# SPM—A Pseudo Machine for String Processing

KIYOSHI ASAI*, YASUO INAMI* AND NAOYUKI SAITO*

Recently much of technical information is being exchanged or disseminated in the form of computer programs. For engineering computation, most of computer programs are written in FORTRAN languages. The FORTRAN has, however, many dialects and, for this reason, it is an important problem to convert FORTRAN dialects into the standard FORTRAN or to convert a dialect into an another dialect.

Features of a conversion program are basically concentrated as follows;

(1) versatile features to process strings,

(2) efficient set-theoretic operations for sets of strings.

The conversion program SPM is a general purpose program which executes the above mentioned features efficiently.

## 1. *Introduction*

Many computer programs have been developed to rewrite a FORTRAN language to another FORTRAN dialect [1]. These programs are called conversion programs. The each method used in the conversion programs has its own merits and demerits. We can say that common demerits of the methods are lack of feasibility and efficiency, because these methods are too rigid and cannot apply to an appropriate procedure according to complexity of problems. We can, however, solve the problem by using a symbol manipulation language specially designed for string processing. By the word string we mean a string of characters. We may classify basic functions to convert a FORTRAN program as follows;

(1) arithmetic operations on integers,

(2) set-theoretic operations on sets of strings,

(3) dynamic labels which we can define or refer dynamically,

(4) operations on variable length strings,

(5) quick access to strings,

(6) efficient recognition of characters,

(7) quick branching operations by characters,

(8) indirect addressing by characters.

We must moreover add following three functions if we want to realize these functions on a computer program;

This paper first appeared in Joho-Shori (the Journal of the Information Processing Society of Japan), Vol. 11, No. 1 (1970), pp. 11–19.

* Japan Atomic Energy Research Institute, Tokaimura, Ibarakiken.

(9) input/output operations on variable length strings,

(10) easy programming as a symbol manipulation language,

(11) compatibility between computers.

The SPM is a FORTRAN written pseudo machine which aims to meet the above mentioned functions [2].

## 2. *SPM*

The SPM is a pseudo machine with two-address instructions. The address is given to each string. The two-address method is adopted since the string processing in general consists of operations such as string comparison, copy, retrieval and transfer on two strings.

The SPM operation is represented in the form

OPC ADR, BDR, VAR

where OPC, ADR, BDR and VAR mean operation code, A-address, B-address and variant respectively.

### 2.1 *Registers*

(1) IC: Instruction counter

This register contains the address of the current SPM instruction.

(2) ADR: A-address register

This register contains an address of a string specified by A-address part of an instruction. When a dynamic label or an indirect address is specified in the A-address part, the address decoder analyzes the specification and put a true string address into the ADR.

(3) BDR: B-address register

This register contains an address of a string specified by the B-address part of an instruction. Its operation is similar to the ADR.

(4) VAR: Variant register

This register contains a character which is used as a string boundary of transfer operation or an object of a comparison.

(5) APR: A-address character pointer register

Strings specified by the address parts of an instruction are processed character by character by the SPM. The APR contains the next character position of A-address at that place the processing has just finished.

(6) BPR: B-address character pointer register

This register is similar to the APR.

(7) IND: Indicator register

This register contains a numerical value which is set by a result of an instruction execution.

### 2.2 *Labels*

There are two classes of strings in the SPM operations. The one is a set of

strings defined before the SPM instruction execution and the other is a set of strings that are generated in the process of the SPM instruction execution. Labels for the former strings are celled static labels and labels for the latter strings are called dynamic labels.

Variables used in the SPM arithmetic operations are also assumed as strings with numeric values and the variable names are treated as labels.

These labels are the numeric labels. We can also give static labels to the SPM instructions. These labels are the instruction labels and are mainly used for branching addresses.

Thus we can classify the labels as Fig. 1.

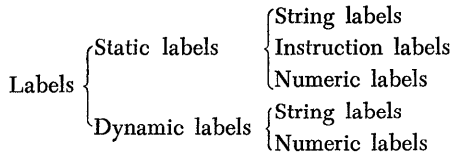


Fig. 1. Labels

2.3 *Label Representations*

(1) A static label is defined by a string of up to four characters. The string may be composed of 0,1, ……, 9, A, B, C, ……, Z and special characters (excluding (,), ., ', comma and blank).

