情報工学科におけるプログラミング実習の一例†

島崎真昭†† 林　恒俊††† 北澤茂良††
古谷俊二†† 渡辺正子†† 渡辺勝正††††

本報告では情報工学科において専門教育として行われているプログラミング実習について述べている．実習は半年を1学期として2学期，1年間で行われる．ただし，1学期はさらにハードウェア実習と半期ずつに分割される．

第1学期は，アセンブリ言語と PL/I を対象とし，各受講者は最初に，各言語で，各人1題ずつ基礎課題を解く．ついで3～5人のグループでアセンブラおよび PL/I を結合して使用する課題を解かなければならない．例えば多重精度演算でアルゴリズムを PL/I，演算をアセンブラで記述する．第2学期では，システム・プログラムの構造的理解を目的として，グループ単位で言語プロセッサまたはモニタ・プログラムを作成する．言語プロセッサ・コースは PL/I でリカーシブ・ディセント法を使用して，パスカル風ミニ言語 KPL の処理系を作成する．翻訳された目的コード・プログラムは仮想スタック機械によって実行される．モニタ・コースでは仮想ミニ・コンピュータ上で動作する単一ジョブ・ストリームのモニタ・プログラムを作成する．受講生はシミュレータによって割込み処理を含むすべてのモニタ・プログラムを PL/I で作成する．ただし，ジョブは仮想計算機の機械語によって与えられる．

1. はじめに

電子計算機のプログラミングに関しては，議論の対象となっているいくつかの重要な問題がある．その中で，プログラミング教育をどのように行うか，とくに，情報工学の専門家養成のためのプログラミング教育はいかにあるべきかは，最も基本的なものである．それは単にソフトウェアだけにかかわる事柄ではなく，ハードウェアの特徴を取り入れた，計算機システムとしての把握ができるように，入門者を導くという役割をも担っている．

そのためには，“与えられた問題をいかに解くか”，“それをどのようにプログラミングするか”を習得させるだけでなく，対象とする計算機の特徴を理解して，“計算機に依存する部分と独立な部分”を分析し，“与えられた問題にはどのような計算機が望ましいか”を考えさせることが必要である．もちろん，これは「プログラミング実習」という1つの実験科目だけが負うことではなく，学科全体のカリキュラム，とくに，関連する講義との深いつながりの中で考えられる問題である．京都大学工学部情報工学科では，これらの前提の上に立って，実験およびプログラミング実習を行っている．その内容は年度によって少しずつ変化してはいるが，その方針はつぎのように定められている．

- アセンブリ言語と高水準言語 (PL/I) を使う．
- 各々の言語の入門的な問題の他に，システム・プログラムの基本となる応用問題を与える．
- 応用問題は2つの言語を使い分けて，計算機の特徴を活かすようにする．
- 3人ないし5人のグループ作業を行う．

本報告では，プログラミング実習について，昭和52年度に3年生の学生*に対して実施した内容を紹介し，その結果を報告する．使用した計算機は HITAC 8350 である．前期，後期 15 週の内，1人の学生に与えられた実習の時間と問題は，つぎのようである．

1）前期（情報工学実験および演習第一の中で），

a．180分を単位として，4週間に6単位時間．
　アセンブリ言語と PL/I の解説を行う．
　各言語の基本的な問題を1題ずつ，1人で解く．

b．180分単位　×3回／週　×3週．
　アセンブリ言語と PL/I を用いて，応用問題1題を，3人のグループで解く．

2）後期（情報工学実験および演習第二の中で），

† A Laboratory Course on Programming in Department of Information Science by MASAAKI SHIMASAKI, TSUNETOSHI HAYASHI (Kobe University of Commerce), SHIGEYOSHI KITAZAWA, SHUNJI FURUTANI, MASAKO WATANABE (Department of Information Science, Kyoto University), and KATSUMASA WATANABE (Department of Information Science, Fukui University).

