

Additions to the SAM76 Language collection

Additions to the SAM76 Language collection
--

A	Additions
	1. Channel Control
	2. Texts containing executable code
B	Bestiary: A book of beasts
C	Characters, Codes and Conventions
D	Description of the SAM76 language
E	Examples and algorithms
F	Function definitions
G	Glossary
H	h1
I	i1
J	j1
K	k1
L *	Loading SAM76 systems
M *	Machine dependent programs
N *	Notices and updates
O	o1
P *	p1
Q	q1
R *	Reprints
S *	Scripts and samples
T	t1
U *	User linkages to the SAM76 system
V	v1
W	w1
X	X-perimental functions
Y	y1
Z	z1

Section List

*

Note: Asterisk signifies section not furnished with book but reserved for user filing of material that may be provided with particular system releases or published from time to time.

Initialization of Channel Control

A number of the SAM76 language functions are concerned with accepting or delivering data between the SAM76 system and the outside world. For convenience two communication groups are defined and the initial assignments are as listed below:

Group 1	This group of functions is normally initialized to interact with the user console {as vectored by CI CS and CO}. These functions are: "is" input string, "os" output string, "ic" input character, "id" input d characters, "im" input to matching string, "vt" view text, "tm" trace mode.
---------	---

Group 2	This group of functions is normally initialized to interact with the so called "Reader" and "Punch" channels {as vectored by RI and PO}. Functions in this group are "it" input text, "idt" input d texts, "ot" output text, "xu,xmt" xperimental user transmit, and "xu,rcv" xperimental user receive.
---------	---

Means are provided within the SAM76 language interpreter to reassociate these two groups with other devices than those normally assigned. This is independent of the assignments that may be made using the "IOBYTE" if available; note that the assignments will however be made within the general framework of the IOBYTE setting as modified through use of the "sio" set IO function.

The functions in the channel control family include:

lic,s0	List Input Channels
loc,s0	List Output Channels
sic,sym	Select Input Channel
soc,sym	Select Output Channel

Caution!	The user must be careful to construct procedures in such a manner that the system will not be "hung" in a mode that is difficult to recover from. Also judicious use of the "nud", and "sem" functions should be considered.
----------	--

Texts containing executable code

Version 23 of the SAM76 language interpreter was the first version to allow the user to create and place in the text area modules of executable object code - obviously executable only in a particular machine. The SAM76 interpreter is able to differentiate between texts created using the "dt" define text function and machine code through the use of the "protection class" functions.

When the protection class of a particular text in the text area is set to be HEX 80, either using the "cpc" change protection class function, or during the creation of this type of text, then the usual text manipulation functions have no effect on that text. Furthermore any type of fetch, that is to say explicit or implied, as well as active or neutral will cause the interpreter to execute a call to the first location after the text name which is embedded in the text itself.

In the special case of 8080/Z80 class machines, the H & L registers contain the actual address of that particular point so that the user program may make use of this information as required to modify or relocate addresses in the object code of the "text". Additionally the zero/non zero flag bit is set if the function was invoked neutrally (non zero), and the A register contains a zero. The user program in the text is then executed (safely, it is hoped,) and a return to the SAM76 language system is achieved by doing any type of "return" instruction; that is only true if the user program handled the stack properly - if this is doubtful the user may always achieve a safe return to the SAM76 scanner by jumping to location "BEGIN+6" or to the appropriate point in the system subroutine vector table.

It is obvious that the variability of texts in the text area requires that any object code to be executed using this system be written carefully to be always position independent, either through appropriate use of position independent instructions, or through relocation program provided by the user.

An assembly language program designed to provide required pointers and to correctly organize a unit that may reside in the text area is shown in figure 1. This program may be assembled using some appropriate macro assembler and injected into the text area. Another way of creating an executable object code text is to use the "xu,dt" Xperimental User Define Text function.