(2) A dynamic label is defined by two static labels and a period between the two labels. The latter label must be a numeric label.

(3) The above two labels can have one dimensional subscripts (except the numeric label and the instruction label) to specify the character positions in the strings.

(4) The SPM accepts indirect addresses of one depth.

(5) In case of a datum transfer to a string of dynamic label, the length of the string is extended automatically according to the datum length. In case of a datum transfer to a string of static label, the length of the string is not extended automatically (except in RCS and CAT operations) and the transfer finishes at the boundary of the datum or the string.

(6) Thus labels and their addressing methods are classified as Fig. 2.

| | Simple addressing | Subscript addressing | Indirect addressing |
|---|---|---|---|
| Numerical label | YES | NO | YES |
| Instruction label | YES | NO | NO |
| String label | YES | YES | YES |

Fig. 2. Labels and their addressing methods.

(7) Examples of label definitions

(a) Simple direct and indirect addressings KOKO, (KOKO)

```
        MCM     KOKO, WORK, '('
        MCM     (KOKO), SPAC
          :
KOKO    DS      'READ ('
WORK    DS      '#####INPUT#TAPE'
SPAC    DS      '######OUTPUT#TAPE'
READ    DS      'WRITE'
```

Note: A # means a blank.

Fig. 3.

After the execution of the first MCM instruction in the above example, the contents of the string KOKO is transferred to WORK and as the result, the WORK contains a string READ INPUT TAPE. After the execution of the second MCM instruction, the contents of the string READ is transferred to SPAC and as the result, the SPAC contains a string WRITE OUTPUT TAPE.

(b) Direct and indirect addressing with subscripts KOKO(P), (KOKO(P))

```
        ADD     ONE, P
        MCM     KOKO (P), WORK, '('
        MCM     (KOKO(P)), SPAC
          :
KOKO    DS      'READ ('
WORK    DS      '#####INPUT#TAPE'
SPAC    DS      '######OUTPUT#TAPE'
READ    DS      'WRITE'
P       DC      0
ONE     DC      1
```

Fig. 4.

In the above example, all results are same as the example (a)'s.

(c) Dynamic direct and dynamic indirect addressing A.P, (A.P)

```
A.      SIZE    100
        ADD     ONE, P
          :
        MCM     KOKO, A.P, '('
        MCM     (A.P), SPAC
          :
KOKO    DS      'READ ('
WORK    DS      '#####INPUT#TAPE'
SPAC    DS      '######OUTPUT#TAPE'
READ    DS      'WRITE'
P       DC      0
ONE     DC      1
```

Fig. 5.

After the execution of two MCM instructions in the above example, the SPAC contains a string WRITE OUTPUT TAPE.

(d) Dynamic direct and indirect with subscripts addressing A.P(Q), (A.P(Q))

```
A.      SIZE    100
          :
        ADD     ONE, P
        ADD     ONE, Q
        MCM     KOKO, A.P(Q), '('
        MCM     (A.P(Q)), SPAC
          :
KOKO    DS      'READ ('
WORK    DS      '##### INPUT # TAPE'
SPAC    DS      '###### OUTPUT # TAPE'
READ    DS      'WRITE'
P       DC      0
ONE     DC      1
Q       DC      0
```

Fig. 6.

In the above example, all results are same as the (c)'s.

2.4 *Instructions*

The SPM can allow the address chaining specification. Some instructions must have labels with or without subscripts. When labels are not specified explicitly for these instructions, the contents of previous ADR, BDR, APR and BPR registers will be used if necessary. The address chaining of the variant register is not allowed. All instructions of the SPM are shown in the Table 1. The Table 2 shows the indicator status after the instruction execution.

Let us give explanation about some instructions to make the SPM functions clear.

