Compiler Generating System

藤　野　喜　一*

1. Introduction

特定のディジタル計算機のためにコンパイラ (compiler または Compiling system 略して CS) を作るには多くの労力とかなりの時日 (5～30 man years) を必要とする．一般に用いられてきた方法は，対象とする計算機の機械語（アセンブラ言語も含む）で CS の Implementation を記述するもので，機械（マシンコード）の特性を十分に利用できるから，作成された CS はきわめてオプティマルになる長所があるが，時間，労力その他多くの困難を伴う．したがってできるだけ容易にかつ短期間で高い機能をもつ CS を作る方式の研究が望まれてきている．この考え方を実現するプログラムを Compiler Generating System (略して CGS) とよぶ．この論文はその基本的な考察と実現の方式について述べたもので，「情報処理学会，第6回プログラミング・シンポジウム」で発表したものをかき改めたものである．

なおこの研究に当たり種々有益な御指導と御助言を戴いた早大理工学部野口広教授，実験のため多くの計算機時間を割りあてて戴いた早大電子計算室長難波正人教授に厚く感謝します．コーデングには片貝祐康氏に多大の協力をうけたことを感謝します．

2. コンパイラの表現

2.1. CS の表現

コンパイラの性格を明確にするために以下の記号を用いて表現する．自動プログラム言語（略して APL）の一つを $L(\mathrm{i})$ とかき（i は APL の種類を示すインデックスで $\mathrm{i}\in \mathrm{I}=\{1, 2, \cdots, l\}$），その集合を $\{L\}=\{L(\mathrm{i})|\mathrm{i}\in\mathrm{I}\}$ とかく．ALGOL, FORTRAN および NUMERIC〔文献 1), 2) 参照〕などは $L(\mathrm{i})$ の例である．また計算機の一つを $M(\mathrm{i})$ で，その機械語を $\tilde{L}(\mathrm{i})$ で示せば，それらの集合は $\mathrm{J}=\{1, 2, \cdots, m\}$ として $\{M\}=\{M(\mathrm{i})|\mathrm{i}\in\mathrm{J}\}$, $\{\tilde{L}\}=\{\tilde{L}(\mathrm{i})|\mathrm{i}\in\mathrm{J}\}$ とかける．ここで $M(\mathrm{i})$ に対する機械語は $\tilde{L}(\mathrm{i})$ 一つだけとする．このとき CS は $CS(\mathrm{x}, \mathrm{y}, \mathrm{z})$ と表現される．ただし x は CS のソースプログラムの記述言語が $L(\mathrm{x})$, y は CS の対象とする機械が $M(\mathrm{y})$，すなわち CS の Implementation の記述言語が $\tilde{L}(\mathrm{y})$, z は CS のオブジェクトプログラムの記述言語が $\tilde{L}(\mathrm{z})$ であることをそれぞれ意味している．

通常のコンパイラでは $\mathrm{y}=\mathrm{z}$ であるから $CS(\mathrm{x}, \mathrm{y}, \mathrm{y})$ となるので，特に $CS(\mathrm{x}, \mathrm{y})$ とかくこともできる．

2.2. CS の機能

APL $L(\mathrm{i})$ が $CS(\mathrm{i}, \mathrm{j}, \mathrm{k})$ の入力言語であるとき，$M(\mathrm{j})$ によってその表現に影響をうけるからこの意味で $L(\mathrm{i})$ を $L(\mathrm{i}, \mathrm{j})$ とかく．

あるプログラム P を $L(\mathrm{i}, \mathrm{j})$ でかいたものを $P(\mathrm{i}, \mathrm{j})$ と表わせば，$CS(\mathrm{i}, \mathrm{j}, \mathrm{k})$ は $P(\mathrm{i}, \mathrm{j})$ を object program $\tilde{P}(\mathrm{i}, \mathrm{k})$ に変換する．$\tilde{P}(\mathrm{i}, \mathrm{k})$ は $\tilde{L}(\mathrm{k})$ がその記述言語で，source 言語が $L(\mathrm{i})$ であることを示す．

$CS(\mathrm{i}, \mathrm{j}, \mathrm{k})$ の機能を $\mathrm{f}(\mathrm{i}, \mathrm{j}, \mathrm{k})$ とすれば次の関係が成りたつ．

$$f(\mathrm{i}, \mathrm{j}, \mathrm{k})P(\mathrm{i}, \mathrm{j})=\tilde{P}(\mathrm{i}, \mathrm{k})\in P(\cdot\mathrm{k})$$

$$(P(\mathrm{i}, \mathrm{j})\rightarrow \boxed{CS(\mathrm{i}, \mathrm{j}, \mathrm{k})} \rightarrow \tilde{P}(\mathrm{i}, \mathrm{k}))$$

ただし $P(\cdot\mathrm{k})$ は $\tilde{L}(\mathrm{k})$ でかかれたプログラムの集合である．

いま一つの問題 P を $L(\mathrm{i}, \mathrm{j})$, $L(\mathrm{i}', \mathrm{j})$ で記述して，それぞれ $P(\mathrm{i}, \mathrm{j})$, $P(\mathrm{i}', \mathrm{j})$ とし，これらが数学的には同一の問題ということを $P(\mathrm{i}, \mathrm{j})\equiv P(\mathrm{i}', \mathrm{j})$ で表わせば，$P(\mathrm{i}, \mathrm{j})$ と $P(\mathrm{i}', \mathrm{j})$ はプログラムの構造が一般に異なるから $P(\mathrm{i}, \mathrm{j})$ から $P(\mathrm{i}', \mathrm{j})$ への変換があるはずである．(P の数学的なフローチャートを仲立ちにして考察すればよい)．これを $\varphi(\mathrm{i}, \mathrm{i}', \mathrm{j})$ とかき，$CS(\mathrm{i}, \mathrm{j}, \mathrm{j})$, $CS(\mathrm{i}', \mathrm{j}, \mathrm{j})$ の機能を $f(\mathrm{i}, \mathrm{j}, \mathrm{j})$, $f(\mathrm{i}', \mathrm{j}, \mathrm{j})$ とすれば

$$f(\mathrm{i}, \mathrm{j}, \mathrm{j})P(\mathrm{i}, \mathrm{j})=\tilde{P}(\mathrm{i}, \mathrm{j})\in P(\cdot\mathrm{j})$$

$$f(\mathrm{i}', \mathrm{j}, \mathrm{j})P(\mathrm{i}', \mathrm{j})=\tilde{P}(\mathrm{i}', \mathrm{j})\in P(\cdot\mathrm{j})$$

となり，$\tilde{P}(\mathrm{i}, \mathrm{j})\equiv\tilde{P}(\mathrm{i}', \mathrm{j})$ となるはずである．よって $\tilde{P}(\mathrm{i}, \mathrm{j})$ から $\tilde{P}(\mathrm{i}', \mathrm{j})$ への変換を $\sigma(\mathrm{i}, \mathrm{i}', \mathrm{j})$ とすれば，

