R-17

SPS TIME-SHARING STRING PROCESSING SYSTEM

REFERENCE MANUAL

Butler W. Lampson

L. Peter Deutsch

Larry L. Barnes

University of California, Berkeley

Document No. 30.10.20

Issued April 21, 1965

Revised July 27, 1966

Contract SD-185

Office of Secretary of Defense

Advanced Research Projects Agency

Washington 25, D.C.

TABLE OF CONTENTS

1.0 General

The String Processing System (SPS) consists of eight SYSPOPs and six system subroutines (BRS instructions). SPS strings are stored three 8-bit characters per word. Strings are addressed by two-word pointers. The first word contains the character address of the character before the first character of the string. The second word contains the character address of the last character of the string. The character address of a character is obtained by multiplying by 3 the address of the word containing it and adding 0, 1 or 2 depending on its position in the word. All string pointers contain character addresses. The character pointers used by GCI, GCD, WCI and WCH must have the first 8 bits cleared.

The user's attention is called to the string package (SP) which is a collection of POPs and subroutines for string processing which can be obtained from a system file and included in the user's program. This package affords the user a number of additional facilities and considerably greater flexibility than the resident routines described in this manual.

The following SYSPOPs are independent of the hash table mechanism which is described later. Any of them may be indexed or indirectly addressed (as may most other SYSPOPs).

1.1 String Pointer Load and Store Operations

```
LDP     OPD     16600000B,1,1,0,1
```

LDP ADDR loads the A and B registers with the contents of ADDR and ADDR+1. X is undisturbed.

```
STP     OPD     16700000B,1,1,0,1
```

STP ADDR stores the contents of the A and B registers in locations ADDR and ADDR+1. A, B, and X are undistrubed.

1.2 String Read and Write Operations

```
GCI     OPD     16500000B,1,1,0,1
```

GCI ADDR tries to load the A register with the first character of the string addressed by the pointer pair in ADDR and ADDR+1. If the string is null or empty (i.e., if the contents of ADDR is greater than or equal to the contents of ADDR+1), then nothing is done and the next instruction in sequence is executed. If the string is not null, its first character is loaded into A right-justified and the contents of ADDR are incremented by 1, so that the string pointer now points to the string with the first character deleted. The next instruction in sequence is skipped. Unless a copy of the original pointer is saved, the contents of the string are effectively destroyed by GCI. For example, the code:

```
    GCI     STRING
    BRU     ØUT
    BRM     PRØCESS
    BRU     * - 3
ØUT ...
```

will call the subroutine PRØCESS with each character of the string addressed by STRING and go to ØUT after the last character is processed. To save the contents of STRING, the following commands could have been executed first:

```
LDP     STRING
STP     SAVE
etc.
```

The X register is not disturbed by GCI. The B register is destroyed. Timing: 43 cycles.

```
        GCD     OPD     13700000B,1,1,0,1
```

GCD is in every way similar to GCI except that the character is taken from the end of the specified string and the second string pointer is decremented.

```
        WCI     OPD     15700000B,1,1,0,1
```

WCI ADDR writes the character in A on the end of the string addressed by ADDR. The contents of ADDR+1 are incremented by 1. A and X are not changed. B is destroyed.

To use a WCI in constructing a string, it is necessary to start with a null string. Suppose the string is to be put into a buffer called LINE and defined by

```
        LINE     BSS     20
```

The instructions

```
        LDA     =LINE
        MUL     =3
        LSH     23
        STA     PTR
        STA     PTR+1
```

will make PTR a pointer to a null string beginning (and ending) with the first character (not the 0th) in LINE. To start with the 0th character a SUB =1 could be inserted after the LSH. LINE can now be filled, say from the teletype by

```
        TCI     CHAR
        WCI     PTR
        BRU     * - 2
```

```
WCH     OPD     15700000B,1,1,0,1
```

Takes a character in A and a table address in the operand field.

The table comprises three words:

```
ZRO   CLB

ZRO   CUB

OP    ADD
```

WCH tries to write a character into the area defined by the character addresses CLB, CUB. Provided that CUB>CLB, WCH will write the character in A into character position CLB+1 and increment CLB. If CLB≥CUB the character is not written and control is transferred to the third word of the table with A, X undisturbed and the address of the offending WCH in B. This can be an error trap or an exit to a routine which allocates more memory, by garbage collection or otherwise, for successive WCH's.

1.3 String Compare Operations

```
SKSE     OPD     16300000B,1,1,0,1
```

SKSE ADDR skips if the string addressed by the pointer in AB is identical with the string addressed by ADDR. If the strings are of different lengths or have different contents, SKSE does not skip. This instruction is essentially identical to SKE, except that it acts on strings rather than numbers. A, B, X are not disturbed by SKSE.

```
SKSG     OPD     16200000B,1,1,0,1
```

SKSG ADDR skips if the contents of the string addressed by AB is greater than the contents of the string addressed by ADDR and ADDR+1. Comparison is made character by character, and terminates with the first unequal characters; the numerical, internal code representation of characters is used to determine inequality. If the strings are equal for the entire length of the shorter one, the longer one is indicated as the greater. A, B and X are not disturbed by SKSG.

1.4 String Input

BRS 33

Accepts a string pointer address in A, a file number in X and a "terminal character" in B. It collects characters from the file and appends them to the string until the terminal character is seen; this is not added to the string. It then returns the updated string pointer in AB; the string pointer in core is also updated. If bit 0 of A is set on entry the string is taken as null with the second pointer equal to the first.

