Compiler Generating System

Kiichi Fujino*

1. *Introduction*

Generally, we have written the implementation of a compiler system (*CS*) by the machine language (including assembler language) of a specified digital computer. In this case the generated *CS* is sufficiently optimal because we can fully use the speciality of machine instructions. However it requires a great deal of effort and considerable time (5-10 man years). Therefore, it has been desired to develop a method sufficiently simple and requiring a shorter interval to make a *CS* with higher capabilities. The program which realizes this idea is called the Compiler Generating System (*CGS*). This paper presents a basic investigation and a method of realization.

The author is grateful for Professor H. Noguchi of WASEDA University for his useful advice and to Professor M. Namba who has given so much machine-time to the author and to S. Katagai who assisted in the implementation.

2. *Notaion of the Compiler System*

To make clear the nature of the compiler system, the following notations are used.

2. 1. *Notation of CS.* Let $L(\iota)$ be one of the automatic programming language (*APL*) where $i \in I=\{1,2,....,n\}$. For example, $L(i)$s are ALGOL, FORTRAN and NUMERIC (c.f. [1], [2]). Let $M(i)$ be a computer and $\tilde{L}(i)$ be its machine language, where $i \in J=\{1,2,....,m\}$. A *CS* is denoted as $CS(x,y,z)$; where the descriptive language (*DL*) of the source program given to *CS* is $L(x)$, the *DL* of the implementation of *CS* is $\tilde{L}(y)$, and the *DL* of the object program is $\tilde{L}(z)$. In an ordinary compiler, since $y=z$, thus $CS(x,y,z)=CS(x,y,y)$. It is thus written as $CS(x,y)$.

2. 2. *Function of CS.* The input language $L(i)$ of $CS(i,j,k)$ has some effect from the machine $M(j)$ in its representation, so it is denoted as $L(i,j)$. Let $P(i,j)$ be a program written on $L(i,j)$, then $CS(i,j,k)$ translates $P(i,j)$ to the object program $\tilde{P}(i,k)$. If the function of $CS(i,j,k)$ is written as $f(i,j,k)$, the relation

---

This paper first appeared in Japanese in Joho Shori (the Journal of the Information Processing Society of Japan), Vol. 7, No. 5 (1966), pp. 245-255.

* Waseda University Electronic Computation Center.

$$f(i,j,k)\,P(i,j)=\tilde{P}(i,k)\subset P(\ .k)$$
$$(P(i,j)\longrightarrow\boxed{CS(i,j,k)}\longrightarrow\tilde{P}(i,k))$$

holds, where $P(.k)$ is the set of programs written in $\tilde{L}(k)$.

Now let $P(i,j)$ and $P(i',j)$ be programs for a problem P. If we write $P(i,j)\equiv P(i',j)$ to show they are equivalent in mathematical meaning, there should be a a translation $\varphi(i,i',j)$ from $P(i,j)$ to $P(i',j)$ because the program structures are different in general. Therefore the next relations hold,

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

and $\tilde{P}(i,j)\equiv\tilde{P}(i',j)$ should be true. Let $\sigma(i,i',j)$ denote the transformation from $\tilde{P}(i,j)$ to $\tilde{P}(i',j)$,

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

holds and hence the relation

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

is obtained. This relation is the basic assumption for generation of a programming system for each machine.

2. 3. *Self Expressible CS.* Suppose a $CS(i,j,j)$ exists, and let $P(CS(i,j,j)/L(i,j))$ denote the implementation of $CS(i,j,j)$, itself written with its input language $L(i,j)$. When this program is given to $CS(i,j,j)$ as a source program and if the object program $CS^*(i,j,j)=\tilde{P}(CS(i,j,j)/L(i,j))$ on $\tilde{L}(j)$ is the implementation having an identical function with $CS(i,j,j)$, then $CS(i,j,j)$ is said to be self expressible. The usual compiler does not always have this property, but if we neglect the speciality of $M(y)$ and the limit of the memory, we may consider each compiler to be (in wide sense) self-expressible.

