースレジスタ2の値, 10→その2の補数値, 00 or 11→0」を選択し, ディスティネーションレジスタにストアする命令(いわゆる掛算ステップ命令)の増設を提案したが, 命令セットが汚くなるという理由で却下された.

ところで, 切り出すのが2bitなのに対して, シフトするのは1bitであることに注目すると, ループのひとつ前の回でswitch文で実行したところ(00,01,10,11のひとつ)と, シフトの結果30bit目に入ってきた値で動作を決めることができる. つまり4状態1bit入力のステートマシンとして, **図3**のようなコードに書き直すことができる. (MSBのチェックは比較的安価である.) これをアセンブラで実装して, システムのスタートアップコードに組み込んだ.

### 3.2 除算サブルーチン

wとxをそれぞれ被除数と除数とし, qとrにそれぞれ商と余りを格納するとしよう. 大抵の論理設計学の教科書に載っている引きっぱなし法による除算アルゴリズムは**図4**のようになっている. これでは, 32bit÷32bitを求めるのに64bitのレジスタを用いなければならない. これは, MIPSのようなキャリーや拡張フラグを持たないアーキテクチャでは, 実装が難しい. 最終的に剰余はrの下32bitに入るが, 見方を変えればxxの元々のxの値を埋め込まれた位置(ループを回る度に右にシフトされていく)に対応する部分に入るともいえる. そこで, **図5**のようにxxを固定し, rをループを回る度に左に1bitずつシフトすることでも正しく剰余を求めることができ, xxは32bitで十分であることに気づいた. さらに, 商と剰余の補正を図5のようにアセンブラ向きに修正したものをアセンブラで実装して, システムのスタートアップコードに組み込んだ.

```
count = 30;
if (bのMSBが1) goto s1x;
s0x: b = b << 1;
     if (bのMSBが1) goto s01;
s00: w = w << 1;
     if (c-- == 0) goto end;
     b = b << 1;
     if (bのMSBが0) goto s00;
s01: w = (w + a) << 1;
     if (c-- == 0) goto end;
     b = b << 1;
     if (bのMSBが1) goto s11;
s10: w = (w - a) << 1;
     if (c-- == 0) goto end;
     b = b << 1;
     if (bのMSBが1) goto s01;
     goto s00;
s1x: b = b << 1;
     if (bのMSBが0) goto s10;
s11: w = w << 1;
     if (c-- == 0) goto end;
     b = b << 1;
     if (bのMSBが1) goto s11;
     goto s10;
end: if (b < 0) w = w - a;
```

**図3** 状態機械を使ったBooth法による乗算

## 4. 南によるTinyCの拡張

標準的な学部3年生がどの程度TinyCを理解し, それに手を加えることができるかを見極め, どこに要点

```
  long long  r = w, xx = x << 31;
  q = 0;
  for (32回繰り返す) {
    if (SAME_SIGN(r, x)){ // 同じ符号?
      r = r - xx; q = q * 2 + 1;
    } else {
      r = r + xx; q = q * 2 - 1;
    }
    xx = xx >> 1;
  }
// 商と剰余の補正
  if (((r != 0) && !SAME_SIGN(r, w)
                && SAME_SIGN(r, x)) ||
      ((r == x) && (r < 0))) {
    r = r - x; q = q + 1;
  } else if (((r != 0) && !SAME_SIGN(r, w)
                       && !SAME_SIGN(r, x)) ||
     ((r == -x) && (r < 0))) {
    r = r + x; q = q - 1;
  }
```

**図4** 通常の引きっぱなし法による除算

を置いて指導すべきかを検討するために, 2000年度の研究室の卒業研究のテーマとしてTinyCを拡張し, システムプログラミングに使えるようにするという卒業研究テーマを設定し, 南が着手した.

最初に鈴木から渡されたのは, TinyCをSPIMでの実行向けにmips.Cを変更したものと, BCPLの本[11]であった. BCPLを参考にしてシステムプログラミングに必要な最低限の拡張を考えてもらう以外は, 実験で学生に渡すものと同じ条件となっている. そして, 楯岡がTinyCの機能を横目に見ながらANSI Cで記述したプロトコルスタックのプロトタイプが渡された. これらを参考にし, 鈴木と議論しながら追加する仕様を決めた.