1.5 String Output

BRS 34

Accepts a file number in X, a word address in A and a count in B. It outputs B consecutive characters starting with the first character of the specified word. If B=-1 on entry characters are output until / is encountered; the character $ is interpreted as carriage return, line feed.

BRS 35

Accepts a file number in X and a string pointer in AB. It outputs the string to the file.

2.0 String Manipulation via a Hash Table

The hash table is a structure for minimizing the effort required to perform certain scan-and-compare operations when the operands are strings.

A hash table is a contiguous set of 3-word "augmented string pointers". The addresses of the first and last-plus-one locations of the hash table we shall denote by HT, EHT respectively. Each augmented string pointer occupies three consecutive locations of the hash table. Bits 8 to 23 of each of the first two locations hold the actual string pointer; bits 0 to 7 of these two words, as well as the entire third word (the so-called string "value") may hold arbitrary information. Note, however, that bits 0 to 7 of the string pointer words must be zero if used with GCI or WCI.

2.1 The Hash Structure System Subroutines

There are three system subroutines to perform operations on a hash structure; they are BRS 5, BRS 6, BRS 37. BRS 6 is used to introduce new string pointers into the structure; the strings will normally have been created by WCI or WCH. BRS 5 and BRS 37 each perform a scan of the hash table for a string to match a given string.

Before using BRS 5 and BRS 6 to insert string pointers into an initially empty hash table, the hash table area must be cleared to zeros.

BRS 5

Takes a string pointer in A, B, a table address in X. The table comprises 3 words

```
ZRO  HT
ZRO  EHT
ZRO  0
```

The first two define the hash table bounds, the third is used for communication with BRS 6 (q.v.).

BRS 5 searches the hash table for a string to match the given one. If successful it returns in B the address of the hash table string pointer (the string "index") -- and in A the string "value"; it skips on return. If the search is unsuccessful, BRS 5 returns with A, B unchanged and the address of the next free table entry in word 3 of the table (this will be -1 if the table is full). X is not disturbed.

BRS 6

Takes a string pointer in A,B and a table address in X. The table is as for BRS 5.

BRS 6 inserts the string pointer into the hash table at the point determined by the last BRS 5 which failed. If the table was then found to be full, and the "communication word" (third word of the table) is -1, there is an illegal instruction trap. BRS 6 is intended for use only in inserting into the hash table a string pointer for which BRS 5 failed to find a match and should not be used other than after a failing BRS 5. Furthermore, string pointers should not be placed in the hash table other than with BRS 6 (otherwise the scanning algorithm used in BRS 5, BRS 37 will not work). Note that BRS 6 does not physically move the characters to which (AB) points.

On exit, BRS 6 returns in B the address of the first word of the new hash table entry and in A, the "value" word of the entry; X is not disturbed. To delete a hash table entry, put -1 (not 0) in the first word.

BRS 37

Takes a dual file number in A, a string pointer address in B and, in X, the address of a 2-word table containing hash table bounds HT, EHT. A dual file number is a single word holding an output file number in the first 12 bits and an input file number in the second. If the output file number is zero, the user's teletype will be used. The behavior of BRS 37 depends on the command recognition mode currently set for the user's Exec (see the TSS Exec Manual, Section 5.5).

If the mode is BEGINNER, the hash table is scanned for a string to match exactly the given one. If none is found but the given string matches the initial part of some hash table string, characters from the input file are appended to it until <u>either</u> an exact match is obtained <u>or</u> a match becomes impossible. The exit is described below.

If the mode is NOVICE, the hash table is scanned for a string to match the given one. If none is found but the given string matches the initial part of some hash table string, characters from the input file are appended until the string is long enough <u>either</u> to determine a unique hash table string, with a matching initial part, or for no match to be possible. In the former case, if the hash table string now contains 3 or less as-yet-unmatched characters, more characters are taken from input until an exact match is obtained or no match is possible; if the hash table string contains 4 or more as-yet-unmatched characters these unmatched characters are sent to the O-P file. If the input file is the teletype, BRS 37 waits until all the characters have been output, and the input file buffer is cleared before exit.

If the mode is EXPERT the hash table is scanned for a string to match the given one. If none is found but the given string matches the initial part of some hash table string, characters from the input file are appended until the string is long enough <u>either</u> to determine a unique hash table string, with a matching initial part, <u>or</u> for no match to be possible. In the former case the remaining characters of the hash table string are sent to the O-P file.

Exits are as follows:

The no-match condition causes a no-skip exit with a string pointer in AB to the string so far collected; X is undisturbed. If a match is found there is a skip exit with the hash table string index in A and the string value in B; X is undisturbed.

2.2 <u>Example</u>

The following subroutine illustrates a use of the hash table facility. A string is input from the teletype and appended to WCH string storage until a carriage return is encountered; it is assumed that string storage does not overflow in the process. The hash table is then searched for the string; if it is not already there it is inserted. In any case, an exit is made with the value of the string in A and the address of the string pointer in B. On entry X contains the address of the table for BRS 5, 6. CTL is the address of a table for WCH.

```
INPUT   ZRO     INPL
        LDA     CTL      remember beginning of string
        STA     TEMP
LOOP    TCI     CHAR
        SKE     =155B    terminator?
        BRU     WRITE
        LDA     TEMP     yes
        LDB     CTL
        BRS     5
        BRS     6
        SBRR    INPUT
WRITE   WCH     CTL
        BRU     LOOP
```