One of the sufficient conditions for $CS(i,j,j)$ to be self-expressible is that $L(i,j)$ contains $\tilde{L}(j)$ as a subset. This property is important to analyze *CGS*.

3. *Compiler Generating System* (*CGS*)

3. 1. *Notation of CGS.* Let $CGS(x,y,z\,;U,j)$ be a representation of *CGS*, where the input language of *CGS* is $L(U)$, the implementation language of *CGS* is $\tilde{L}(j)$, and x,y,z represent that *CGS* generates a compiler $CS(x,y,z)$.

The input data of *CGS* are:

(i) The general representation of $CS(x,y,z)$ written with $L(U)$.

(ii) The set $\delta(A,B,C)$ of necessary information,

The output data of *CGS* are:

(i) $CS(A,B,C)$ written with $\tilde{L}(B)$

(ii) The number of words contained in the implementation of $CS(A,B,C)$ and other

information.

Generally $L(U)$ and $M(j)$ are fixed for a *CGS* so that it may be written as *CGS* (x,y,z). On the other hand, $CS(x,y,z)\in P(.y)$, $CGS(x,y,z\,;\,U,j)$ can be considered as a compiler $CS(U,j,y)$. Therefore not only *CGS* has the same function as *CS*, but also *CGS* needs to be able to change the part associated with the output language $\tilde{Z}(y)$. In addition, the information defined by x,y,z has to be given as not constant but variable in the source program written with $L(U)$. Therefore, it is necessary for us to make clear structure of $CS(x,y,z)$ in the problem to realize a $CGS(x,y,z)$.

4. *Constitution of Compiler*

4. 1. *Constituent Elements of Automatic Language* (*APL*). There are three kinds of constituents for *APL* $L(x)$. *Carrier* representing data for the program, *Subroutine* which has its own name, and is a procedure representing a mapping from α to β, where α and $\beta\subset$ C. *Programming word* which is necessary to write a program other than the two elements above. These sets are denoted by *C*, *Z*, and *W* respectively.

4. 2. *Structual property of* $L(x,y)$. The set of operators (arithmetic and logical) belonging to $L(x,y)$ and the instruction codes of $\tilde{L}(y)$ contained in $L(x,y)$, is called set of rank-0 subroutines of $L(x,y)$, and denoted by Z^0. Z^0 is called rank-0 Library written as $LB(0)$, which is naturally unique to each $L(x,y)$. A program written on $LB(0)$ is called rank-1-program and when it is registered to Library it is called a rank-1-subroutine whose set is denoted by Z^1. Standard subroutines sin, cos and $\sqrt{\ }$(square-root) etc. belong to Z^1 in ALGOL and FORTRAN. A program written by using subroutines of LB(r−1) represented as $\bigcup_{k=0}^{r-1} Z^k$ is called a rank-r-program. If it contains the subroutines of Z^{r-1}, it is said to be proper, and when registered it is called a rank-r-subroutine and its set is written as Z^r. For example, a program solving a quadratic equation is rank-2, since it uses the subroutine $\sqrt{\ }\in Z^1$.

Each program element (*i.e.* statement) is represented as $e=(z,c,w)$ in the structural meaning, $z\subset LB(r\text{-}1)$, $c\subset C$ and $w\subset W$.

Examples:

In DC (declare) element, no subroutine is contained, thus it can be written as $e^0=(\phi,$c,w). Ex. array A, B (100, 20) etc..Here ϕ represents an empty set.

In OP (operation) element, $z\subset LB(1)$. Ex. $A+B\rightarrow C$; $\sin(X+Y)+SQRT(X**2+Y**2)\rightarrow T$; Here $<+,**>\subset Z^0$, <sin, sqrt $(=\sqrt{\ })>\subset Z^1$.