†† 京都大学工学部情報工学科
††† 現在，神戸商科大学
†††† 現在，福井大学工学部情報工学科

* 本学科では2年生に対し「情報工学序説」があり，個別の情報関係の専門教育の講義は3年生，4年生に対し行われる．4年生は卒業研究として「特別研究」を行う．

c．180分単位 ×3回／週 ×7週，
言語プロセッサ，または，モニタ・プログラムを，3〜4人のグループで解く．

この内，a, b については第2章，c については第3章と第4章で紹介する．第5章では実施結果と問題点について報告する．

2. 基本問題と応用プログラム

まずはじめに，HITAC 8350 のオペレーティング・システムの概略を説明し，その中におけるアセンブラの役割とアセンブリ言語の解説を行う．それに基づいて，1人が1題ずつ与えられた問題について演習をする．問題は，

- データを入力する（カードリーダまたはコンソールより），
- データ形式の変換を行う（内部形式—外部形式），
- コンソールを通して操作指令を与える，
- 結果をラインプリンタに出力する．

といった計算機入出力操作を中心にしたものが与えられ，アルゴリズムを考えるという性格のものではない．

この演習結果は中途のまま，次いで，PL/I の解説を実例を中心に行う．PL/I の演習には，

- アルゴリズムを工夫する，
- 配列または構造体を取り扱う，

という基準で問題が与えられる．このとき，GOTO 文はできるだけ使用しないようにして，プログラムを分割し，わかり易い構造になるようにすること，欄揃え，名札のつけ方等の記述法に配慮するように指示している．4週間の間，空いた時間には適宜計算機の使用を許して，各自2つの問題を完成させるようにした．

応用問題では，グループで，仕事を解析，分担し，統合することを学ぶ．与えられる問題は，システム・プログラムの基礎となるもので，計算機の特徴を理解すると共に，道具としてのソフトウェアを開発することを目指している．

- $e, \pi, \sqrt{n}$（nは整数）の100桁以上の計算と結果のプリント，
- モデル計算機のアセンブラの作成，
- モデル計算機のシミュレータの作成，
- 会話型診断プログラムの作成，
- プログラムの編集・プリントルーチンの作成，

これらの問題では，グループでプログラムを分担できるように計画をたてるほか，PL/I とアセンブリ言語でそれぞれどの部分を書いたら良いかを判断することが含まれている．たとえば，多倍長の計算では，多倍長数と単語長数の乗算／除算ルーチンをアセンブリ言語のサブプログラムとして作成し，計算の制御を PL/I で書くことが考えられる．

問題によっては，時間的にかなり負担の大きいものがあるが，まとまった大きなプログラムを短期間に協力して完成させて大きな自信を得ているようである．

3. 言語プロセッサの設計および製作

3.1 実習の目的

実習では言語プロセッサ作成に関する種々のアルゴリズムを部分的に教える方針はとらず，限定された言語を対象とするが，システムとして完成させることを第一義として，言語プロセッサ作成過程の全体像を把握させることを目的としている．言語仕様の詳細は次節に述べるが，極めて限定されたアルゴル型算法言語を処理系作成の対象とした．これは講義との対応がよいこと，近年 PL/I や PASCAL の重要性が増していることを考慮したことによる．

実習で作成される処理系は PL/I で総計 1,500〜2,000 ステートメントの規模のもので，これをグループの学生が分担，協同して作成する．この規模のプログラムの作成とくにその分担作成は，学部3年次の学生にとっては初めての経験であるので，プログラム作成にあたっては，モジュール性を高め，分担者間のインタフェースを明確にし，組織的にプログラムを作成させることも実習の大きな目標となっている．

3.2 言語仕様とその特徴

言語プロセッサ作成の対象となるアルゴル型算法言語[1]の設計の基本方針を箇条書にまとめるとつぎのとおりである．

（1） 1記号先読みで確定的に下降型構文解析可能な言語とする．

（2） 1パスコンパイルを可能とするため，手続き名，関数名を含め，識別子名は宣言してから使用することとする．PASCAL[2] における **forword** 宣言は導入しないので，相互に参照し合う手続き，関数は除外される．