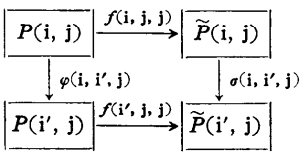
$$\sigma(\mathrm{i}, \mathrm{i}', \mathrm{j})f(\mathrm{i}, \mathrm{j}, \mathrm{j})P(\mathrm{i}, \mathrm{j})=\sigma(\mathrm{i}, \mathrm{i}', \mathrm{j})\tilde{P}(\mathrm{i}, \mathrm{j})$$
$$=\tilde{P}(\mathrm{i}', \mathrm{j})$$

* 早稲田大学

となり，これから次の関係が得られる．

$$\sigma(i, i', j)f(i, j, j)=f(i', j, j)\varphi(i, i', j) \quad *$$

図にかけば，



$\varphi(i, i', j)$: $P(i, j)\to P(i', j)$ および $\sigma(i, i', j)$: $\tilde{P}(i, j)\to\tilde{P}(i', j)$ を仮定すると，二つの $CS(i, j, j)$ と $CS(i', j, j)$ の機能 $f(i, j, j)$ と $f(i', j, j)$ は上の * を満足しなければならない．この関係は，任意の機械のプログラミング・システム作成の際の基本的な仮定である．特に $\varphi(i, i', j)$ を language Translation, $\sigma(i, i', j)$ を Machine Language Translation といい，これらを実現する Program を Language Translator, Machine language translator といい，$\Psi(i, i')$ $\Sigma(i, i')$ とかくことにする．Ψ, Σ は translation の実際的な証明といえる．

2.3. Self expressible な CS

一つの $CS(i, j, j)$ を仮定するとき，$CS(i, j, j)$ 自身の Implementation をその入力言語 $L(i, j)$ でかいたものを $P(CS(i, j, j)/L(i, j))$ とし，これを $CS(i, j, j)$ にソース・プログラムとして与えるとき，オブジェクトプログラム $CS^*(i, j, j)=\tilde{P}(CS(i, j, j)/L(i, j))$ が $CS(i, j, j)$ と同一の機能をもつ CS の Implementation であれば，$CS(i, j, j)$ は self expressible（自己表現可能）であるという．コンパイラは一般に必ずしもこの性質をもたないが，$M(j)$ の特性と記憶容量の制限を無視すれば（広義の）自己表現可能と考えられる．ある $CS(i, j, j)$ が（広義でない）自己表現可能となる一つの十分条件は $L(i, j)$ が $\tilde{L}(j)$ を部分集合として含むことである．

CGS にとってこの性質は重要な手がかりを与える．

3. Compiler Generating System (CGS)

3.1. CGS の記号的表現

CGS は記号を用いて

$CGS(x, y, z|U, j)$

と表わす．U は CGS の入力言語が $L(U)$, j は CGS の Implementation の記述言語が $\tilde{L}(j)$（すなわち，base の計算機が $M(j)$）であること，x, y, z は CGS がコンパイラ $CS(x, y, z)$ を作成することを示す．

CGS の入力データは (i) $CS(x, y, z)$ の一般的な表現を $L(U)$ でかいたもの．(ii) $x=A$, $y=B$, $z=C$ とするとき，必要な情報の集合 $a(A, B, C)$ である．

CGS の出力データは (i) $CS(A, B, C)$, (ii) $CS(A, B, C)$ の $\tilde{L}(B)$ に関する words 数，その他の情報である．

CGS にとっては，$L(U)$, $M(j)$ は一般に固定されるから $CGS(x, y, z)$ と表わしてもよい．また $CS(x, y, z)\epsilon\tilde{P}(\cdot, y)$ であるから，見方をかえると，$CGS(x, y, z|U, j)$ は入力言語 $L(U)$, base 計算機 $M(j)$, 出力言語 $\tilde{L}(y)$ のコンパイラ $CS(U, j, y)$ とみることもできる．この性質より CGS は CS として機能の他に出力言語 $\tilde{L}(y)$ に関係する部分が変更可能であることが必要である．また x, y, z で定まる情報は $L(U)$ でかかれる Source Program の中で定数ではなく変数として与える．

以上の考察から，$CGS(x, y, z)$ を実現する問題には，$CS(x, y, z)$ の構造をはっきりさせることが必要である．

4. コンパイラの構成

4.1. APL の構成要素

APL $L(x)$ は次の3種の要素 **Carrier** Program が対象とするデータを表現するものをいい，その集合を C で示す．**Subroutine** 特定の名前をもち，C の二つの部分集合を α, β とするとき，α を β に map する手順を表わすものを subroutine といい，その集合を Z で示す．**Programming Word** 上記の二つの要素の他にプログラムをかくために必要であるものをいい，その集合を W で表わす．

4.2. $CS(x, y, y)$ の構成要素

次の七つからなる．

(1) $L(x, y)$ の文法，(2) $L(x, y)$ の構成要素，(3) $\tilde{L}(Y)$ の文法，(4) $\tilde{L}(Y)$ の構成要素，(5) $\tilde{L}(Y)$ の loading program，(6) $L(x, y)$ の Library，(7) $\tilde{L}(Y)$ の Library

4.3. $L(x, y)$ のプログラム

$L(x, y)$ でかいたある計算の手順を示すフローチャートを $s=(X, S)$ とすれば s は一つのグラフとなる．ただし $X=\bigcup_{i=1}^{n} b_i$, $S=\bigcup_{i=1}^{n} s_i$ で b_i は block, s_i は b_i と任意の b_j との関係を表わす switch といい，X, S をそれぞれの集合という．

各 b_i には block の座標（ラベル）をつけ，次の五

つの成分 (component) よりなる. (1) declaration 成分 d_i: b_i 内の carrier の性質を規定する. (2) operation 成分 op_i: b_i 内の計算を表わす部分, (3) Machine 成分 mI_i: b_i 内の機械コードでかかれた部分, (4) Substitution 成分 sb_i: subroutine calling を示す部分. (5) Switch 成分 s_i: (上述).

各成分は $L(x, y)$ の構成要素がその文法に従って配列された relation である element の列である. b_i の成分に属する element を $s_{i,j}$ とかけば $b_i=(s_{i,1}\cdots s_{i,n(i)})$ となり, $p=\{b_i\}_i=\{\{s_{i,j}\}_j\}_i$ を s のソースプログラムという. ただし p は, block b_i が他の b_j をあるいは element $s_{i,j}$ が他の $s_{i,k}$ を declare する関係にあるときは b_i は b_j より, あるいは $s_{i,j}$ は $s_{i,k}$ より前にあるという条件を充している. これが充される限り p における b_i の順序は任意である.

p の前に program declaration (program の性質を規定する) を, 後に program end mark (%) をつける. 前者には次の三つがある.

PRG NAME INITIAL-ADDRESS

REG NAME (Formal parameter list)

TSB NAME (Formal parameter list)

PRG は s を直ちに実行可能な program に作成して INITIAL ADDRESS から格納できるようにし, REG は s を formal parameter をもち, NAME を Library ($L(x, y)$ の) に登録され, $\tilde{L}(y)$ の Library に格納された subroutine にし, TSB は s を Library に登録されない一時的な subroutine として作成する.