In MI (machine instruction) element, $z\subset LB(0)$ and z are machine codes. Ex. ADD (X); <ADD> $\in Z^0$.

In SB (subroutine calling) element, $z\subset Z^k\,(1\leq k\leq r-1)\subset LB(r-1)$. Ex. MATRIXINVERS (X,Y); <MATRIXINVERS> $\in LB(1)$.

In SW (switch) element, $z\subset LB(1)$. Ex. α: (SIN(X) gtq COS(Y))$\rightarrow\beta$: else goto γ:;,

here $<$SIN, COS$>\subset Z^1, <\text{gtq}(\geqq)>\in Z^0$.

4. 3. *Main-part and Reduction.* Generally, writing a program with $L(x,y)$ is to write it by using subroutines in $LB(r\text{-}1)$ set up by that time. In this meaning, a rank-r-program is denoted as $[s]=(X,S)/\text{on } LB(r\text{-}1)$ and we call it the main-part of the program s. Here X is the set of blocks in s and S is the set of switches in s.

A subroutine which is contained in s and not belonging to Z^0 is called a directly subroutine of s and its set is written as $D(s)=\{S^1, S^2,\ldots,S^{r-1}\}$ and $S^i=\bigcup_{k=1}^{l(k)} s_k{}^i$, $s_k{}^i\in Z^i(1\leq i\leq r\text{-}1)$, and we assume at least one of S^i is not empty.

Reduction ($\rho(s)$) of program s is defined as that in which we replace all subroutines $s_k{}^i$ of $D(s)$ with their main-parts; in the other words, we replace rank-i-subroutine $s_k{}^i(\in Z^i)$ with $[s_k{}^i]$ on $LB(i\text{-}1)$. If the subroutines of $D(s)$ are at most rank-i $(1\leq i\leq r\text{-}1)$, the subroutine of $D(\rho(s))$ is at most rank-$(i\text{-}1)$. So on most occasions $(r\text{-}1)$ reductions a rank-r program can be reduced to program on $LB(0)$.

Example; Let the main-part of a program α be $[\alpha]=(\beta_1, \beta_2. \beta_3, \alpha^0, C_\alpha)$ where $D(\alpha)=\{\beta_1,\beta_2,\beta_3\}\subset LB(r-1)$ and C_α is the set of carriers in α, $\alpha^0\subset Z^0$ $C\alpha\subset C$, then $\rho^3(\alpha)$ is a rank—0—program. The following in formation is stored in LB $(r-1)$.

$$\beta_1=(\gamma_1,\ \gamma_1,\ \delta_1,\ C_{\beta_1}),\ \beta_2=(\gamma_2,\ C_{\beta_2}),\ \beta_3=(\gamma_3,\ \delta_2,\ C_{\beta_3}),\ \gamma_1=(\delta_1,\ \delta_2,\ C_{\gamma_1}),$$
$$\gamma_2=(\delta_3,\ C_{\gamma_2}),\ \gamma_3=(\Phi,\ C_{\gamma_3}).\ \delta_1=(\Phi,\ C_{\delta_1}),\ \delta_2=(\Phi,\ C_{\delta_2}),\ \delta_3=(\Phi,\ C_{\delta_3})$$

They are illustrated as follows:

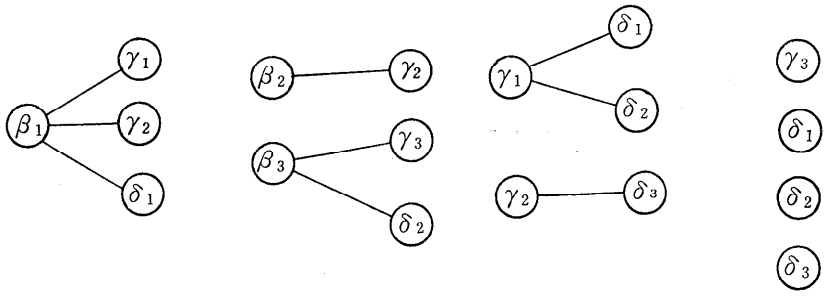

Fig. 1.