拡張は主に次の5点であった: (1)2項演算子(&, %, |, <<, >>)の追加; (2)オブジェクトタイプの追加(char)とポインタの導入; (3)定数リテラルの拡充(16進定数, 文字定数); (4)関数のパラメタのローカルスコープ化; (5)ローカル変数を設ける. ポインタは宣言と型付きのものとして実現された. (鈴木はBCPL程度の機能で十分であると考えていた.) 論文[10]では, それぞれの拡張作業がどの程度の難しさを有し, どのような指導を行えば良いかを議論している.

結果的に, 上記(1)と(3)の実施は比較的容易でコンパイラ全体を概観できる内容であったに対し, (4)と(5)を限られた実験時間内(3時間×10回)で遂行するのは難しいということが判った. また, (2)はBCPL式のものなら比較的容易な作業であることが判った. 逆に言えば, 扱える型を増やす作業やポインタを型として扱えるようにする作業は, 変数宣言の処理から変数参照の処理に至るまで, コンパイラの多くの個所を変更する必要が生じる. また, 標準的な学部3年生程度の知識や経験では, 良い参考書があってもシステムプログラミングに必要な機能の洗い出し作業を短期間に

```
  int r, ww;// 64bitレジスタを使わない
  int q = 0;
  // rとwwの連接で64bitレジスタを構成
  r = w >> 31; // 符号拡張
  ww = w;
  for (32回繰り返す) {
    if (SAME_SIGN(r, x)) {
      r = r << 1;  // この3ステップで
      if (ww < 0)  // r:ww レジスタを
        r = r + 1; // 1bit左ビットシフト
      ww = ww << 1;//
      r = r - x;   // 上だけ引けば十分
      q = (q << 1) + 1;
    } else {
      r = r << 1;  // <上と同様>
      if (ww < 0)  // このifは分岐を
        r = r + 1; // 使わなくても
      ww = ww << 1;// 実装できる
      r = r + x;
      q = (q << 1) - 1;
    }}
// 商と剰余の補正
  if (r == 0) goto end;
  if (r >= 0) goto adj1;
  if (w < 0) goto adj2;
  if (x >= 0) goto adjm;
  adjp: r = r - x; q = q + 1; goto end;
  adjm: r = r + x; q = q - 1; goto end;
  adj1: if (w >= 0) goto end;
        if (x < 0) goto adjm;
        goto adjp;
  adj2: if (r == x) goto adjp;
        if (r == -x) goto adjm;
  end:
```

**図5** 改良型引きっぱなし法による除算

行うことは難しかった. 5節の仕様は, これらを踏まえて決定された.

## 5. ベースライン TinyC

オリジナルのTinyCは単純なプログラミングに必要な最低限の機能しか持たないため, そのままでは2.2節に示したプロトコルスタックの実現が困難であるので, 言語仕様の拡張が必要である. 4節の結果を参考にしながら, 実験の準備段階で妥当な仕様の設定を行い, それを基準に学生間の話し合いで仕様を調整させることにした. その際, どれだけオリジナルの処理系を変更しないで, 要求にマッチするようにできるかという点と, ANSI Cに無い機能を付け加えないという点を考慮した.

16進定数の追加は字句解析器の拡張, 論理演算(`&`, `|`, `<<`, `>>`)の追加は式の翻訳の部分の拡張ということになり, それぞれ学生への課題として適当であるので必修課題とした.

ローカル変数を必修とするか否かについて少なからず議論されたが, プロトコルスタックで再帰するような個所がないので, 関数名と本来つけるべきローカル変数名をつなげた名前をグローバル変数名として使うようにした. (南はコンパイラがこれを内部で自動的に行うことも提案していた[10].) また, 関数呼び出しの引数はスタックに確保されるので, ダミーの仮引数を宣言しておき, それをローカル変数代わりに使うとリエントラントな関数になる.