4.4. $L(x, y)$ の構造的性質

$L(x, y)$ のもつ $L(x)$ の operator (Arithmetic および logical) の集合および $L(x, y)$ が含む $\tilde{L}(y)$ の命令コードの集合を合わせたものを $L(x, y)$ の 0 位の subroutine の集合といい, $\mathbf{Z}^0$ で表わす. また $\mathbf{Z}^0$ を $L(x, y)$ の 0-位の Library といい LB(0) とかく. 当然これは各 $L(x, y)$ ごとに固有のものである. 1 位の program とは LB(0) の subroutine のみを使用した program である. (たとえば LB(0) でかいた平方根の program は 1 位である.) 1 位の program に名前をつけて subroutine として Library に登録 (registration) するとき 1 位の subroutine といい, その集合を $\mathbf{Z}^1$ とかく. ALGOL, FORTRAN などにおける sin, cos, $\sqrt{\ }$ などの基本関数は $\mathbf{Z}^1$ に含まれる.

$\mathbf{Z}^0\cup\mathbf{Z}^1$ を LB(1) とかき 1 位の Library という. このようにして (r−1) 位の Library が LB(r−1) $=\bigcup_{k=0}^{r-1}\mathbf{Z}^k$ とかけるとき, LB(r−1) の上にかかれた (LB(r−1) の subroutine を使用した) program を r 位の program という. 特に $\mathbf{Z}^{r-1}$ の subroutine を含むとき proper な r 位の program といい, これを subroutine として登録したものを r 位の subroutine といい, その集合を $\mathbf{Z}^r$ とかく. 二次方程式の根を求める subroutine は $\sqrt{\ }\in\mathbf{Z}^1$ を使用しているので 2 位である.

block の各成分は次のようなエレメントから作られる. ただし $z\subset$LB(r−1), $c\subset\mathbf{C}$, $w\subset\mathbf{W}$ とする.

DC element は $e^0=(\phi, c, w)$, すなわち subroutine を含まない. たとえば array A, B(100, 20);

OP element は $e=(z, c, w)$, $z\subset$LB(1) たとえば A+B → C; sin(X+Y)+sqt(X**2+Y**2) → T; ここで $\langle +, **\rangle\subset\mathbf{Z}^0$, $\langle$sin, sqt(=$\sqrt{\ }$)$\rangle\subset\mathbf{Z}^1$ である.

MI element は $e=(z, c, w)$, $z\subset$LB(0) で z は機械語コードである. たとえば ADD(x); ADD$\in z$

SB element は $e=(z, c, w)$, $z\in\mathbf{Z}^k(1\leqq k\leqq r-1)\in$LB(r−1) で e はサブルーチンコーリングとなる. たとえば MATRIXINVERS (X, Y); MATRIXINVERS$\in$LB(1)

SW element は $e=(z, c, w)$ で $z\in$LB(1) たとえば α: (sin (X) gtq cos (Y)) → β: else GOTO γ:; ここで $\langle$sin, cos$\rangle\in\mathbf{Z}^1$, $\langle$gtq(≧)$\rangle\in\mathbf{Z}^0$

4.5. Main Part と Reduction

一般に APL $L(x, y)$ で program をかくことは, その時までに作成された Library LB(r−1) の subroutine を用いてかくことである. この意味で r 位の program を s=(X, S)/on LB(r−1) とかき, これを s の main part といい, [s] で表わすことにする.

s が含む subroutine の中で $\mathbf{Z}^0$ に属さぬものを s の第 1 代の subroutine (directly sub.) とよびその set を D(s) とかけば D(s)=$\{S^1, S^2, \cdots, S^{r-1}\}$ となり, $S^i=\bigcup_{k=1}^{l(k)}s_k^i$, $s_k^i\in\mathbf{Z}^i$ $(1\leqq i\leqq r-1)$ である. ただし S^i は少なくとも一つは ϕ (空) でないとする.

s の main part [s]=(X, S)/on LB(r−1) の全ての directly sub.'s をその main part でおきかえることを program s の reduction という. すなわ

ち D(s) の i 位の sub. $s_k^i(\in Z^i)$ を LB(i−1) の上の program で表現したもの i, e s_k^i/on LB(i−1) でおきかえる. したがって D(s) の sub. が高々 i 位 ($1 \leqq i \leqq r-1$) であれば $D(\rho(s))$ の sub. は高々 (i−1) 位となるから s は高々 (r−1) 回の reduction により LB(0) の上の program にできる. $\rho(s)$ は s を 1 回 reduction したものを表わす.

例 いま program α の mainpart を $\alpha=(\beta_1, \beta_2, \beta_3, \alpha^\circ, C_\alpha)$ とする. ただし $D(\alpha)=\{\beta_1, \beta_2, \beta_3\} \subset$ LB (r−1) で C_α は program α が使用する carrier の集合 i,e. $C_\alpha \subset C$ また $\alpha^\circ \subset Z^\circ$ とする.

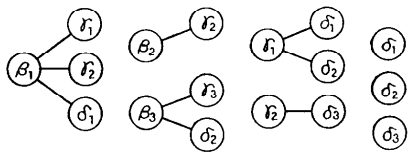
このとき, α に reduction を行なって LB(0) に移す手順は次のようになる. ただし LB(r−1) から次の情報, すなわち第 1 代 sub. の情報として

$\beta_1=(\gamma_1, \gamma_2, \delta_1, C_{\beta_1})$, $\beta_2=(\gamma_2; C_{\beta_2})$, $\beta_3=(\gamma_3, \delta_2, C_{\beta_3})$

$\gamma_1=(\delta_1, \delta_2, C_{\gamma_1})$, $\gamma_2=(\delta_3, C_{\gamma_2})$, $\gamma_3=(\Phi, C_{\gamma_3})$

$\delta_1=(\Phi, C_{\delta_1})$, $\delta_2=(\Phi, C_{\delta_2})$, $\delta_3=(\Phi, C_{\delta_3})$

すなわち



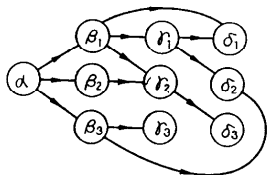
が得られるとする. このとき

$\rho(\alpha)=[\alpha][\beta_1][\beta_2][\beta_3]$

$\rho^2(\alpha)=[\alpha][\beta_1][\beta_2][\beta_3][\gamma_1][\gamma_2][\delta_1][\gamma_3][\delta_2]$

$\rho^3(\alpha)=[\alpha][\beta_1][\beta_2][[\beta_3][\gamma_1][\gamma_2][\delta_1][\gamma_3][\delta_2][\delta_3]$

となる. これをグラフにかけば下図のようになる.



なお 5.3. を参照.

5. $CS(x, y)$ の機能

CS は一般に INTRODUCTION ANALYSERS ALLOCATION CONTROLDIVISION の四つの部分およびこれらに使用される TABLES より成りたつ.

5.1. INTRODUCTION

program $s=\{s_{ij}\}$ の構成要素は $L(x, y)$ の文法に合う限り表現は自由である. ただし, $L(x, y)$ に固有な形をもつ operater, delimiter は除く. s が使用した各要素を $CS(x, y)$ が規定する標準コードに変換し, 分類して CS の標準集合 (CSS) に対応させることにより, 全ての構成要素を一定の Rule で処理できる.