Therefore,

$$\rho(\alpha)=[\alpha]\ [\beta_1]\ [\beta_2]\ [\beta_3]$$
$$\rho^2(\alpha)=[\alpha]\ [\beta_1]\ [\beta_2]\ [\beta_3]\ [\gamma_1]\ [\gamma_2]\ [\delta_1]\ [\gamma_3]\ [\delta_2]$$
$$\rho^3(\alpha)=[\alpha]\ [\beta_1]\ [\beta_2]\ [\beta_3]\ [\gamma_1]\ [\gamma_2]\ [\delta_1]\ [\gamma_3]\ [\delta_2]\ [\delta_3].$$

This is illustrated by Fig. 2.

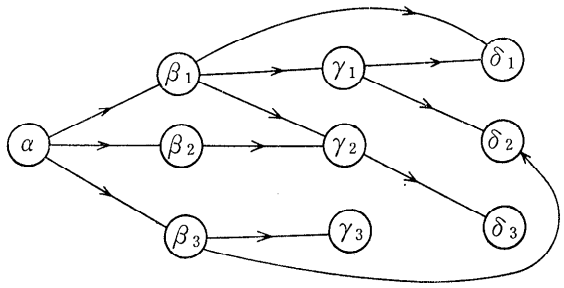

Fig. 2.

5. *Function of CS(x, y)*

$CS(X, Y)$ has four parts named INTRO, DECLARE, TRANSFORMATION and ALLOCATION.

(1) INTRO and DECLARE parts change a source program $s=\{s_{ij}\}$ written on $LB(r-1)$ of $L(X)$ to $\bar{s}=\{\bar{s}_{ij}\}$ on standard code of $CS(X, Y)$.

(2) TRANSFORMATION part (Operation, Substitute and Switching) transform $\bar{s}$ to $\acute{s}$ by the following steps.

1) Divide $\bar{s}_{ij}$ the transformable relation $\bar{s}_{ijk}$.

2) Generate the object relation $\bar{s}_{ijk}$ on $\bar{LB}(r-1)$ of $\tilde{L}(Y)$ corresponding to $\bar{s}_{ijk}$.

3) Make the undecided coefficient $\tilde{a}_{ijk}$ for allocation associated with $\tilde{s}_{ijk}$.

Therefore $\tilde{s}=\{\tilde{s}_{ijk}\cup\tilde{a}_{ijk}\}$ is obtained.

(3) ALLOCATION part

1) Make the decided information $ALI(\tilde{a}_{ijk})$ for $\tilde{a}_{ijk}$.

2) If s requires registration, $\tilde{s}$ is made as closed subroutine and register it into $\tilde{LB}(r-1)$ of $\tilde{L}(Y)$.

3) If s requires reduction, $\tilde{s}$ is reduced to s on $\tilde{LB}(0)$ of $\tilde{L}(Y)$.

6. *Methods of CGS*

6. 1. *Self Growing method.* In this method,

(1) Make a basic compiler $CS^0(i, j, j)$ being necessary for self-expansion. Its input language is $L^0(i)$, being a basic part of $L(i)$.

(2) Write the implementation of the next step compiler $CS^1(i, j, j)$ using $L^0(i)$, and give it to $CS^0(i, j, j)$, then the resultant $CS^1(i, j, j)$ has the input language $L^1(i)$ containing $L^0(i)$ as a subset.