CS: Compare strings

```
CS    ADR, BDR
```

```
CS    ADR
```

```
CS
```

In this instruction the ADR and BDR labels are assumed to be string labels. These labels may have subscripts and if subscripts are not specified explicitly, they are assumed to be 1's. The SPM compares ADR and BDR strings from the character positions specified by subscripts. The SPM finishes the operation CS if the two strings are equal or all characters of one of the two strings are used or unequal characters are found in that comparison. If the two strings are equal, the value of indicator is set to 5 and it is set to 6 otherwise. The

Table 1. The instructions of SPM.

| Instructions | Arithmetic Control |
|---|---|
| ADD | Add ADR to BDR |
| SUB | Subtract ADR from BDR |
| ZADD | Zero and add to BDR |
| ZSUB | Zero and subtract from BDR |
| MPY | Multiply ADR and BDR |
| DIV | Divide BDR by ADR |
| AND | And ADR to BDR |
| OR | Or ADR to BDR |
| COMPL | Store complement of ADR in BDR |
| | **Logical Control** |
| CS | Compare strings |
| CN | Compare numbers |
| BCE | Branch if character equal |
| BCT | Branch on condition test |
| BRA | Branch unconditional |
| NOP | No operation |
| | **Data Control** |
| CNVRT | Convert string to binary or binary to string |
| MCM | Move characters to mark or string boundary |
| MAC | Move a character |
| MCS | Move characters and suppress variant |
| SAR | Store A address register |
| SBR | Store B address register |
| SAP | Store A address character pointer |
| SBP | Store B address character pointer |
| STI | Store indicator |
| SRCH | Search a string ignoring variant |
| RCS | Replace a character by a string |
| ERASE | Erase a string |
| CAT | Catenate ADR and BDR setting variant |
| LEN | Get length of ADR string |
| LDA | Load A to ADR |
| MOD | Get absolute modulo of BDR |
| LAST | Get remaining available storage size |
| TON | Trace on |
| TOFF | Trace off |
| | **System Control** |
| REWIND | Rewind a unit |
| READ | Read a string |
| PUNCH | Punch a string |
| PRINT | Print a string |
| WRITE | Write a string |
| EXIT | Exit from SPM control |
| | **Processor Control** |
| DS | Define string |
| ETC | Extend card for DS operation |
| DC | Define constant |
| SIZE | Define max size of dynamic label |
| END | End of SPM instructions |

Table 2. The indicator status of SPM.

| Indicator Status | Explanation |
|---|---|
| 1 | [ADR]<[BDR] in CN operation |
| 2 | [ADR]=[BDR] in CN operation |
| 3 | [ADR]>[BDR] in CN operation |
| 5 | [ADR]=[BDR] in CS operation |
| 6 | [ADR]≠[BDR] in CS operation |
| 10 | A subscript greater than BDR string length specified in BCE operation |
| 12 | String boundary of ADR is reached in MCM operation |
| 13 | Static string boundary of BDR is reached in MCM operation |
| 15 | String boundary of ADR is reached in MAC operation |
| 16 | Static string boundary of BDR is reached in MAC operation |
| 18 | String boundary of ADR is reached in MCS operation |
| 19 | Static string boundary of BDR is reached in MCS operation |
| 21 | [ADR]=[BDR] in SRCH operation |
| 22 | [ADR]≠[BDR] in SRCH operation |
| 26 | A subscript greater than ADR string length specified in CAT operation |
| 31 | A subscript greater than ADR string length specified in CNVRT oper. |
| 32 | A subscript greater than BDR string length specified in CNVRT oper. |
| 33 | String boundary of BDR is reached before all of ADR binary numbers are not converted in CNVRT operation |
| 34 | First character is not a number in characters to a number CNVRT oper. |

Note: [ADR] means the contents of ADR string.

APR and BPR point to next character positions the comparison finished.

BCT: Branch on condition test

```
BCT     ADR, BDR
```

In this instruction, the ADR must be an instruction label and the BDR must be a number. In the BCT instruction execution, the number specified by the BDR is compared with the contents of the indicator. When they are equal, the SPM transfers control to the address specified in the ADR. When they are not equal or no static label indirectly addressed by the ADR exists, the SPM transfers control to next instruction of the BCT. After execution of the BCT instruction, the value of the indicator is set to zero.

CNVRT: Convert string to binary integer or binary integer to string

```
CNVRT     ADR, BDR, V
```

This instruction is used to convert a character string to a positive integer or a positive integer to a character string. If a number 1 is specified in the variant part, a string specified by the ADR is converted to an integer and the integer becomes the content of the numeric label specified by the BDR. If 2 is specified in the variant part, content of a numeric label specified by the ADR is converted to a character string and the string is embedded in a position of

the string specified by the BDR.