Compiler standard set (C.S.S) は

(1) Process Mark, 成分 mark の set, (2) type declaration のための symbols の set, (3) 文字, 数字の set, (4) $L(x, y)$ が含む operator の set, (5) $L(x, y)$ が含む subroutine の set, (6) variables の set, (7) constants の set, (8) program names の set, (9) coordinate の set の 9 個の部分集合よりなる. (1)〜(4) は $L(x, y)$ ごとに一定であるが Machine $M(y)$ によって多少変更をうける. (5)〜(8) は CS が作用する program ごとに登録されて作成される. 特に (6)〜(8) は適当な指示がなければ一回ごとに cancel される.

5.2. Declaration Analyser (DC)

DC element s_{ij} で定義される carrier の type についての情報を作成し, carrier に与える. またその他の成分における carrier の登録と search を行なう.

一般に carrier は (1) 所属する subset の分類, (2) type, (3) 所属する subset における登録番号, (4) その subset における相対的な initial address, (5) word length, (6) array の場合の構造の情報の格納番地; から構成される. したがって carrier は compiler standard set においては上記の六つの情報を成分とする一つのベクトルと考えることができる.

相対 Address の決定には, 単長には 1, 倍長および複素数数値の carrier には 2 によって計算される. 登録番号の counter には s (symbolic variable) f (formal variable) c (constant number), initial address の counter には s_1(symbolic variable), s_2(array variable), f(formal variable), c_1(constant number) がある.

5.3. 変換の Part

この Part は次の四つに分かれる.

Operation Analyser (OPA), Machine Instruction Analyser (MIA), Substitution Analyser (S

BA), Switching Analyser (SWA).

INTRO および DCA で構成要素が標準コードに変えられた program s を $\bar{s}$ とかくと $\bar{s}=\{\bar{s}_{1j}\}=\{(INT\cdot DC)(s_{1j})\}=(INT\cdot DC)(s)$ となる.

変換の手順 $\bar{s}=\{\bar{s}_{1j}\}$ の element $\bar{s}_{1j}$ が, 所属する成分に対応する Analyser により, 次の段階を経て変換される.

(1) $\bar{s}_{1j}$ を変換可能な relation $\bar{s}_{1jk}$ に reduce する.

(2) $\bar{s}_{1jk}$ に必要な情報を作成する.

(3) $\bar{s}_{1jk}$ に対応する $\tilde{L}(y)$ でかかれた object relation $\bar{s}_{i,j,k}$/on $\widehat{L\bar{B}}/(r-1)$ を作成する.

$ds_{1,j,k}$, c_{1jk} をそれぞれ $\bar{s}_{1jk}$ にを含まれる第1代のサブルーチンの set および carrier の set とすれば $\bar{s}_{1jk}=(d\bar{s}_{1jk}, \tilde{c}_{1jk})$ とかけるから, このとき $\bar{s}_{1jk}$ は未定係数として

(i) $\tilde{c}_{1jk}$ に対する subset の parameter sy, cs, sa, wm, f (これらの内容は各 subset の initial address を示す)

(ii) $d\bar{s}_{1jk}$ に対応する parameter を使用した subroutine の set UST に記入される). 構造上のサブルーチンの parameter (complex, index, double precision etc) も含む.

の2種類を含んでいる.

(4) $\bar{s}_{1jk}$ の Allocation の未定係数として

$\tilde{a}_{1jk}=ALC(PNOP(\bar{s}_{1jk})-BNW(\tilde{b}_1))$ 〔i〕: op Z をつけて Tape に格納または punch out する. ここで NOP は CS が作り出す $\tilde{L}(Y)$ の命令の順序を示す数で, PNOP は $\bar{s}_{1jk}$ の先頭命令の NOP, BNW は $\tilde{b}_1$ の先頭命令の NOP で, 〔i〕は object の operation part $\widetilde{OP}=\bigcup_{i=1}^{n}\tilde{b}_1$ の先頭番地 op からの $\tilde{b}_1$ の相対的アドレスである.

(5) PM=REG のとき, formal variable を使用した命令の $\bar{s}_{1jk}$ 内の相対順序を FT(Formal variable address table) に記入する. (FT について Allocator 参照)

以上の (1)~(5) によって $s=\{s_{1j}\}$ は列 $\{\bar{s}_{1jk}\cup\tilde{a}_{1jk}\}$ に変換され, $\bar{s}_{1j}=\sum_k(\bar{s}_{1jk}\cup\tilde{a}_{1jk})$, $\tilde{b}_1=\sum_j\bar{s}_{1j}$, $\widetilde{OP}=\sum_i\tilde{b}_1$ となる.

5.4. ALLOCATION PART

5.1~5.3 で作成された列 $\{\bar{s}_{1jk}\cup\tilde{a}_{1jk}\}$ の各 $\tilde{a}_{1jk}$ および未定の parameter の set の内容の決定に必要な計算を行ない確定した係数にかえるための情報 (ALI) を作る. REGISTRATER, ALLOCATOR, REDUCTER の三つの part よりなる.

(1) REGISTRATER PM=REG のとき s を subroutine 化する機能 $\widetilde{RG}$ を $\widetilde{OP}$ に付加する.

(i) $\widetilde{OP}$ に使用された Formal variable とそれに与えられる Actual variable との結合のための Actual address reception $\widetilde{RG}1$ (pr) と Formal address modification $\widetilde{RG}2$ (se) より成り, これらはおもに FT を使って作られる.

(ii) Index Register, その他の Register または variable を必要に応じて待避し復活するための $\widetilde{RG}3$ (rs) $\widetilde{RG}4$ (rr) (それぞれ Register set, Register reset)

(iii) Main part との link operation $\widetilde{RG}5$ (l_0) PM=PRG のときは $\widetilde{RG}$ は付加しない.

(2) ALLOCATOR 次の機能を行なう.

(i) sup 以外の parameter の確定. pr, se, rs, op, rr, lo, sba, cs, sy, wm, sa (これらを $p_1, p_2, \cdots, p_{11}$ で表わす) をこの順で pr=0 として決定する. このとき $\widetilde{OP}$ の words 数は NOP-1 である. ただし NOP の値は op-part 変換終了時の値である. $\widetilde{RG}=\phi$ のときは pr=se=rs=op, rr=lo=sba である.

(ii) $\tilde{b}_1$ の相対アドレス 〔i〕 の確定 〔1〕=0, 〔i〕=〔i−1〕+(b_{i-1} の words 数) ($1\leqq i\leqq n$) とすると

$\tilde{a}_{1jk}$ の内容=op+〔i〕+(PNOP($\bar{s}_{1jk}$) −BNN($\tilde{b}_1$))

(iii) CST (constant の変換された標準形の subset) の打ち出しまたは Tape への格納

(iv) $\tilde{b}_1$ の Allocation の情報の打ち出しまたは格納.

ALI i 〔i〕 ($1\leqq i\leqq n$)

(v) usp 以外の parameter θ_j の Allocation の情報.