（3） ブロック構造，名前の有効範囲については PASCAL と同じ考え方を採用する．

（4） 手続き，関数の再帰的引用を許す．パラメータについては，call by value および call by reference の2種の方式をとり，call by name の方式は採用しない．

```
<program>           =PROGRAM <identifier>; <block>.
<block>             =<constdcl><typedcl><variabledcl>
                      {<procfuncdcl>}
                      BEGIN <statement>{; <statement>}
                      END
<constdcl>          =<empty>|CONST <constdef>
                      {<constdef>}
<constdef>          =<identifier>=<constant>;
<typedcl>           =<empty>|TYPE <typedef>
                      {<typedef>}
<typedef>           =<identifier>=<type>;
<variabledcl>       =<empty>|VAR <vardef>{<vardef>}
<vardef>            =<identifier>: <type>;
<procfuncdcl>       =<procdcl>|<funcdcl>
<procdcl>           =PROCEDURE <identifier>; <block>;
                      |PROCEDURE <identifier>
                      <paramlist>; <block>;
<funcdcl>           =FUNCTION <identifier>:
                      <basictype>; <block>;
                      |FUNCTION <identifier><paramlist>:
                      <basictype>; <block>;
<statement>         =<assignst>|<proccall>|<groupst>|
                      <ifst>|<whilest>|<forst>
                      |<empty>
<assignst>          =<lefthandside>:=<expression>
<lefthandside>      =<variable>|<functionidentifier>
<functionidentifier>=<identifier>
<proccall>          =CALL <procedureidentifier>
                      |CALL <procedureidentifier>
                      <actualparameter>
<procedureidentifier>=<identifier>
<groupst>           =BEGIN <statement>{; <statement>}
                      END
<ifst>              =IF <condition>
                      THEN <statement><elsepart>
                      |IF <condition> THEN <statement>
<elsepart>          =ELSE <statement>
<whilest>           =WHILE <condition> DO <statement>
<forst>             =FOR <variableidentifier>
                      :=<expression> TO <expression>
                      DO <statement>
<variableidentifier>=<identifier>
<empty>             =
<variable>          =<variableidentifier>{<modifierpart>}
<modifierpart>      =(.<expression>.)
<factor>            =<unsignedconstant>|<variable>
                      |<functionidentifier>
                      |<functionidentifier>
                      <actualparameter>|(<expression>)
<actualparameter>   =(<expression>{,<expression>})
<term>              =<factor>{<opfactor>}
<opfactor>          =<multiplyingoperator><factor>
<multiplyingoperator>=*|/
<expression>        ={<unaryoperator>}¦<term>
                      {<opterm>}
<opterm>            =<addoperator><term>
<addoperator>       =+|-
<unaryoperator>     =+|-
<condition>         =<expression><relationaloperator>
                      <expression>
<relationaloperator>==|<|>|¬=|<=|>=
<paramlist>         =(<paramspec>{; <paramspec>})
<paramspec>         =<param>: <basictype>
<param>             =<valueparam>|<variableparam>
<valueparam>        =<identifier>
<variableparam>     =VAR <identifier>
<type>              =<typeidentifier>|<basictype>
                      |<arraydef>
<typeidentifier>    =<identifier>
<basictype>         =INTEGER|CHAR
<arraydef>          =ARRAY (.<unsignedinteger>.)
                      OF <type>
<unsignedconstant>  =<constantidentifier>
                      |<unsignedinteger>|'<character>'
<constant>          =<constantidentifier>
                      |<unaryoperator>
                      <constantidentifier>
                      |<unsignedinteger>
                      |<unaryoperator><unsignedinteger>
                      |'<character>'
<constantidentifier>=<identifier>
<unsignedinteger>   =<digit>{<digit>}
<identifier>        =<letter>{<letterordigit>}
<letterordigit>     =<letter>|<digit>
<digit>             =0|1|2|3|4|5|6|7|8|9
<letter>            =A|B|……|Z|‡|-|¥
<character>         =<digit>|<letter>
                      |+|-|*|/|(|,|.|)|:|;|>|=|<
                      |_|¬
```

入出力のための標準手続き READC, READI, WRITEC, WRITEI, WRITELN が用意されている．それらの手続きは

```
PROCEDURE READC (VAR C: CHAR);
PROCEDURE READI (VAR I: INTEGER);
PROCEDURE WRITEC (C: CHAR);
PROCEDURE WRITEI (I: INTEGER);
PROCEDURE WRITELN;
```

と宣言されているとみなす．これらは1文字および整数の入出力用であり，WRITELN は出力において改行することを意味する．

図 1 KPL の構文の BNF による定義
Fig. 1 BNF of KPL.