MCM: Move characters to mark or string boundary

```
MCM    ADR, BDR, V
```

```
MCM    ADR, BDR
```

```
MCM    ADR
```

```
MCM
```

This instruction is used to transfer a string specified by the ADR into a string specified by the BDR. If a variant is specified, the transfer finishes when the SPM has found a same character as the variant in the ADR string. The character itself is not transferred to, but the APR contains its position. If the match of a ADR character and the variant is unsuccessful or no variant is specified, the transfer finishes at the boundary of the ADR or BDR string.

SAR: Store A-address register

```
SAR    ADR
```

This instruction is used to store the contents of the A-address register of the previous instruction into a numerical label specified by the ADR. This instruction is usually used to store a return address when a branching instruction is executed.

SAP: Store A-address character position

```
SAP    ADR
```

This instruction is used to store the contents of the A-address character pointer register APR into a numeric label specified by the ADR.

SRCH: Search a string ignoring variant

```
SRCH    ADR, BDR, V
```

```
SRCH    ADR, BDR
```

```
SRCH    ADR
```

```
SRCH
```

This instruction is used to find a string specified by the BDR from a string specified by the ADR. If no variant is specified, no character of the ADR is ignored in the search operation. Characters of the ADR string are compared with characters of the BDR string one by one. The SPM stops the comparison when unequal characters are found and it sets an appropriate value in the indicator.

ERASE: Erase a string

```
ERASE    ADR
```

This instruction is used to erase unnecessary strings of dynamic labels. The erased memories are linked to the free storage space for later use.

MOD: Absolute modulo of BDR

```
MOD    ADR, BDR, V
```

```
MOD    ADR, BDR
```

By the execution of this instruction, one or two FORTRAN words of a string specified by the ADR are divided by a numeric value specified by the BDR. The absolute value of the residue is stored in the numeric string of the BDR. If 1 specified in the variant, one FORTRAN word of the ADR string is divided and if 2 is specified in the variant, two FORTRAN words are divided by the numeric value of the BDR. If no variant is specified, the string of the ADR is also assumed as of a numeric label and the division occurs. Using dynamic labels and this instruction, one can utilize the scatter storage techniques to store and retrieve strings [3], [4].

TON: Trace on

```
TON
```

This instruction is used to print out the contents of 26 registers of the SPM. For each instruction execution, the SPM print out the 26 registers and it continues the print until it executes a TOFF (Trace off) instruction.

READ: Read a string

```
READ    ADR, BDR, V
```

```
READ    ADR, BDR
```

This instruction is used to read,from an unit V, characters of specified length by the BDR into a string of the ADR. The unit number V is corresponding to the FORTRAN logical unit number from 1 to 12. No variant specification means that the V is the system input unit.

## 3. *Experience with the SPM*

Several conversion programs are written using the SPM to convert programs of FORTRAN II to these of FORTRAN IV. In the execution of the conversion programs, it has been observed that they are 2 to 5 times faster than the other previous conversion programs. The SPM conversion programs of the IBM 7090 or 7044 computer can convert FORTRAN II programs of 600-900 statements in one minute.

The SPM itself consists of 4000 FORTRAN IV statements (excluding comment

cards) and it requires 20 K FORTRAN words to load into the main memory. It can accept EBCDIC or BCD character code system and by changing a code table it will accept any type of code system. The code system and number of bytes in a FORTRAN word can be specified on the SPM control card.

The SPM is tested by the IBM 7044, IBM SYSTEM 360/75, GE 635 and its latest version is now under operation on the FACOM 230-60 computer at the JAERI computing center.

*References*

[1] Asai, K., Problems on FORTRAN Conversion Programs, *Joho-Shori* (*The Journal of the Information Processing Society of Japan*), *10*, 4, 235–239. (in Japanese)

[2] Asai, K., N. Saito and Y. Inami, The SPM—A FORTRAN Program for String Processing, *JAERI memo 3595*, Japan Atomic Research Institute, (June, 1969)

[3] Maurer, W. D., An Improved Hash Code for Scatter Storage, *The Comm. of the ACM*, (Jan., 1968), 35–38.

[4] Morris, R., Scatter Storage Techniques, *The Comm. of the ACM*, (Jan., 1968), 38–44.