ALI wds(θ_{j-1}) (+) θ_j ($1\leqq j\leqq 11$) ただし wds(θ_0)=0

wds(θ_j) は θ_j で表現される set の word 数

(vi) PM=REG ならば

ALI 0000: s の parameter の打ち出しまたは格納と subroutine として s の Library への登録と s の Library $\widehat{LB}(r-1)$ への登録のための情報として

(a) s の第一代の subroutine の名前の set

(b) s の NAME, 入出力変数の数, s の語数

を打ち出しまたは格納する．この (a), (b) の情報は，s が登録されて，他の program または subroutine の descending sub. として call されるとき，指定される任意の位置に $\widetilde{L}$(y) の Library から s の main part が取り出されて Allocation される必要から生ずる Link の情報という（Reduction 参照）．

（3） REDUCTOR PM=PRG のとき次の機能を行なう．

（i） 変換の part で作られた s̃/on $\widetilde{\text{LB}}$(r−1) に (r−1) 回以下の reduction を行ない，M(y) 上で実行可能な program すなわち $\widetilde{\text{LB}}$(0) 上の 1-program s̃/on $\widetilde{\text{LB}}$(0) にかえる．

（ii） s̃/on $\widetilde{\text{LB}}$(0) が使用する全ての s の des. sub's の main part を s̃ の与えた先頭番地から互に重ならず s̃/on $\widetilde{\text{LB}}$(0) が最小の語数を占有して配列されるように各 des. sub の parameter に絶対番地を与える．

(iii) des. sub's の parameter の内容の確定

s̃ の directly subroutine を s̃ の 1-st des. sub とすれば，その集合 $\text{us}^{(1)}$ は UST^0 に登録されている．$\rho^{(k)}(\tilde{s})$ が使用する subroutine の集合を $UST^{(k)}$ として，

$\text{us}^{(k+1)}=UST^{(k)}-UST^{(k-1)}$ $(k\geqq 1)$ を s̃ の $(k+1)$ th des. sub. の集合とすれば，$UST^{(k)}=UST^{(0)}+\bigcup_{j=2}^{k+1}\text{us}^{(j)}=\bigcup_{j=1}^{k+1}\text{us}^{(j)}$ となる．s̃ が r 位ならば $\text{us}^{(r)}=\phi$ であるから $\bigcup_{j=1}^{r-1}\text{us}^{(j)}=UST^{(r-2)}$ となる．

(1°) $UST^{(0)}$ すなわち $\text{us}^{(0)}$ の *ALI*

$\text{us}_j^{(0)}\in UST^{(0)}$ の先頭番地 θ_j^0 は $\text{us}_k^{(0)}$ の words を $\text{wds}(\text{us}_k^{(0)})$ として，$(\text{us}^{(0)}=\bigcup_{k=1}^{l(0)}\text{us}_k^{(0)}$ とする)

$\theta_1^0=$先頭番地 $(\tilde{s})+\text{wds}(\tilde{s})+\sum_{k=1}^{j-1}\text{wds}(\text{us}_k^{(0)})$

$(1\leqq j\leqq l(0))$

これを s̃/$\widetilde{\text{LB}}$(r−1) が含む $\text{us}_j^{(0)}$ と対応させるために

$L\ \theta_j^{(0)}\cdot\text{us}_j^{(0)}$ の parameter $(1\leqq j\leqq l(0))$

を打ち出しまたは格納する．

(2°) $\text{us}^{(k)}$ $(r-1\geqq k\geqq 1)$ の search 次のようになる．

ただし $\text{us}^{(k-1)}=\bigcup_{t=1}^{l(k-1)}\text{us}_t^{(k-1)}$ とする．

0： $2\rightarrow k$;

1： $1\rightarrow t$, $\phi\rightarrow$V(set V を clear する)

2： $\text{us}_t^{(k-1)}$ の第 1 代 sub. $\text{us}_{t,m}^{(k-1)}$ $(1\leqq m\leqq\nu(t,k-1))$ を Library からとり出して

$\text{us}_{t,m}^{(k-1)}\notin UST^{(k-2)}+\text{V}$

となる m がなければ 3: へ; あればその m を m' とし，

$\text{V}+\bigcup_{m'}\text{us}_{t,m'}^{(k-1)}\rightarrow\text{V}$

3： $t+1\rightarrow t$; $\{t\leqq l(k-1)\}\rightarrow$ 2: else $\rightarrow$ 4:

4： $\{\text{V}=\phi\}\rightarrow$ (3), else $\rightarrow$ 5:

5： $\text{V}\rightarrow\text{us}^{(k)}$; $UST^{(k-2)}+\text{us}^{(k)}\rightarrow UST^{(k-1)}$; go to 1:;

(3°) このとき s̃ の reduction は完了

(4°) えられた $UST^{(r-1)}$ が含む subroutine の中 $\text{us}_j^{(0)}$ 以外の全ての $\text{us}_j^{(i)}$ の先頭番地 θ_j^i を次のように定める．

$\theta_1^1=\theta_{l(0)}^1+\text{wds}(\text{us}_{l(1)}^{(1)})$

$\theta_j^2=\theta_{j-1}^2+\text{wds}(\text{us}_{j-1}^{(1)})$

$\theta_l^i=\theta_{l(i-1)}^{i-1}+\text{wds}(\text{us}_{l(i-1)}^{(i-1)})$

$\theta_j^i=\theta_{j-1}^i+\text{wds}(\text{us}_{j-1}^{(i)})$

$\theta_{l(r-1)}^{r-1}=\theta_{l(r-1)-1}^{r-1}+\text{wds}(\text{us}_{l(r-1)-1}^{(r-1)})$

(5°) $\text{wds}(\rho^{(r-1)}(\tilde{s}))=\theta_{l(r-1)}^{r-1}+\text{wds}(\text{us}_{l(r-1)-1}^{(r-1)})-$先頭番地 $(\tilde{s})$

(6°) Descending subroutine の *ALI*

1： $1\rightarrow k$;

2： $1\rightarrow j$;

3： $\text{us}_j^{(k)}\in\text{us}^{(k)}$mainpart$[\text{us}_j^{(k)}]$ を Tape から移すための pseudo code として

LOAD θ_j^k $\text{us}_j^{(k)}$ の parameter

の打ち出しまたは格納

4： us_j^k の第 1 代 subroutine の parameter の *ALI*

$L\ \theta_{j,t}^k$ $\text{us}_{j,t}^{(k)}$ の parameter $(1\leqq t\leqq l(j,k))$ の打ち出しまたは格納

5： $j+1\rightarrow j$; $\{j\leqq l(k)\}\rightarrow$ 3:, 6:

6： $k+1\rightarrow k$; $\{k\leqq r-1\}\rightarrow$ 2:, 7:

7： 終了

(7°) L s̃ の先頭番地 s̃ の parameter の打ち出しまたは格納

(8°) Program Information の打ち出しまたは格納

名前 word 数 $(\rho^{(r-1)}(\tilde{s}))$ s̃ の parameter

これは s̃ を他の program から call されない独立な program として格納し，必要に応じて使用する場合の見出しとして用いられる．

例 4.5 の例の program α の ALI

4.5 より α の discending subroutine は

$us^{(1)}=\{\beta_1, \beta_2, \beta_3\}$, $us^{(2)}=\{\gamma_1, \gamma_2, \delta_1, \gamma_3, \delta_2\}$, $us^{(3)}=\{\delta_3\}$ である. *ALI* は次のようになる.