（5） 構造をもつ文として，**if** 文，**while** 文，**for** 文を導入する．**if** 文について，いわゆる 'dangling else' の解釈は PL/I, PASCAL と同じ考え方をとる．

（6） 定数，タイプの定義を可能とする．ただし簡単化のため，基本データタイプは整数型，文字型のみとし，データ構造は配列に限定する．

以上の方針に従い設計された実習用言語 KPL の構文則を**図 1** に示す．PASCAL-S[3] は本来プログラミング入門教育のために設計されているが，これを簡単化すれば，言語プロセッサ設計の入門教育に適すると考えられ，KPL の言語設計に PASCAL-S を参考にした．KPL の一応の設計後参考文献[4]が入手され，その中で定義されている PL/0 とも比較された．KPL は PL/0 と比較して，ⅰ）単純化されているが，タイプの概念があり，ⅱ）関数があり，ⅲ）手続き，関数に引数が許され，引数受渡しの処理方式の教育ができ，

iv) **for** 文がある．そしてこれらは言語プロセッサ設計入門教育用として望ましいものと考えられた．構文図は教育に適しており，BNF との変換等，構文図の扱いを Wirth[4]に準拠して，実習として教育している．KPL の構文図の1部を**図2**に示す．

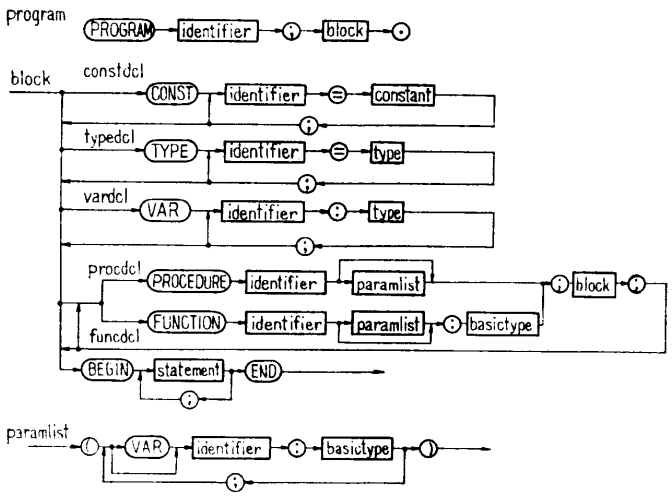


図 2 KPL の構文の構文図 (Syntax Diagram) による定義（部分）
Fig. 2 A part of syntax diagram of KPL.

3.3 意味解析およびコード生成

意味解析，コード生成部においては，記憶域の割当，レジスタの管理，入出力を含めて OS とのインタフェースが重要な問題であるが，学部3年生に対する実習の複雑さおよび実習時間を考慮して，仮想的なスタック計算機を設定し，目的コードを生成する方針を採用した．したがって生成された目的コードは仮想スタック計算機のシミュレータ（目的コードのインタプリタ・プログラム）により実行され，結果が得られる．仮想スタック計算機の命令セット設計の方針はつぎのとおりである．

（1） アドレス指定にはブロックレベル，ディスプレースメントによる方式を用いる．スタティックリンクをたどる機能を仮想計算機にもたせる．

（2） 手続き，関数の呼び出し，復帰のための命令語を設ける．

（3） 入出力処理簡単化のため，整数および1文字入出力の命令を用意する．

（4） (2), (3) 以外は特殊な機能をもつ命令語の導入は避ける．PASCAL-S[3] においては高級言語のステートメント処理を意図した形の命令も導入されているが，現実の計算機との距離をできるだけ拡大しないため，プリミティブな命令に限定する方針をとる．

以上の方針で設計された仮想スタック計算機の仕様を**表1**に示す．

3.4 言語プロセッサの作成

処理系は機能的には，単語解析部，構文解析部，コード生成部，仮想スタック計算機シミュレータ（目的コードインタプリタ）からなり，PL/I を用い，recursive descent 法により作成させている．実習はつぎの過程を踏んでいる．

（1） factorial 関数を含む程度の簡単な例題プログラムとその目的コードを説明し，机上シミュレーションを行わせて，仮想スタック計算機の理解を深めさせる．

（2） 例題プログラムを作成させ，翻訳用内部テーブルの作成を含めて机上コンパイルを行わせ，翻訳過程の理解を深めさせる．

（3） 作業分担を相談し，プログラムを分担作成し，単体テストを経て，結合し，システムを完成させる．

52年度において5グループが実習し，全グループが一応システムを完成し，グループによっては，ハノイの塔やクイックソート等種々のプログラムの実行を行った．作業分担は，recursive descent 法を用いていることもあって，構文解析と意味処理，コード生成を分割せず，単語解析，仮想計算機のシミュレータ作成，算術式処理，ブロック管理およびステートメント処理に分割している場合が多い．各種テーブルのデータ構造，テーブル探索の手法，エラー処理は簡単なものを用いる指導方針をとっているが，グループによっては多少の工夫を行っている．