Preliminaries ↓

```

10      .TITLE /Sample Program/
11      ;
12      0000  Z80=0
13      0001  RSTINT=1
14      ;
15      0100  $COM=000400      ; ^h100
16      ;
20      $$SWA=^D3
21      $$SER=^D5
22      .DEFINE LINKC(ARG)=
23      [CALL $COM+3*ARG]
24      ;
25      .DEFINE TSTA=
26      [ORA A]
27      ;
41      ;***** Pointer to Text Area End
42      ;
43      349E  TALsym= 032236 ; ^h349E CP/M
44      ;
45      ;
46      ;***** Arbitrary Load Point
47      ;
48      4000  ENTLOC= 040000 ; ^h4000
49      ;
50      ;
51      ;user program specials
52      ;
53      .DEFINE .DIMEN(ARG)=
54      [ .WORD [[ARG]<^D81&177400]
55      ! [[ARG]>^D81&377]]
56      ;
57      ;***** Origin statement for Loading
58      ;
59      0000  0      .LOC ENTLOC
60      ;
61      ;***** Once only to move in place
62      ;
63      4000  21 9E 34      LXI H,TALsym;as wanted
64      4003  E5      PUSH H
65      4004  5E      MOV E,M
66      4005  23      INX H
67      4006  56      MOV D,M
68      4007  2A 20 40      LHLD TRPNT
69      400A  4D      MOV C,L
70      400B  44      MOV B,H
71      400C  21 52 40      LXI H,LOCTOP
72      ;
73      400F  7E      MOVE: MOV A,M
74      4010  12      STAX D
75      4011  2B      DCX H
76      4012  1B      DCX D
77      4013  0B      DCX B
78      4014  79      MOV A,C
79      4015  B0      ORA B
80      4016  C2 0F 40      JNZ MOVE
81      4019  E1      POP H
82      401A  73      MOV M,E
83      401B  23      INX H
84      401C  72      MOV M,D
85      401D  C3 00 00      JMP 0; or as desired
86      ;

```

```

87      ;
88      ;text position independent code
89      ;
90      ;***** REVERSE POINTER and NAME
91      ;
92      4020  00 32      TRPNT: .DIMEN LOCPFN+2 - .
93      4022  50      TNAME: .ASCII PROGRAM
94      4023  52
95      4024  4F
96      4025  47
97      4026  52
98      4027  41
99      4028  4D
100     ;
101     ;***** OBJECT CODE SECTION
102     ;normal entry point
103     4029      LOGGOT:
104     ;
105     ;address of "locgot" in H & L
106     ;
107     ;
108     ;***** Position Independence
109     ;
110     4029  E5      STRT: PUSH H
111     402A  01 19 00      LXI B,TABLE-LOGGOT
112     402D  09      DAD B
113     402E  EB      XCHG
114     402F  E1      POP H
115     4030  01 0E 00      LXI B,LOOP-STRT
116     4033  09      DAD B
117     ;
118     ;
119     ;***** SAM76 Housekeeping
120     ;
121     4034  CD 0F 01      LINKC $$SER
122     ;
123     ;***** Program Loop
124     ;
125     4037  EB      LOOP: XCHG
126     4038  7E      MOV A,M
127     4039  B7      TSTA
128     403A  C8      JRQZ
129     403B  4F      MOV C,A
130     403C  CD 09 01      LINKC $$SWA
131     403F  23      INX H
132     4040  EB      XCHG
133     4041  E9      PCHL
134     ;
135     ;
136     4042  3C      TABLE: .BYTE 74
137     4043  70      .ASCII program
138     4044  72
139     4045  6F
140     4046  67
141     4047  72
142     4048  61
143     4049  6D
144     404A  3E      .BYTE 76
145     404B  00      .BYTE 0
146     ;
147     ;
148     ;***** Text Header Information
149     ;
150     404C  00 00      LOCPFI: .WORD 0
151     404E  00 23      LOCPFF: .DIMEN LOCPFI-LOGGOT
152     4050  07      LOCPFN: .BYTE LOGGOT-TNAME
153     4051  80      .BYTE 200 ; ^h80
154     ;
155     ;
156     4052      LOCTOP:
157     ;
158     ;
159     4052  0      .END ;editor added

```

[illegible]