(3) In the same way as (2), generate $CS^n(i, j, j)$ from $CS^{n-1}(i, j, j)$ until $L^n(i)$ of $CS^n(i, j, j)$ satisfies the required specification of $L(i)$.

$$\text{Program } (CS^p(i, j, j)/\text{on } L^{p-1}(i)) \rightarrow \boxed{CS^{p-1}(i, j, j)} \rightarrow CS^p(i, j, j).$$
$$(1\leq p\leq n)$$

6. 2. *Transformation method.* When one $CS(i, j, j)$ exists, we generate $CS(i, k, k)$ having the same function with $CS(i, j, j)$, through the following steps in this method.

$$CS(i, j, j) \overset{(i)}{\text{——}} CS(i, j, k) \overset{(ii)}{\text{——}} CS(i, k, k).$$

Step i) 1) A table called MIGT(j) containing the group of $\tilde{L}(j)$ instructions composing each operation, and a table called OSNT(j) associated with the number representing the stored position on MIGT(j), are converted to MIGT(k) and OSNT(k) respectively. 2) Modify the address part of the instructions contained in the implementation of $CS(i, j, j)$ about the conversion of MIGT. 3) If the basis of the numeric expression of $\tilde{L}(j)$ and $\tilde{L}(k)$ is different, we change the counter necessary to the computation of the undecided coefficient of allocation and the necessary counter to the relative address of variables and constants, to the base of $\tilde{L}(k)$. Otherwise, the necessary function for modification of the basis is given to the loading routine of $\tilde{L}(k)$.

step ii) Write $CS(i, j, j)$ with $L(i, j)$ and input it to the $CS(i, j, k)$, then we have $CS(i, k, k)$.

7. *L(U) of CGS*

In this section $L(U)$ is described briefly.

7. 1. *Special declaration*

TABLE Declare statement, TABLE SEP-10 (5//2, sp-10) defines a table named SEP-10 and having 5 terms, and each term has two elements and sp-10 is a variable representing the current depth of the table. Such informations about each table is stored in the table called TABLENAME-TABLE.

wl It specifies the word length of variables being not array. For example, wl x, y, $z(2)$;. This means the word length of x, y and z is 2 words.

7. 2. *Operation Part (OP)*

This part represents computation ; the following operations are allowed : Arithmetic operator (+, —, *), Relational operator (gtr(>), geq(≧), =, neq(≠)), Logical operator (or and, not), Special operator (//word pointing operator, $A//2$ means the 2nd word of A), Assignment operator (→). Operators associated with index : *ind*, for example ind(2) means the content of the index-2, @, $A@2$ represents the element specified by the index-2.

7. 3. *Switching Function (SW).* It has the forms :

SW GOTO ?α ; or

SW(B_1)→(C_1), (B_2)→(C_2), ..., (B_{n-1})→(C_{n-1}), ELSE(C_n) ; is equivalent to

if B_1 then C_1 else if B_2 then C_2 if, ..., if B_{n-1} then C_{n-1} else C_n ;

Here ?α is a label, label identifier must begin with symbol ?.

7. 4. *Subroutine calling (SB)*

For example, SB INCI(X, Y;a , b; ?p, ?q); See Example in §9.

7. 5. *MACRO instruction.* We define the following MACRO instruction having the speciality of the machine codes fully, to level up the efficiency of the generated compiler, and we can make the necessary MACRO instructions by using declaration statement described below.

MACRO Declaration Statement:

(A) DC DMI XXXXX(X_1, X_2,....., X_m)

(B)
[machine instruction (1)]
[machine instruction (2)]
—
—
—
[machine instruction (n)]

Part (A) XXXXX is an identifier representing the name of MACRO instruction. (X_1, X_2,....., X_m) is the list of the formal variables, where $0 \leq m \leq 10$.

Part (B) This is the sequence of machine instructions ($\in \tilde{L}(y)$) composing MACRO operation, and each instruction is a string of 12 digits. For example, MACRO order JEQ represents a statement:

if A=B then go to ?P else ?Q;

```
DMI JEQ  (A, B, ?P, ?Q)
301000000000
212000000000
423001000000
434001000000
①  ②  ③
```

① Operation code ($\epsilon \tilde{L}(y)$)
② Number or necessary information
③ Flag representing to require information as a coordinate.

Fig. 3.