α wds (α) α の parameter
L 先頭番地 (α), α の parameter } program information

$L\ \theta(\delta_3)p(\delta_3)$, LOAD $p(\delta_3)$ ····· δ_3 ··· $us^{(3)}$

$L\ \theta(\delta_1)p(\delta_1)$, $L\ \theta(\delta_2)p(\delta_2)$
$L\ \theta(\gamma_1)p(\gamma_1)$, LOAD $p(\gamma_1)$ } ····· γ_1
$L\ \theta(\delta_3)p(\delta_3)$, $L\ \theta(\gamma_2)p(\gamma_2)$
LOAD $p(\gamma_2)$ } ····· γ_2
$L\ \theta(\delta_1)p(\delta_1)$, LOAD $p(\delta_1)$ ····· δ_1
$L\ \theta(\gamma_3)p(\gamma_3)$, LOAD $p(\gamma_3)$ ····· γ_3
$L\ \theta(\delta_2)p(\delta_2)$, LOAD $p(\delta_2)$ ····· δ_2
} $us^{(2)}$

$L\ \theta(\gamma_1)p(\gamma_1)$, $L\ \theta(\gamma_2)p(\gamma_2)$
$L\ \theta(\delta_1)p(\delta_1)$, $L\ \theta(\beta_1)p(\beta_1)$
LOAD $p(\beta_1)$ } ····· β_1
$L\ \theta(\gamma_2)p(\gamma_2)$, $L\ \theta(\beta_2)p(\beta_2)$
LOAD $p(\beta_2)$ } ····· β_2
$L\ \theta(\gamma_3)p(\gamma_3)$, $L\ \theta(\delta_2)p(\delta_2)$
$L\ \theta(\beta_3)p(\beta_3)$, LOAD $p(\beta_3)$ } ····· β_3
} $us^{(1)}$

$L\ \theta(\beta_1)p(\beta_1)$, $L\ \theta(\beta_2)p(\beta_2)$
$L\ \theta(\beta_3)p(\beta_3)$,
ALI for C_α
$L\ \theta(\alpha)\ p(\alpha)$
} α の main part に対する *ALI*

α の main part

5.5. CONTROL DIVISION

この DIVISION は以下にのべる compiling state (c-state と略す) の情報に基づいて *CS*(x, y) の全体を control する. c-state とは source program s を s に変換する際の CS の活動の状態を規定する情報の組をいい, ある s に対する c-state の列は $s \to \tilde{s}$ を行なうための CS の履歴である. 次の七つの情報からなる.

(1) 処理しつつある program の name (P.P.N)

p=PRG α ··· REG β ··· TSB γ ··· % ··· REG δ ··· % % ··· % ならば CS がある時点で処理している program の name をいい. この列, たとえば $\alpha, \beta, \gamma, \delta, \beta, \alpha$ を PPN の列という.

(2) program level (PRL)

s がただ一組の PM と END をもつとき, s の element s_{ij} の PRL を 1 とし, s が multilevel であるとき, すなわち複数個の PM と END を含むとき, s_{ij} をつつむ PM と END の組の個数を s_{ij} の PRL という.

(3) Process counter, Preprocess counter

CS が処理する順番に Program name を登録した番号を Process counter (PC) といい, ある Program に先行する Program name の PC を Pre Process Counter (PPC) という.

上の例の Source Program に関する上記の情報は以下のようになる.

PC	登録 Program	PPC
1	α	0
2	β	1
3	γ	2
4	δ	2

PRL	0	1	2	3	2	3	2	1	0
PC	0	1	2	3	2	4	2	1	0
PPC	0	0	1	2	1	2	1	0	0
備考		Process 開始	Process 進行中 ——→ 処理の方向						Process 終了

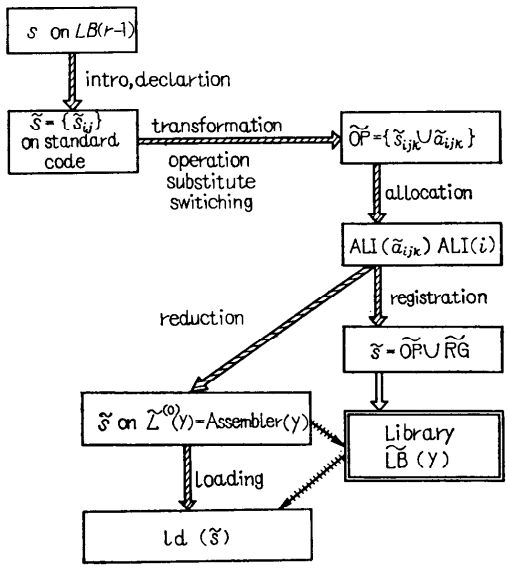
(4) SUBF

SUBF は登録された Program が Subroutine 化を要求する (PM=REG, TSB) ならば 1, しなければ 0 となる FLAG である.

(5) Coding state

Process されているある Program に関する Process end mark (%) がまだよみこまれていないとき, coding state は transformation であり, <%> がよまれたあとは Allocation であるという.

(6) Block coordinate indicator



Compiler の働き

Process をうけている program の block に関する情報を示すものである.

(7) Analyser state

ある時点において, Process される element s_{ij} を処理する Analyser を示すものを analyser state といい, DC, OP, MI, SB, SW (Transformation) ALC, REG, RED (Allocation) の8種類がある.

以上七つの情報を $p_1, p_2, \cdots, p_7$ とかき表せば

Compiling state$=(p_1, p_2, \cdots, p_7)$

である.

6. CGS の方式

6.1. 自己拡張方式

自己拡張に必要な基本的部分からなる Compiler $CS_B(i, j, j)$ を $\tilde{L}(j)$ でかき, これを基礎にして $CS(i, j, j)$ を作る方式である.

$CS_B(i, j, j) \rightarrow CS(i, j, j)$

6.2. 変換の方式

ある $M(j)$ に対して $CS(i, j, j)$ が一つあるとき, 他の $M(k)$ に対して $CS(i, j, j)$ と同等の機能をもつ $CS(i, k, k)$ を次の step で作る方式である.

$$CS(i, j, j) \xrightarrow{(i)} CS(i, j, k) \xrightarrow{(ii)} CS(i, k, k)$$

(i) の手順① $CS(i, j, j)$ の出力のコード表, すなわち, 出力言語 $\tilde{L}(j)$ の各 operation を構成する命令の group のテーブル (MIGT(j)) と, 格納されている位置を示す数値のテーブル (OSNT(j)) を $\tilde{L}(k)$ に対応する MIGT(k) と OSNT(k) に変更する.

② ①による MIGT の変更に関連する $CS(i, j, j)$ の各部分の命令のアドレス部分を変更する.