1998, 1999, 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017, 2018, 2019, 2020, 2021, 2022, 2023, 2024, 2025, 2026, 2027, 2028, 2029, 2030, 2031, 2032, 2033, 2034, 2035, 2036, 2037, 2038, 2039, 2040, 2041, 2042, 2043, 2044, 2045, 2046, 2047, 2048, 2049, 2050, 2051, 2052, 2053, 2054, 2055, 2056, 2057, 2058, 2059, 2060, 2061, 2062, 2063, 2064, 2065, 2066, 2067, 2068, 2069, 2070, 2071, 2072, 2073, 2074, 2075, 2076, 2077, 2078, 2079, 2080, 2081, 2082, 2083, 2084, 2085, 2086, 2087, 2088, 2089, 2090, 2091, 2092, 2093, 2094, 2095, 2096, 2097, 2098, 2099, 2100, 2101, 2102, 2103, 2104, 2105, 2106, 2107, 2108, 2109, 2110, 2111, 2112, 2113, 2114, 2115, 2116, 2117, 2118, 2119, 2120, 2121, 2122, 2123, 2124, 2125, 2126, 2127, 2128, 2129, 2130, 2131, 2132, 2133, 2134, 2135, 2136, 2137, 2138, 2139, 2140, 2141, 2142, 2143, 2144, 2145, 2146, 2147, 2148, 2149, 2150, 2151, 2152, 2153, 2154, 2155, 2156, 2157, 2158, 2159, 2160, 2161, 2162, 2163, 2164, 2165, 2166, 2167, 2168, 2169, 2170, 2171, 2172, 2173, 2174, 2175, 2176, 2177, 2178, 2179, 2180, 2181, 2182, 2183, 2184, 2185, 2186, 2187, 2188, 2189, 2190, 2191, 2192, 2193, 2194, 2195, 2196, 2197, 2198, 2199, 2200, 2201, 2202, 2203, 2204, 2205, 2206, 2207, 2208, 2209, 2210, 2211, 2212, 2213, 2214, 2215, 2216, 2217, 2218, 2219, 2220, 2221, 2222, 2223, 2224, 2225, 2226, 2227, 2228, 2229, 2230, 2231, 2232, 2233, 2234, 2235, 2236, 2237, 2238, 2239, 2240, 2241, 2242, 2243, 2244, 2245, 2246, 2247, 2248, 2249, 2250, 2251, 2252, 2253, 2254, 2255, 2256, 2257, 2258, 2259, 2260, 2261, 2262, 2263, 2264, 2265, 2266, 2267, 2268, 2269, 2270, 2271, 2272, 2273, 2274, 2275, 2276, 2277, 2278, 2279, 2280, 2281, 2282, 2283, 2284, 2285, 2286, 2287, 2288, 2289, 2290, 2291, 2292, 2293, 2294, 2295, 2296, 2297, 2298, 2299, 2300, 2301, 2302, 2303, 2304, 2305, 2306, 2307, 2308, 2309, 2310, 2311, 2312, 2313, 2314, 2315, 2316, 2317, 2318, 2319, 2320, 2321, 2322, 2323, 2324, 2325, 2326, 2327, 2328, 2329, 2330, 2331, 2332, 2333, 2334, 2335, 2336, 2337, 2338, 2339, 2340, 2341, 2342, 2343, 2344, 2345, 2346, 2347, 2348, 2349, 2350, 2351, 2352, 2353, 2354, 2355, 2356, 2357, 2358, 2359, 2360, 2361, 2362, 2363, 2364, 2365, 2366, 2367, 2368, 2369, 2370, 2371, 2372, 2373, 2374, 2375, 2376, 2377, 2378, 2379, 2380, 2381, 2382, 2383, 2384, 2385, 2386, 2387, 2388, 2389, 2390, 2391, 2392, 2393, 2394, 2395, 2396, 2397, 2398, 2399, 2400, 2401, 2402, 2403, 2404, 2405, 2406, 2407, 2408, 2409, 2410, 2411, 2412, 2413, 2414, 2415, 2416, 2417, 2418, 2419, 2420, 2421, 2422, 2423, 2424, 2425, 2426, 2427, 2428, 2429, 2430, 2431, 2432, 2433, 2434, 2435, 2436, 2437, 2438, 2439, 2440, 2441, 2442, 2443, 2444, 2445, 2446, 2447, 2448, 2449, 2450, 2451, 2452, 2453, 2454, 2455, 2456, 2457, 2458, 2459, 2460, 2461, 2462, 2463, 2464, 2465, 2466, 2467, 2468, 2469, 2470, 2471, 2472, 2473, 2474, 2475, 2476, 2477, 2478, 2479, 2480, 2481, 2482, 2483, 2484, 2485, 2486, 2487, 2488, 2489, 2490, 2491, 2492, 2493, 2494, 2495, 2496, 2497, 2498, 2499, 2500, 2501, 2502, 2503, 2504, 2505, 2506, 2507, 2508, 2509, 2510, 2511, 2512, 2513, 2514, 2515, 2516, 2517, 2518, 2519, 2520, 2521, 2522, 2523, 2524, 2525, 2526, 2527, 2528, 2529, 2530, 2531, 2532, 2533, 2534, 2535, 2536, 2537, 2538, 2539, 2540, 2541, 2542, 2543, 2544, 2545, 2546, 2547, 2548, 2549, 2550, 2551, 2552, 2553, 2554, 2555, 2556, 2557, 2558, 2559, 2560, 2561, 2562, 2563, 2564, 2565, 2566, 2567, 2568, 2569, 2570, 2571, 2572, 2573, 2574, 2575, 2576, 2577, 2578, 2579, 2580, 2581, 2582, 2583, 2584, 2585, 2586, 2587, 2588, 2589, 2590, 2591, 2592, 2593, 2594, 2595, 2596, 2597, 2598, 2599, 2600, 2601, 2602, 2603, 2604, 2605, 2606, 2607, 2608, 2609, 2610, 2611, 2612, 2613, 2614, 2615, 2616, 2617, 2618, 2619, 2620, 2621, 2622, 2623, 2624, 2625, 2626, 2627, 2628, 2629, 2630, 2631, 2632, 2633, 2634, 2635, 2636, 2637, 2638, 2639, 2640, 2641, 2642, 2643, 2644, 2645, 2646, 2647, 2648, 2649, 2650, 2651, 2652, 2653, 2654, 2655, 2656, 2657, 2658, 2659, 2660, 2661, 2662, 2663, 2664, 2665, 2666, 2667, 2668, 2669, 2670, 2671, 2672, 2673, 2674, 2675, 2676, 2677, 2678, 2679, 26