These four machine instructions are stored, for example, from the location ξ to $\xi+3$ of the table called machine instruction table (MIGT), and the information

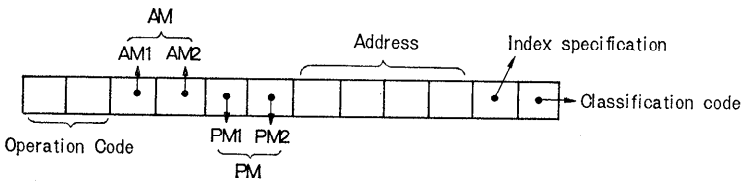

Fig. 4.

(ξ, 4) are put on the MACRONAMETABLE (MNT). MAIT is a table storing up to 10 actual parameters. Machine instruction is illustrated as Fig. 4.

If AM part i.e. (AM1, AM2)=(i, j), then its address part is made from the i-th actual parameter stored in MAIT (j). PM part (PM1, PM2) shows the print format of address part and the parameter to be attached to the instruction. If PM1 =0, information is given from other part. If PM1=2, the address is made as absolute address. PM2 represents the kind of parameters:

1: coordinate, 2: variable, 3: constant, 5: array variable, 6: formal variable, 7: working memory, 9: common variable.

The content of classification codes specifies the following characteristics of the instruction:

0: the address part is not required from MAIT.

instruction	result
5900e9000200	59+0002;
33003d002200	33*000b;
120003000100	12*0001 (constant);

1: the left operand is required.

instruction	left operand	output form
120000000001	2000200	12*0002 (variable);
120000000001	5040110	12*0401 (array variable), 1 index;

2: the right operand is required, as same as case of 1:

3: subroutine parameter is required.

instruction	output form
4200001ext03	42+0001*0(ext);

4: only the address part is required.

instruction	output form
000003000504	*0005(constant);

Remarks: We can construct a CS^0 (i, j, j) by using only variable, constant, coordinate, PRG, REG, DC (array, TABLE) and MACRO instruction in the grammer of L(U). The example at the end of this paper shows this method.

8. *Notes on Input Data*

The size of the tables cannot be decided because each compiler naturally a dif ferent capacity according to the base machine M(B), and we cannot decide the maximum number of the variables, constants, formal variables, labels and array variables available before the implementation of CS(A, B) is generated. Moreover, the format of each of the terms of tables and the size of tables should be

given as variables. Representataion of elements and their internal codes are affcctcd with the input and output equipment of the base M(B). Therefore, we have to write the syntactical data specifying these elements with variablcs.

9. $CS(A,B)$

The following steps explain a method to generate a compiler $CS(A, B)$.

1) Decide $X=A$, $Y=B$, $Z=B$

2) Make MACRO orders declaration for $\tilde{L}(B)$

3) Remove the unnecessary parts from $CS(X, Y, Z)$ written with $L(U)$, and append the lacking parts to it.

4) Give the MACRO declaration (by step 2)) and the modified $CS(X, Y, Z)$ on $L(U)$ (by step 3)) into CGS, then formal implementation of $CS(A, B)$ written with $\tilde{L}(B)$ and information $a(A, B)$ associatd with it are obtained.

5) Required compiler is $CS(A, B)$ on $\tilde{L}(B)+a(A, B)$

Note: Generated programs cannot often be directly loaded into $M(B)$ since they are typed out by the output equipment of the base machine $M(j)$ of CGS. In this case we have to type it with equipment of $M(B)$.