バイトデータの問題とポインタの問題には, 実験の準備段階で議論が繰り返された.

バイトデータを扱えるようにする作業を必修課題に盛り込むのは, 実験時間的に難しいので諦めた. 鈴木はプロトコルスタックで取り扱うパケットを32bitに詰め合わせることを提案したが, 楯岡のコードの見通しとポータビリティが悪化するという意見と, メモリが十分(2MB)あるという事実から, 32bitデータの下8bitだけを使うことにした.(反リソコン的!)

プロトコルスタックでは不定長のデータを取り扱うので, バッファ中のパケットの先頭の位置ををデータとして扱わねばならない. また, 配列を途中から関数に渡したり, 一つの関数で複数のバッファが取り扱える必要がある. ポインタを型として扱えるようにするのは, やはり南が指摘したように, 実験時間的に難しいので諦めざるを得ない.

議論を重ねた結果, オブジェクトのアドレスを得る単項演算子"`&`"のみを追加することにした. 関数呼び出しで配列の途中の領域を渡すには, "`putbuf(&buffer[j])`"のように記述する. また合法的ではないが, "`j = &buffer[k]`"のようにして, アドレスを整数として記憶しておくこともできる. 配列中のパケットの先頭位置は インデクスで管理することにした.

オリジナルのTinyCに以上の機能を加えたものをベースラインTinyCと定義する.

### 5.1 MinIPSシミュレータ

学生がコンパイラの開発とMinIPSシステムの開発とプロトコルスタックの開発を並行して行うために, MIPSのシミュレータであるSPIM[6]を用いて, MinIPSシステムのシミュレータを用意した. SPIMは, アセンブラのソースを入力とし, R2000の命令セットや, 割り込み, シリアルコンソールをシミュレートする. CNP実験向けにSPIMに対して施した改造は, 主に次の3項目である: (1) メモリマップをMinIPSシステム向けに変更し, 分岐命令やメモリ参照命令の振る舞いをMinIPSにあわせた; (2)SPIM起動時に読み込まれ, `main()`を呼び出す`trap.handler`に, 符号付乗除算のサブルーチンと, Ethernetパケット送受信ルーチンを組み込んだ; (3) アセンブラ・リンカーの代用とするため, ローディング直後のメモリイメージをファイルとして出力する機能を追加した.

## 6. 学生の反応

履修した学生全員に対して行ったアンケート結果は, 論文[2),3)]で報告されている. そこでは, 全員がこの実験を「非常に楽しめた」, あるいは「楽しめた」と回答していた.

C実験の学生からは計17編のレポートを受理しており, その全てが必修課題をクリアし, その中の12名が自主課題を実施していた. またC実験に限れば, アンケートでほとんどの学生が実験時間は「充分だった」, あるいは「余った」と答えていた. ここではC実験の学生の感想の中の一部を紹介し, 実験の運営方法を振り返る. レポートには次のような感想が書かれていた.

- 作業内容がプラモデル的だった.
- 個別実験ではテストが不十分で見つからなかったバグが, N実験の学生に使わせたら見つかった.
- N実験では, ローカル変数の有無よりも, 16進数が使えるほうが可読性を増す. 実現の難しさと, その効果はあまり関係がない.
- どうせC++を使っているならオブジェクト指向の機能をフルに活用すべきである.
- 中間管理職として板挟みになったのは辛かったが, プロジェクトを鳥瞰できたのが良かった.

- 最初「本当にこんなこと出来るのか?」と思ったが, 先生もミスするのを見て気楽にできた.
- ソースを読むのに強くなった. しかしCNP相互の関連性が掴み難かった.
- 自分ができないと統合実験が失敗すると思うとプレッシャーを感じずにはいられなかった.