[illegible]

00000001	00000001
00000002	00000002
00000003	00000003
00000004	00000004
00000005	00000005
00000006	00000006
00000007	00000007
00000008	00000008
00000009	00000009
00000010	00000010
00000011	00000011
00000012	00000012
00000013	00000013
00000014	00000014
00000015	00000015
00000016	00000016
00000017	00000017
00000018	00000018
00000019	00000019
00000020	00000020
00000021	00000021
00000022	00000022
00000023	00000023
00000024	00000024
00000025	00000025
00000026	00000026
00000027	00000027
00000028	00000028
00000029	00000029
00000030	00000030
00000031	00000031
00000032	00000032
00000033	00000033
00000034	00000034
00000035	00000035
00000036	00000036
00000037	00000037
00000038	00000038
00000039	00000039
00000040	00000040
00000041	00000041
00000042	00000042
00000043	00000043
00000044	00000044
00000045	00000045
00000046	00000046
00000047	00000047
00000048	00000048
00000049	00000049
00000050	00000050
00000051	00000051
00000052	00000052
00000053	00000053
00000054	00000054
00000055	00000055
00000056	00000056
00000057	00000057
00000058	00000058
00000059	00000059
00000060	00000060
00000061	00000061
00000062	00000062
00000063	00000063
00000064	00000064
00000065	00000065
00000066	00000066
00000067	00000067
00000068	00000068
00000069	00000069
00000070	00000070
00000071	00000071
00000072	00000072
00000073	00000073
00000074	00000074
00000075	00000075
00000076	00000076
00000077	00000077
00000078	00000078
00000079	00000079
00000080	00000080
00000081	00000081
00000082	00000082
00000083	00000083
00000084	00000084
00000085	00000085
00000086	00000086
00000087	00000087
00000088	00000088
00000089	00000089
00000090	00000090
00000091	00000091
00000092	00000092
00000093	00000093
00000094	00000094
00000095	00000095
00000096	00000096
00000097	00000097
00000098	00000098
00000099	00000099
00000100	00000100

[illegible][illegible][illegible]