③ $\tilde{L}(j)$, $\tilde{L}(k)$ の数値表現の base が異なるときは, variable, constant の相対アドレス計算の counter, Allocation の未定係数の計算に必要な counter を $\tilde{L}(k)$ の base に改める. さもなければ $\tilde{L}(k)$ の loading Routine に必要な変換機能を与える.

(ii) の手順 ① $CS(i, j, j)$ を $L(i, j)$ でかいて $CS(i, j, k)$ に入力すると, $CS(i, k, k)$ ができる.

6.3. 混合方式 上の二つを組み合わせたもので次の手順で行なわれる.

$$CS_B(i, j, j) \xrightarrow{(i)} CS_B(i, j, k) \xrightarrow{(ii)} CS_B(i, k, k) \xrightarrow{(iii)} CS(i, k, k)$$

7. CGS

CGS は source language $L(u)$, および CGS 本体の $\tilde{L}(j)$ による Implementation よりなる.

7.1. $L(u)$

(1) Basic symbol: letter, 数字, 記号

(2) Variable: letter で始まる letter と数字の string で5文字以下である.

(3) Constant: integer type の数値で 12 桁以下.

(4) coordinate: ?で始まる5文字以内の letter と数字の string.

(5) Process Mark: PRG. REG. %

(6) Component Mark: DC, OP, SB, MI, SW, K

(7) Operator: (+, −, ×), (gtr(>) geq(≧), =, neq(≠), ¥, or, and, not

(8) 特別な operator // (word 指定オペレータで A/2 とかいて, A が2以上の word length をもつときAの第2番目の word を指定する).

lxt, rxt は extract された結果をそれぞれ register の MSD, LSD まで移す. **ind** array variable の添字による element 指定のほかに, 直接 index を指定する. たとえば E ¥ ind(2) は E の値を index(2) に代入する.

@はA@2, と用いて array A の index 2 で指定される element を示す.

7.2. Declaration

area 変数が array のとき, はじめにその elements の個数の上限とその word length を与える.

array は array variable の構造を規定し, wl は array でない variable の word length を規定する.

たとえば area A, B(100/2); array A(10.5) wl x, y, z(2)

TABLE Compiler は多くの table を使うので次のような形の declaration を行なうことができる. たとえば

TABLE SEP 10 (5/2, sp 10); とかけば, 名前が SEP 10 で5個の term をもち, 各 term が2個の element をもち, table の current depth を表わす variable が SP 10 である一つの table が定義される. declare された table の情報は table name table に格納される.

7.3. Operation Part (OP)

variable, constant, operator と (,) により作られるもので演算を表わす. たとえば

OP S+T¥C; X/3+Y/2¥Z/3;

7.4. Switch function (SW)

SW go to α_t; または SW(B_1)¥(C_1), (B_2)¥(C_2), …, (B_{n-1})¥(C_{n-1}), else¥(C_n); K の形をし, C_t は OP, SW, SB, MI の成分の列で最後に必ず SW 成分をおく. B_t は logical expression で値が true ならば (C_t) を行ない, false ならば B_{t+1} を調べ B_t が全て false ならば (C_n) を行なう.

7.5. Subroutine Calling SB

REG によって登録された program を calling する. たとえば

SB INC 1(x, y; a, b; α, β); (9. 例参照)

7.6. MACRO INSTRUCTION (MI)

作成される Compiler の効率を高める手段の一つとして, 表現しやすくかつ Machine の特性を十分もつ, 次のような MACRO INSTRUCTION を定義する.

この system では MI part で使用する全ての macro 命令を宣言して作成できるから, あらかじめ特別な機械語命令を組み込む必要はない. また場合によって OP, SW の両 part は全て MACRO で代用できる.

MACRO の declaration の方式

DC DMI XXXXX(X_1, X_2, …, X_m) (A)

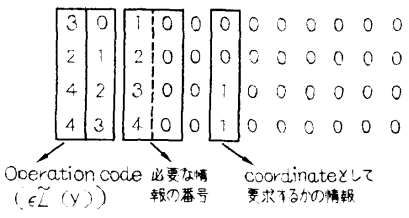
〔machine instruction (1)〕
〔machine instruction (2)〕
⋮
〔machine instruction (n)〕; (B)

(A) の部分: XXXXX は宣言したい MACRO 命令の名前で variable と同構成の string で, (X_1, …, X_m) はこの命令が必要とする情報を表わす formal variable の list で $0 \leqq m \leqq 10$ とする.

(B) の部分: 命令 XXXXX を構成する machine instruction ($\in \tilde{L}(y)$) の列で各命令は規定に従う 12 桁の定数である.

たとえば **if A=B then go to** ?α **else** ?β; (A, B は variable, ?α, ?β は coordinate) は MACRO 命令 JEQ で代用できる.

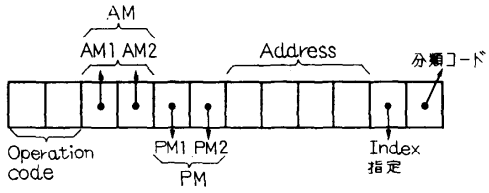
DMI JEQ(A, B, ?α, ?β)



この宣言により, この MACRO 命令を構成する機械語命令群は instruction group table (MIGT) の $\xi \sim \eta$ 番地に格納され, 次に名前 JEQ と, MIGT における格納番地 ($\xi \sim \eta$) を MACRO NAME TABLE (MNT) に記入する.

machine instruction の分類と構成

機械語命令は表現しやすくするために次の構成とする.



Operation Code は $\tilde{L}(y)$ に応じて変化する. AI part は MACRO 命令の actual parameter をよみこまれた順に格納するテーブル MAIT (1〜10) に対するこの MACRO 命令のアドレス part の要求を AM 1, index part の要求を AM 2 で表わす. たとえば (AM 1, AM 2)=(13) ならばこの instruction の Address part:=MAIT (1), INDEX part: =MAIT (3) となる.

PM part PM=(PM 1, PM 2) によりこの instruction に付加すべき parameter および Address の Print 形式を示す. PM 1=0 は情報は他から与えられる. PM 1=2 は address part を絶対番地として作る. PM 2 の内容は 1は coordinate, 2は variable, 3は constant, 5は arrary variable; 6は formal variable, 7は working memory; 9は common variable をそれぞれ表わす parameter を付加させる.

分類コードは次のように形式を分類する.

(1) 0のとき, アドレス part は要求しない.

(例) 5900 e 9000200 → 59+0002;
33003 d 002200 → 33*000 b;
120003000100 → 12*0001(cs);

(2) 1のとき, Left operand を要求する.

OP-part の expression 解析用テーブル上の Left operand の状態により変化する. たとえば

120000000001 ならば

Left operand の状態 打出し form

2000200 → 12*0002(sy);
5040110 → 12*401(sa), 1;
となる.