これらより, システムを作ったという「達成感」を「楽しみながら」体験するという実験の狙いがあらまし達成されているといえる. 特に,「実際に使われてバグが発見された.」という報告や,「プレッシャーを感じた.」という感想は, チームワークの何たるかを示す良い例で, 従来のスタイルの実験では得られない経験であろう. しかしC実験だけに限って言えば,「プラモデル的」という感想が示しているように, 実験の成功率を狙いすぎて必修課題を簡単にし過ぎたきらいがある. (これによりN実験の担当者に少なからず負担を与えているものと思われる.) しかし, 3割の学生が必修課題だけで精一杯なので, 闇雲に必修課題のレベルを上げると, ゴール到達率が下がる可能性がある. 初年度は手薄であったが, 自主課題向けの指導を充実させるべきだろう.

## 7. 終わりに

本報告では, プロセッサ, コンパイラ, コンピュータネットワークを包括的に扱う学生実験「CNP実験」のプロトタイプの開発について, コンパイラ実験の部分を中心に報告した. 当初 我々に許された「3時間×12回」の実験時間を制約であると感じたが, 要所を押さえながら学生をゴールに向かって積極的にリードしていく課題作りによって, コンパクトだがインパクトが強い実験になったと考えている.

アセンブラで乗除算ルーチンを書いたり, コンパイラの目的向けの最低限の機能を議論したりと, 実験のプロトタイプを作る作業している間に思ったのは, 先達が初期のコンピュータの開発を行っていた場面でも, このようなシーンがあったのではないかということである. 実は,20年以上前に中学の物理部の散らかった部室で,半田ごてやオシロスコープや命令コード表を片手にデバッグしていた時の何とも言えない感覚が, 鈴木に蘇ったことを告白する. この実験を通して, 学生がこういったものを少しでも味わうことができることを期待している. そして多少言い過ぎかもしれないが, 準備する側が楽しめない実験は, 学生も楽しめるわけがないと言えるのではないか.

最後に, 実験装置の準備を担当された奈良岡雅人技官と実験の実施を支えてくださったティーチングアシスタントの諸氏, それに未成熟な実験に挑戦し, 実験の改良に貢献した学生諸君に心から感謝する.

## 参 考 文 献

1) 東京大学理学部情報科学科,“プロセッサ実験,” http://www.is.s.u-tokyo.ac.jp/~vu/jugyo/processor/
2) T. Tateoka, M. Suzuki, K. Kono, Y. Maeda, K. Abe, “An integrated laboratory for computer architecture and networking, ” In Proceedings of WCAE2002, Workshop on Computer Architecture Education, Anchorage, AK, May 26, 2002. pp.110-117. http://www4.ncsu.edu/~efg/wcae2002.html
3) 前田洋一, 楯岡孝道, 鈴木貢, 阿部公輝, “MinIPS コンピュータシステムによるプロセッサ/コンパイラ/ネットワーク統合実験, ” 電子情報通信学会論文誌 D-I, vol.J85-D-1, no.10, Oct.2002.(掲載予定)
4) 鈴木貢, 河野健二, 楯岡孝道, 前田洋一, 阿部公輝, 渡邊坦, “計算機システム統合実験, ” 第4回 組込みシステム技術に関するサマーワークショップ (SWEST4) 予稿集, pp.121-128, 2002-7.
5) 葛毅, 大菅大吉, 鶴田三敏, 阿部公輝, “32ビットRISCプロセッサMinIPSの設計と実装, ” 電気通信大学紀要, vol.10, no.2, pp.71-78, 1997.
6) D. A. Patterson, J. L. Hennessy, “Computer Organization & Design: The Hardware/Software Interface, Second Edition, ” Morgan Kaufmann Publishers, Inc, 1998. (邦訳: 成田 光彰, “コンピュータの構成と設計 第2版(上・下巻), ” 日経BP社, 1999.)
7) Altera Co., “System-on-a-Programmable-Chip Development Board User Guide, ” September 2001.
8) 渡辺坦, “コンパイラの仕組み, ” 朝倉書店, 1998.
9) T. Watanabe, K. Sakuma, H. Arai, K. Umetani “Essential Language el($\alpha$) - A Reduced Expression Set Language for System Programming, ” SIGPLAN, vol.26, no.1, pp.85-98, Jan.1991.
10) 南宣正,“システムプログラミング用言語に関する研究,” 電気通信大学電気通信学部情報工学科 卒業論文, 2001-1.
11) M.Richards, C.Whitby-Strevens (和田英一訳), “BCPL-その言語とコンパイラ-, ” 共立出版 (1985).