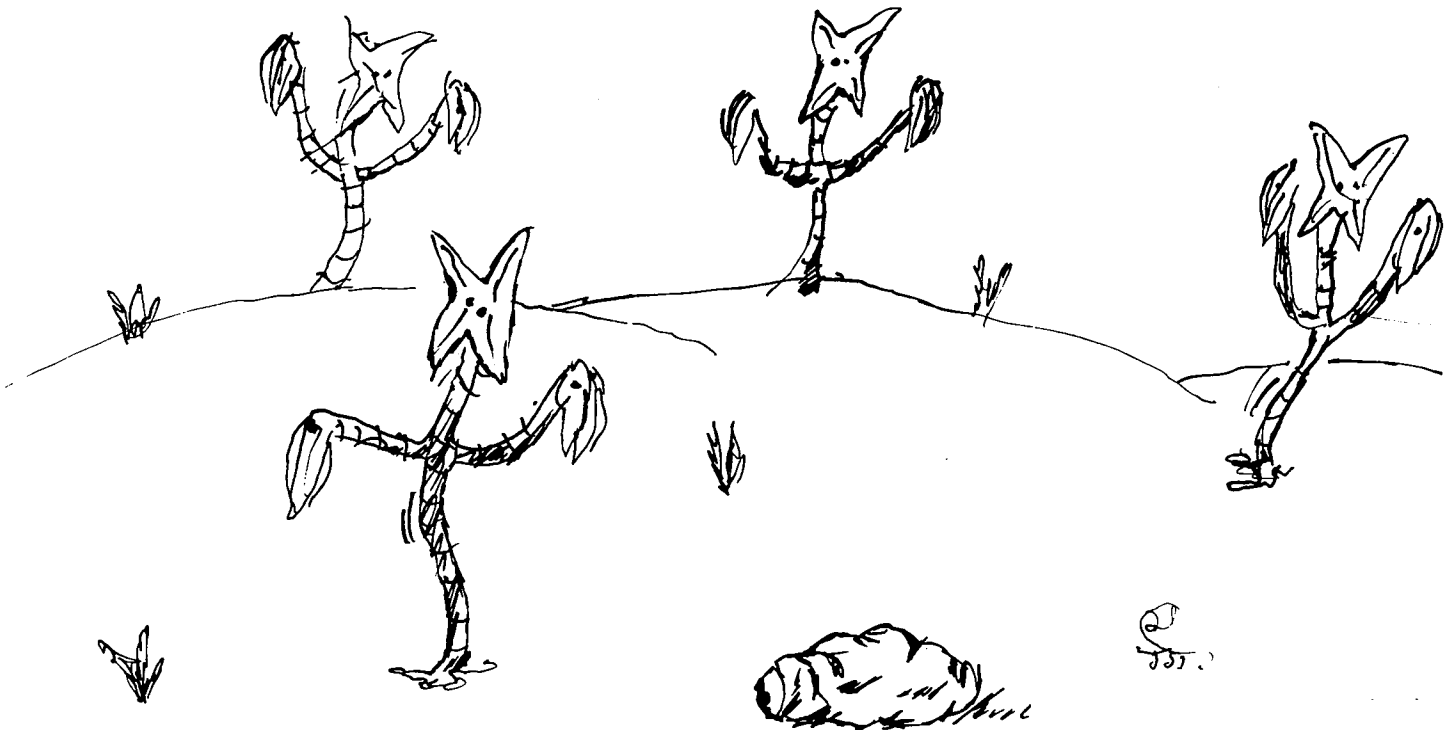
Bestiary - a SAM76 book of beasts

The Land of SAM

This is a high, flying satellite reconnaissance photograph of a distant land (called SAM by its inhabitants) with the residents in their natural habitat. This country is populated by crystoloids* who each have a specific function. Several people have just returned from this land and are now recuperating from terrible wounds incurred during their visit. Inside is a description of each beast's duties and their names given by the authors in both the vernacular and scientific. One cannot appreciate fully the horrible monsters the authors saw during their sojourn because all sketches were made on the run, but if you let your imagination go, you may begin to have as bad nightmares as we did and still do.

*c.f.

John W. Campbell, Crystoloid vs. Colloid man, talk given at the second annual meeting of the Institute for Cybercultural Research, N.Y.C., 1962.



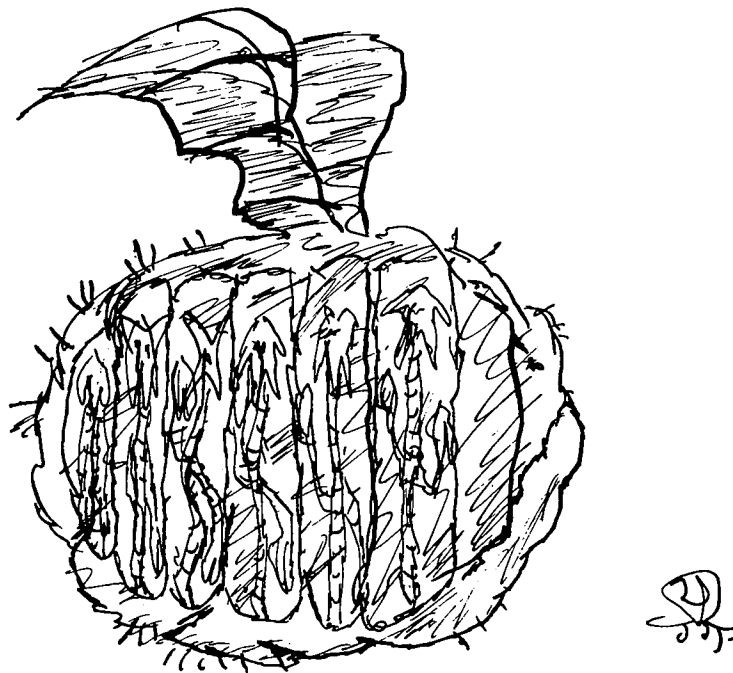
Tybox - (Skokie Tribe, to print)

The Tybox (output string) informs distant lands of the state of affairs in the land of the beasts. With his many feet he imprints on the white clouds what has been found in the remains of the Stringere after they have been devoured by Translatus Uggligr.