(3) 2のとき, Right operand を要求する. (2)と同様

(4) 3のとき, subroutine parameter を打ち出す

4201 ext 03 → 42+0001*0(ext);

(5) 4のとき, Address part のみを打ち出す

000003000504 → *0005(cs);

〔注意〕 MACRO 命令の時は全て分類コードは0であるが, AM によって MAIT から情報をとりその後は上の規則に従う.
L(u) の文法にのべたものの中で,
variable, constant, coordinate, PRG, REG DC (array, TABLE) と MACRO INSTRUCTION だけを用いて CS_B(i, j, j) を作成できる. 終りの Program の例はこの方法である.

8. 入力データ作成の注意

入力データ CS(x, y, z) を L(u) でかいたものを x=A, y=B, z=B とするとき A, B が変わるたびにかきかえなくてもよいように, 変化する部分は variable で表現する.

8.1. Table の大きさ

コンパイラは base M(B) によって当然その記憶容量などが異なるから, そのテーブルの大きさはあらかじめ決定できない. CS(A, B) の implementation ができたあとで, 決めるために, CS(A, B) の変数, 定数, formal variable, coordinate, array 変数などの個数は全て未定とする. またテーブルの各 term の内容の形式, 大きさなどは同じく変数で与える.

8.2. L(A, B) の構成要素

構成要素の表現とその内部標準コードは, base の M(B) の入出力装置, word の構造などにより影響をうける. よってこれらの内容とコードは変数とする. また coordinate, 変数, 定数, program name, LB (r−1) (r≧1) に属する subroutine の名称などの表現の形式および長さの判定限度などを規定する Data は変数とする.

9. CS(A, B) 作成手順

(1) x=A, y=B, z=B を定める

(2) $\tilde{L}$(B) のための MACRO Declaration をかく.

(3) CS(A, B) に必要な機能を定めて, CS(x, y, z) by, L(u) から不要な部分を取除き, 不足の部分を加える.

(4) MACRO DECLATION+CS(x, y, z) by L(u)+(3) による修正を CGS に与えれば, CS(A, B) by $\tilde{L}$(B) が出る. これと同時にえられる data から情報 a(A, B) を作成する.

(5) CS(A, B) by $\tilde{L}$(B)+a(A, B) が求める compiler CS(A, B)=CS(A, B, B) である.

なおこの compiler は $\tilde{L}$(B) でかかれているが, CGS の base M(j) の出力機械で打ち出されたものであるから, その紙テープなどは M(B) にそのまま load できないことがある. その時は作成された compiler の implementation を M(B) のタイプライタでパンチしなおせばよい. もし M(B), M(j) のコードが異なるだけのときは, 変換ルーチンを通してから使用する.

次にあげる Program 例は実験したコンパイラの一般的表現を L(u) でかいたものの一部分である.

ここで CGS(x, y, z, U, j) の j は TOSBAC 3121, x は Fortran 的なもの, y は NEAC 2203, z は NEAC 2203 である.

```
REG READSB
?0    DC   TABLE   SEP 10 (5//2, sp 10), SEP 11
                   (5//2, sp 11),
                   SEP 12 (20//2, sp 12);
           ARRAY   KS (10/1);
           DMI     JEQH (A, B, YES, NO)
                    191000000000 152000000000
                    053001000000 434001000000;
                   CLEAR (A)
                    181000000000;
                   READ 1 (X)
                    66003 d 100200 111000000000;
                   RAISE (A, *n)
                    301000000000 28203 d 000000
                    111000000000;
                   ADDA (X, Y)
                    201000000000 112000000000;
                   RETURN (*1)
                    430100000000;
                   GOTO (?X)
                    431001000000;
                   LSFH (A, *n)
                    191000000000 51203 d 000000;
                   LOADI (x, y)
                    721200000000;
```

```
                STORI (x, y)
                 731200000000;
                EXTH (A, B, C)
                 152000000000 081000000000
                 113000000000;
                SETH (X, Y)
                 191000000000 112000000000;
                SET (X, Y)
                 301000000000 112000000000;
                 K
?1      CLEAR (W), CLEAR (NW), STORE
        (WMI, *1), STORI (WM 2, *2), STORI
        (WM 3, *3) K
?2      READ 1 (C) K
    SB  INCO (C, SEP 10, ? 2) K
?3  SB  INCO (C, SEP 11; ? 4) K
        RAISE (NW, *1) K
    SB  INC 1 (C, SEP 12; W, IW; ?W 20) K
        LSFH (W, *2), ADDA (C, W),
        GOTO (? 2) K
?4      JEQ (NW, zero, ? 2), LOADI (NW, 1),
        EXTH (W, KS@ 1, HW) K
?WCLS   JEQH (HW, D/IG, ?W 2), JEQH (HW,
        D/LET, ?W 3),
        JEQH (HW, D/?, ?W 4, ?WO 1) K
?W 2    SETH (CDusn, IW), GOTO (?W 20) K
?W 3 SB INC 1 (W, DEL; W, IW; ?W 20),
        INC 1 (W, LGVAL; W, IW; ?W 20),
        INC 1 (W, STANF; W, IW; ?W 20) K
        SETH (CDID, IW), GOTO (?W 20) K
?W 4    SETH (CDLAB, IW), GOTO (?W 20) K
?WO1 SB PRINT ('ERROR WCLS 1')
?W 20   LOADI (WM 1, *1), LOADI (WM 2, *2),
        LOADI (WM 3, *3), RETURN (*3) K
%
```

(説　明)

HW : word W の先頭の character
LGVAL : logical value の table
DEL : delimitter の table
STANF : standard function の table
CDusn : unsigned-integen の code
CDID : identifier の code
CDLAB : label の code
SEP 10　1文字よりなり完全によみとばせるものの table
SEP 11　1文字よりなり separater としてだけ意味のあるものの table
SEP 12　1文字よりなる operator 等の table

READSBは read subroutine の name subroutine の意味

INCO (x, y, ?α): If x $\in$ y then go to ?α else next.

INC 1 (x, y; a, b; ?α, ?β):
If x $\in$ y then a:=code of X,
b:=information of X
and go to ?α,
else go to ?β.

MACRO の意味

JEQH (A, B, ?α, ?β)
: If A=B then go to ?α else ?β.
CLEAR (A) : A:=0
READ1 (X) : Read 1 character from tape reader and put it in X.
RAISE (A, *n) : A:=A+n
ADDA (x, y) : Y:=ACC+X
GOTO (?X) : go to ?X
LSFH (A, *n) : ACC=A and left shift ACC by N character
LOADI (x, y) : index (Y):=X
EXTH (A, B, C): C:=A∩B (∩is logical and)
SETH (x, y) : Y:=X (x, y は内容を文字として)
SET (x, y) : Y:=X (x, y は内容を数値として)
STORI (x, y) : X=ind (Y)
RETURN (*3) : go to the address indicated by index (3)

参考文献

1) 野口広，藤野喜一，渡部和，若月宏: NUMERIC-Automatic Coding System for the NEAC 1103 (NEAC 1103 自動プログラム体系)，1963 年 7 月 25 日，電子計算機研究会資料電気通信学会
2) 藤野喜一: NUMERIC における ALLOCATION の問題，昭和 38 年 12 月 5 日，情報処理第 4 回全国大会予稿

(昭和 41 年 2 月 10 日受付)