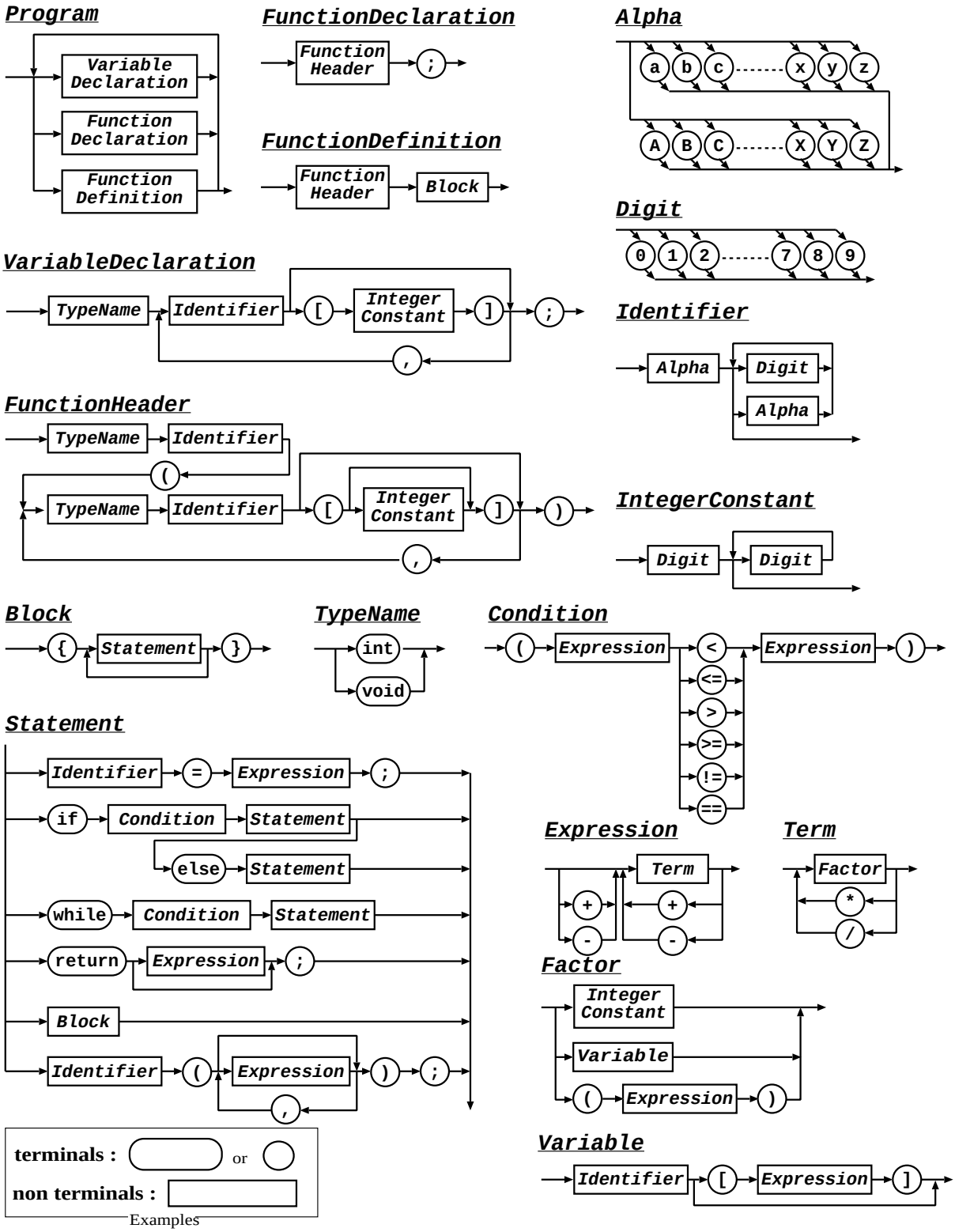


**図6** TinyC の構文図