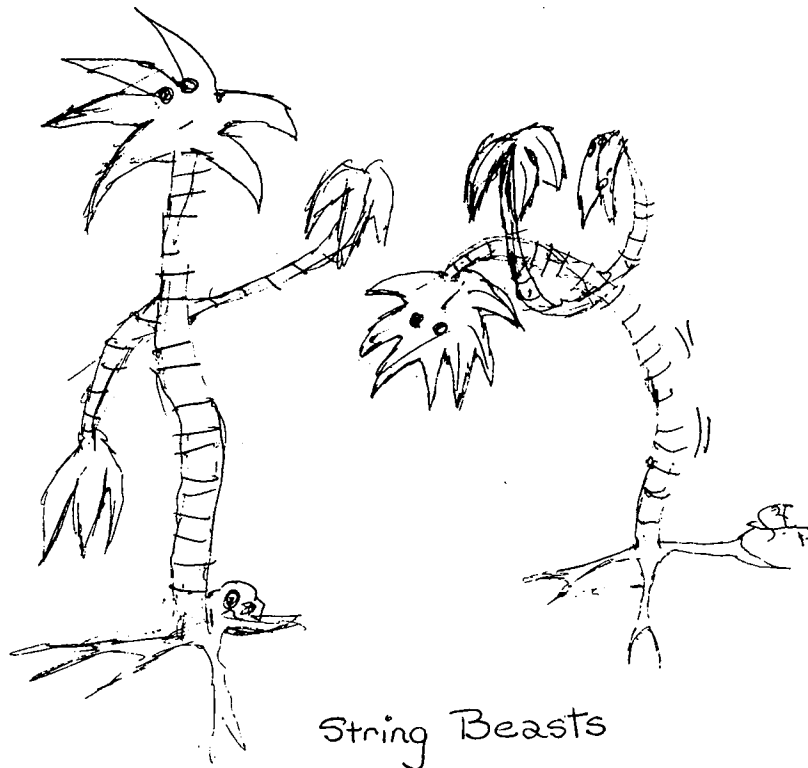
The Palma - (1. palm of the hand)

The Palma (text area) beast and the Stringere (string beasts) have a symbiotic relationship. The Stringere will reside in the stomach of the Palma only after they have been prepared for storage by the Translatus Uggliqr (eval beast) for his later request.