言語の仕様の面からは，実数型，レコード型，ポインタ型が導入されず，データ構造が単純化されていること，処理系作成の面からは，仮想スタック計算機を用いてコード生成を単純化したこと，テーブル類の構成および探索法を簡単にしたことなど，実用のコンパイラより大幅な簡単化が行われているが，学部3年生のシステム・プログラムに関する実習として一応所期の目的を達していると考えられる．なお実習では扱えなかった言語プロセッサ作成に関する諸事項はシステム・プログラムに関する講義の守備範囲となっている．

KPL は言語としては単純なものであるが，言語プロセッサ設計の入門教育のために，現実に教育現場で

表 1 言語プロセッサ・コースの仮想スタック計算機の機械命令
Table 1 Machine instruction format.

ニモニック	命令語	命令語の内容 [は (.,] は .) を表わす.
LA	Load Address	t:=t+1; s[t]:=base(p)+q;
LV	Load Value	t:=t+1; s[t]:=s[base(p)+q];
LC	Load Constant	t:=t+1; s[t]:=q;
LI	Load Indirect	s[t]:=s[s[t]];
INT	Increment T	t:=t+q;
DCT	Decrement T	t:=t−q;
J	Jump	pc:=q;
FJ	False Jump	if s[t]=0 then pc:=q; t:=t−1;
HL	Halt	Halt
ST	Store	s[s[t−1]]:=s[t]; t:=t−2;
CALL	Call	s[t+2]:=b; s[t+3]:=pc; s[t+4]:=base(p); b:=t+1; pc:=q;
EP	Exit Procedure	t:=b−1; pc:=s[b+2]; b:=s[b+1];
EF	Exit Function	t:=b; pc:=s[b+2]; b:=s[b+1];
RC	Read Character	read one character into s[s[t]]; t:=t−1;
RI	Read Integer	read integer to s[s[t]]; t:=t−1;
WRC	Write Character	write one character from s[t]; t:=t−1;
WRI	Write Integer	write integer from s[t]; t:=t−1;
WLN	New Line	CR & LF
AD	Add	t:=t−1; s[t]:=s[t]+s[t+1];
SB	Subtract	t:=t−1; s[t]:=s[t]−s[t+1];
ML	Multiply	t:=t−1; s[t]:=s[t]*s[t+1];
DV	Divide	t:=t−1; s[t]:=s[t]/s[t+1];
NEG	Negative	s[t]:=−s[t];
CV	Copy Top of Stack	s[t+1]:=s[t]; t:=t+1;
EQ	Equal	t:=t−1; if s[t]=s[t+1] then s[t]:=1 else s[t]:=0;
NE	Not Equal	t:=t−1; if s[t]¬=s[t+1] then s[t]:=1 else s[t]:=0;
GT	Greater Than	t:=t−1; if s[t]>s[t+1] then s[t]:=1 else s[t]:=0;
LT	Less Than	t:=t−1; if s[t]<s[t+1] then s[t]:=1 else s[t]:=0;
GE	Greater or Equal	t:=t−1; if s[t]>=s[t+1] then s[t]:=1 else s[t]:=0;
LE	Less or Equal	t:=t−1; if s[t]<s=[t+1] then s[t]:=1 else s[t]:=0;

使用可能であることが確認された点に意義があろう.

4. モニタ・プログラム

4.1 モニタ・プログラムの実験の問題点

言語処理プロセッサの実験・演習においては，ほぼ確立した枠組が考えられるのに対して，モニタ・プログラム，一般的には，オペレーティング・システム，OS，に関する限り現時点では 2,3 の試みを除いて，標準となる枠組は考えられない．これは主としてつぎのような原因によるものと考えられる.

(1) 概念の不統一性・多様性など OS 自身が内包する難しさ.