The following program example is a part of representation written with L(U) of a compiler. This is the input information to the experimental CGS($x, y, z; U, j$) where j is TOSBAC-3121, x is a language similar to FORTRAN, y is NEAC-2203 and z is NEAC-2203. These machines are installed at WASEDA University.

```
REG READSB
?0 DC TABLE SEP-10 (5//2, sp-10), SEP-11 (5//2, sp-11), SEP-12 (20//, sp-12) :
      ARRAY KS (10//1) ;
      DMI     JEQH (A, B, YES, NO)
               191000000000 152000000000 053001000000 434001000000 ;
              CLEAR(A)
               181000000000 ;
              READ 1 (X)
               66003d100200 111000000000 ;
              RAISE (A, *n)
               301000000000 28203d000000 ; 111000000000 ;
              ADDA (X, Y)
               201000000000 112000000000 ;
              RETURN (*1)
               4301000000000 ;
              GOTO (?X)
               431001000000 ;
              LSFH (A, *n)
               191000000000 51203d000000 ;
              LOADI (X, Y)
               7212000000000 ;
              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), STORI (WMI, *1), STORI (WM2, *2), STORI
               (WM3, *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 ; ?W20) K
               LSFH (W, *2), ADDA (C, W), GOTO (? 2) K
?4             JEQ (NW, zero, ? 2), LOADI (NW, 1), EXTH (W, KSO 1, HW) K
? WCLS           JEQH (HW, D/IG, ? W2), JEQH (HW, D/LET, ? W3), JEQH (HW, D/?, ?W4,
                 ? WO 1) K
? W 2            SETH (CDUSN, IW), GOTO (? W20) K
? W 3    SB      INC 1 (W, DEL ; W, IW ; ?W20), INC1 (W, LGVAL ; W, IW ; ? W20),
                 INC 1 (W, STANF ; W, IW ; ? 20) K
                 SETH (CDID, IW), GOTO (?W20) K
? W 4            SETH (CDLAB, IW), GOTO (? W20) K
? W 01   SB      PRINT ('ERROR WCLS 1')
? W 20           LOADI (WM1, *1), LOADI (WM 2, *2), LOADI (WM 3, *3) RETURN (*3) K
    %
```

Explanation of the words used in the above program:

SEP 10...table of the elements we can skip and their length is 1 character.

SEP 11...table of separators whose length is 1 character.

SEP 12...table of operators whose length is 1 character

ADDA (X, Y)...$Y := \mathrm{ACC} + X$

CLEAR (A)...$A := 0$

EXTH (A, B, C)...$C := A \cap B$ ($\cap$ is logical and operator)

JEQH (A, B, YES, NO)...If $A = B$ then go to YES else go to NO;

GOTO (? X)...go to ? X;

LOADI (X, Y)...index (Y) $:= X$

LSFH (A, *n)...ACC $:= A$, and left shift the content of ACC by n characters.

RAISE (A, *n)...$A := A + n$

READ 1 (X)...Read 1 character from tape reader and put it in X.

RETURN (*3)...go to the address indicated by index (3).

SET (X, Y) ...$Y := X$ (X, Y are numbers)

SETH (X, Y)...$Y := X$ (X, Y are numbers)

STORI (X, Y)...$X :=$ index (Y)

INCO (X, Y, ? p)...if $X \in Y$ then go to ? p else go to next statement.

INC 1 (X, Y; a. b; ? P, ? Q)—if $X \in Y$ then begin

a := code of X;

b := information of X; go to ?P end else go to ?Q;

HW—heading character of word W

LGVAL—table of logical value

DEL—table of delimiter

STANF—table of standard function

CDUSN—code of unsigned-integer

CDID—code of identifier

CDLAB—code of label
READSB—name of read subroutine
REG—Process mark requiring to compile the following source program as a subroutine named READSB (in the above example) and register it into CGS Library.

References

[1] NOGUCHI, H., K. FUJINO, H. WATANABE AND H. WAKAZUKI, Numerical-Automatic System for the NEAC-1103 (NEAC-1103 Automatic Programming System) (in Japanese). *The papers of Technical Group on Electronic Computers.* The Institute of Electronics and ommunication Engineers of Japan, July (1963).

[2] FUJINO, K., A Problem of Allocation in Numeric. *The papars of Annual Conference of the Information Processing Society of Japan*, December 5, 1963.