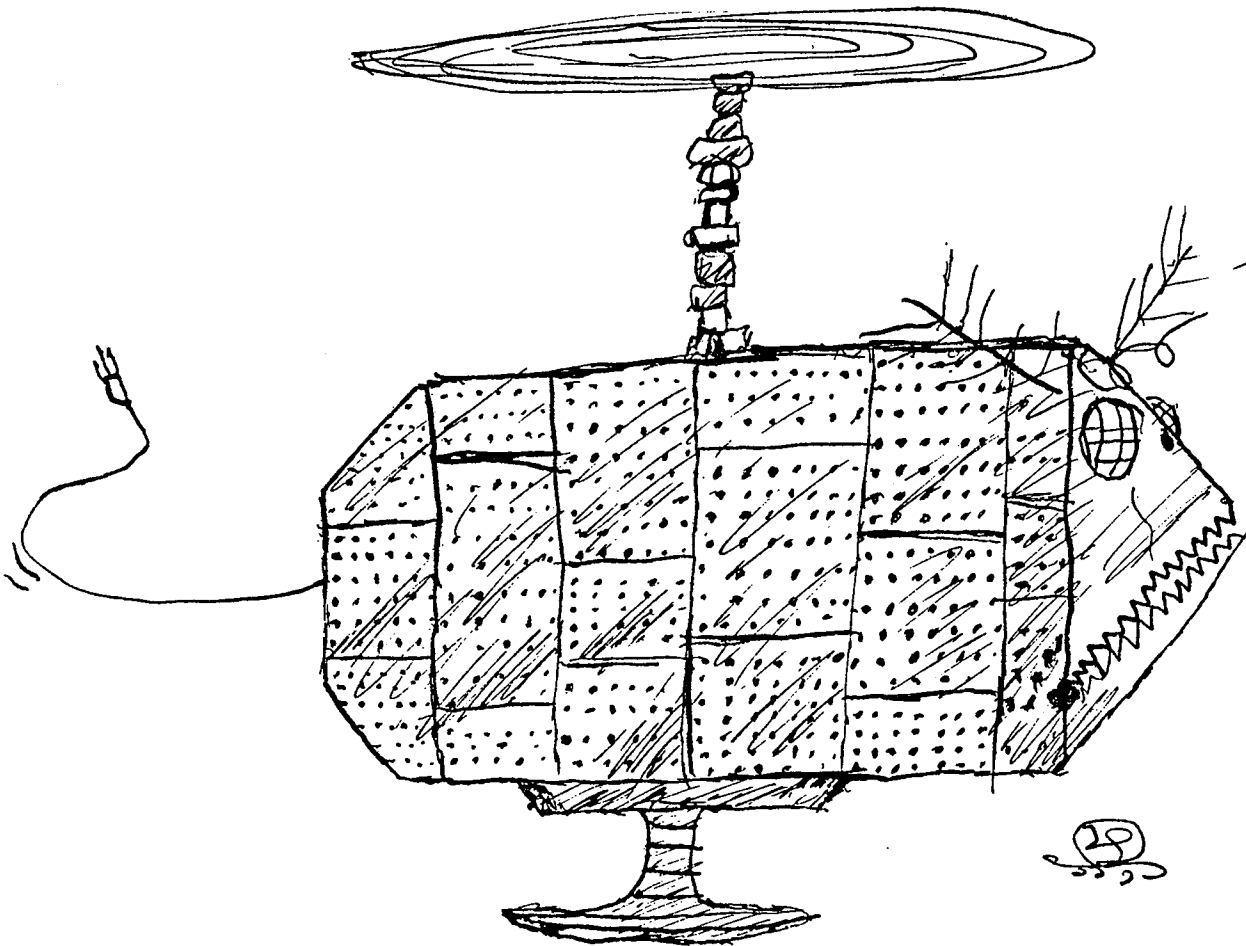
The Stringere - (1. to bind)

The Stringere (string beasts), a newly discovered family of nonvertebrae, are composed of anywhere from one to many cells. These beasts are quite abundant, but are only useful in particular cases where certain strains of cells are combined. Since these nonvertebrae are so simplistic it is truthful to assume that they are insignificant and depend upon all other beasts for their livelihood.



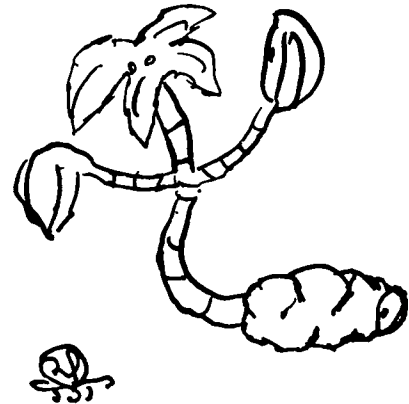
The Medha - (Skr. math)

The Medha (arithmetic-logical-mode) beast is the most magical of all the beasts. He digests the string beast strain Mathematica (Stringere Mathematica) for the Translatus Uggligr (eval beast), converting it to a simpler form. He can also change the Translatus Uggligr's reasoning process by perfecting chemicals that will change the beast's telepathic code. Unfortunately, he can only do this with the eval beast's permission.



The Translatus Uggligr - (1. translate On. Dreadful)
--

Translatus Uggligr arranges the pecking order. He is the sole authority on which beast grazes first. After he has devoured an active Stringere he summons by telepathic code the other beasts needed to help digest the Stringere and then excretes a simplified form of his meal, thereby distributing the seeds for another string beast.



The Raditi - (OSLAV read)

The Raditi (input string) beast is the eyes of the Translatus Uggligr (eval beast). It flies in distant lands searching for new species of Stringere. When it discovers a new species it relays by telepathy to the Translatus Uggligr the genetic structure of the Stringere. Translatus Uggligr then gives birth to an exact duplicate of the Stringere in the distant land.*

-

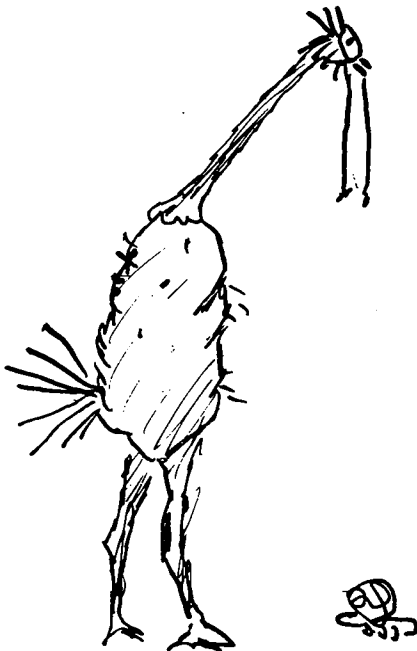
* The analogy "distant land" refers to keyboard input by the user.