例えば OS/MVS, VM/370, Multics のように大規模・複雑なシステムから，Solo や RC 4000 のシステムのように設計論的にすぐれているものまで多様な OS が存在して，その設計・構造も様々である（各種 OS 教科書，文献 5)〜8) 参照).

(2) 実験を実施する上での問題点.

OS の実験では，特にその核部分では，計算機を裸の状態で使用しなければならない．そのためには既存の OS を止めるか，仮想機械を利用すればよいが，その場合，機械語レベルのプログラミングが要求される．Concurrent-PASCAL 等の高水準言語を利用すれば，この問題は起らないが，ハードウェアに密着した割込み処理などの核部分の実験が行えない.

これらの問題を考慮に入れて，モニタ・プログラム実験の課題について以下に述べるような方針を決定した.

4.2 実験の方針

(1) モニタ・プログラムの作成

小規模なモニタ・プログラムを作成させ，実際に運転させる.

(2) モニタ・プログラムの範囲

モニタ・プログラムを作成させる場合でも，その範囲は，(a)割込み処理等の核部分を含む；(b)並列処理以上のレベルで実験する；(c)ジョブ・キューイングやファイル・システムなども含む；などのレベルが考えられる．(b),(c) ではプログラミングの手段として高水準言語が利用できる点では有利であるが，もっとも非構造的で，かつ多重プログラミングを実現す

るモニタ・プログラムの重要な要素である核部分の理解が不十分になる．一方(c)ではジョブ・スタッキング，ファイル構造，アクセス法等に大規模なプログラムを必要とし，学部3年生レベルの実験の枠内に収まらない．

これらの点から，実験で作成するモニタには核部分および簡単な入出力機能とジョブ・スケジューラからなるモデルを採用した．OS には性能，プロテクション等の重要な事項もあるが，ここでは OS の構造の理解を優先的に考え，これらは今回の実習にはとり入れていない．

（3） テスト・ジョブ

作成されたモニタ・プログラムの下で運転されるジョブはあらかじめ，アセンブルされて機械語になっているプログラムのみとする．したがって通常のアセンブル／リンク／実行というジョブ形式は採用しない．

4.3 簡単なモニタ・プログラムの仕様

実験で作成されるモニタ・プログラム（以後 SOS =(Simple OS) と略す）の仕様は以下のとおりである．

（1） 外部仕様

- シングル・ジョブ・ストリーム
- ジョブ・プログラムは機械語（ローダ語）で与える．
- ジョブ制御文およびオペレータ・コマンドは**図3**のように与える（詳細は文献 1)）．
- モニタ・マクロ命令はプロセス間通信機能を中心として定義する．

モニタ・プログラムとして正しい動作の確認を実習の主目的とし，性能等に関する項目は割愛した．

（2） 内部仕様

内部構造および各モジュール間の相互作用を明確に定義するため，SOS は，シングル・ジョブ・ストリームにもかかわらず，複数個の並列プロセスから構成される．ここでプロセスの並列動作は多重プログラミング技法により実現する．SOS は以下に示される要素から構成される．

- 核部分
  割込み処理，プロセス・スケジュール，モニタ手続き呼び出し制御を行う．
- プロセス
  ジョブ・プロセス——ジョブ制御文，ジョブ・プログラムの実行
  オペレータ・プロセス——オペレータ・コマンドの処理
  カード・プロセス——カード入力処理
  プリンタ・プロセス——プリンタ出力処理
  タイプライタ・プロセス——タイプライタ入出力処理
- モニタ手続きの集まりは上記プロセス間の情報交換・同期を行う．その方式は文献 5) で定義された Send message, Wait message, Send answer, Wait answer の体系を採用する．

プロセスおよびプロセス間の相互作用 (interaction) は**図4**に示される．OS の動作の理解を実習の主目的とし，プロセスおよび手続き間のプロテクションの問題は考えないことにした．

4.4 実験の方法

SOS は前述の理由により，HITAC 8350 上に実現された仮想機械上に作成される．さらに高水準言語 (PL/I) による作成を可能にするため，仮想機械についてつぎに述べるような仕様を採用した．

- CPU は2つの状態 P1, P2 をもつ．
- P1 状態では割り込み可能で，16 ビット／語の仮想機械の機械語のユーザ・ジョブ・プログラムを実行する．その機械語を**図5**に示す．
- P2 状態は割込み禁止で，PL/I で作成されたモニ

```
$$JOB ␣ job name          LOAD
$$RUN                     HALT
      job program in      KILL
      object code format  CONT
$*                        RPLY ␣ process name ␣ parameter
$$DATA                    EXIT
      data                DISP
$*
$$END
```

(i) job sample　　(ii) operator command

図 3 SOS の外部仕様
Fig. 3 External specification of SOS.

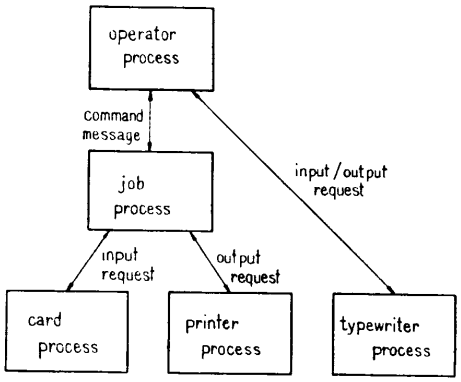
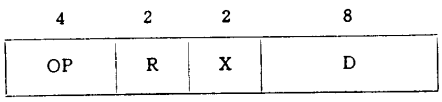


図 4 プロセスおよびその相互作用
Fig. 4 Processes and interaction.



OP: operation code; R: register; X: index register;
D: displacement; effective address= $\begin{cases} D+(X) & X \neq 0 \\ D & X=0 \end{cases}$

mnemonic	meaning
NOP	NO oPeration
L	Load
ST	STore
A	Add
S	Subtract
AND	AND
XOR	eXclusive OR
C	Compare
CL	Compare Logical
SFT	ShiFT
SVC	SuperVisor Call
BAL	Branch And Link
BC	Branch Condition

図 5 モニタ・コースの仮想計算機の機械命令
Fig. 5 Machine word of virtual machine.

タ・プログラムを実行する.

- CPU は割込み状態および入出力割込み状態を示すレジスタ ISR, DSR をもつ. これらのレジスタには割込み原因などが適宜表示される.
- CPU 状態を表示する PSW, 各レジスタ, 主記憶等は構造体または配列として宣言され, PL/I から参照可能である.
- P2 状態においてプログラム割込み命令 (SVC) に対応する PL/I 手続き 'MC', 入出力開始命令に対応する手続き 'IO' が用意され, SOS のプログラムで使用可能である.

PSW, レジスタ類および機械語は, あらかじめこの実験のために作成された実験サポート・システムによってシミュレートされる.

モニタ・プログラムの作成と実行はつぎのように行われる.

- 各要素 (核, 各プロセス, モニタ手続き) のプログラムの大略を, PASCAL に近い言語で記述したものを提示し, 動作, 機能の理解を計る.
- 各部分の完全なプログラムを PL/I で作成させる.
- 実験サポート・システムをオブジェクト・モジュールの一つとして, SOS プログラムと結合する.
- 結合された実験サポート・システムを実行すると実験モニタ・プログラムが運転される.
- SOS のオペレータ・コマンドを与えてユーザ・ジョブを実行する. 簡単なジョブ (たとえばカード入力データのプリンタへの複写) が実行できれば成功である.

以上の方法によると, モニタ・プログラムのすべてが核部分も含めて PL/I によって記述できる. ただし, SOS の多重プログラミングは, 実験サポート・システムを含めるとコ・ルーチンとして実現されているため, MC 命令を呼びだすブロックのレベルはすべて同じでなければならない. したがってモニタ・プログラム中に手続きブロックが使用できず, プログラムの構造化がさまたげられるのがこの方法の欠点である.

なお, SOS のもとではしるジョブ・プログラムを作成するために仮想機械の機械語のアセンブラが用意されている.

4.5 実施結果

52 年度においては前半グループの場合, モニタのモデル理解のために, 実験に割りあてられた時間 (7 週, 69 時間) の大部分が費され, 作成したモニタのテストはすべて追加実験の形で約 134 時間をかけて行われた. 後半グループでは関連する講義も進んでいることと, 前半グループの経験もあってスムーズに実施され, 50 時間程度の追加実験で終了した. 以上で述べたモデルと指導方法が現実の OS をどの程度反映し, 理解に有効かどうか多少疑問が残るが, OS に関するプログラミング実習としては, 適当な試みであると思われる.

5. おわりに

プログラミングの実習には, やさしい問題から順に段階的に与えて慣れさせる方法と, 大きな問題の中で手続きを分割して適用できるアルゴリズムを発見させる方法とが考えられる. 当学科では, 多くの議論の末後者の方針を取ることになったが, 実施の結果を振り返っていくつかの問題点を列記し, 今後の検討の素材とする.

(1) 短期間にアセンブリ言語と PL/I を同時に習得することには無理がある. 多くの者はどちらか一方にかたよる傾向にあり, 両方使いこなすまでにはゆかない. 中には, 両方共未消化のまま前期を終える者もあった. そのため, 少し時期をずらして 1 つずつ完全に理解させた方がよい. 昭和 53 年度 3 年生より, この点改められ, 2 年後期に PL/I の実習, 3 年前期にアセンブリ言語の実習を行っている.

(2) プログラミング言語の解説, 基本的なアルゴリズムの紹介等は, 講義と実習が入り組んだ形で進め

られるのが良い．また，言語そのものよりも，JCLとOS との関係を理解させることの方が難しい．

（3） 言語プロセッサ，モニタの作成に先立って，講義によってその全体像を把握させておくことが必要である．

（4） 言語処理プログラム，モニタ・プログラムの実習を計画する際種々の議論が行われたが，現在は仮想計算機方式を利用している．その仕様の設計方針および実習項目に関連した問題点はそれぞれの項でふれたが，全体としてつぎのことが指摘できる．この方式では，一台の計算機を利用しながら，モニタ・プログラムの実習を含めて，複数グループの実習を OS のもとに効率的に実施できる．かつデバッグにシステム・ソフトウェアのサポートを有効に利用できる．仮想計算機と現実の計算機とのアーキテクチャ上の差異が場合によっては，実習者に実用的ソフトウェア製作上の問題点を看過ないし，過小評価させる可能性を指摘できるが，仮想計算機方式では，優先実習項目とプログラム作成の負担を勘案して，仕様を設計でき，実習項目の把握を容易にすることができる．なお実習で扱えなかったシステム・プログラムに関する上級の項目，言語処理プログラムとモニタとのインタフェースに関する事項は本学科では講義で扱われている．

（5） デバッグのしやすいプログラムを作るように指導することも必要であるが，机上デバッグの仕方，デバッギングツールの効果的な使用法を習得させることが必要である．

多くのことをさせようとしたために，正規の実習時間のほぼ2倍の時間が費された．この点，問題の与え方，指導の仕方に大いに反省すべき点があるが，反面，役に立ちそうなプログラムを協力して完成させたことによる自信と喜びを身につけたようである．なお，前期（a と b）のために，41 人で約 32,200 枚（平均 786 枚／人）のカードを費した．また，ジョブ件数は 4,247 件である（その内，ランをしたものは 3,437 件であった）．

以上，システム・プログラムの基礎となる言語プロセッサないしはモニタの作成を最終段階とした1年間のプログラミング実習について，それらのモデルを中心に述べた．本報告に対する皆様のご批判をいただいて，今後の改善の糧としたい．なお，本報告がプログラミング実習の計画をたてられる際の一助になれば幸いである．

最後に，プログラミング実習の計画および実施にあたり，京都大学工学部情報工学科の全教官が協力したことを記して感謝の意にかえる．

参 考 文 献

1) 情報工学実験及演習，京都大学情報工学教室 (1977).
2) Jensen, K. and Wirth, N.: PASCAL User Manual and Report, Lecture Notes in Computer Science 18, Springer-Verlag (1976).
3) Wirth, N.: PASCAL-S: A Subset and its Implementation, Berichte des Instituts für Informatik ETH, No. 12 (1975).
4) Wirth, N.: Algorithms+Data Structures=Programs, Prentice-Hall (1976).
5) Brinch Hansen, P.: Operating Systems Principle, Prentice-Hall (1973).
6) Madnick, S. E. and Donovan, J. J.: Operating Systems, McGraw-Hill (1974).
7) Coffman, E. G. Jr. and Denning, P. E.: Operating Systems Theory, Prentice-Hall (1973).
8) Tsichritzis, D. C. and Bernstein, P. A.: Operating Systems, Academic Press (1974).

（昭和54年 2 月 16 日受付）
（昭和54年 9 月 20 日採録）