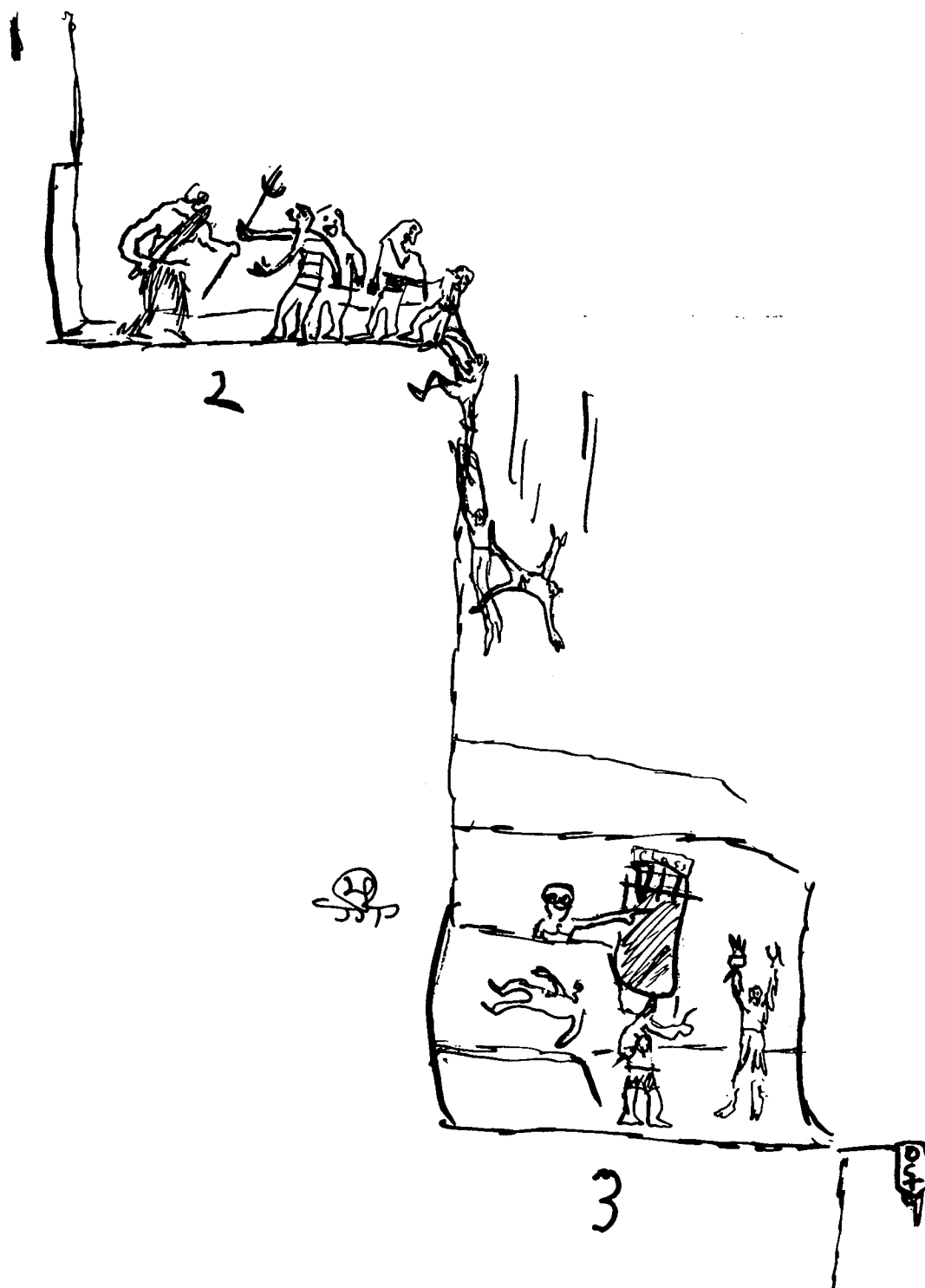
The Lugen - (G. look)

The Lugen (scanner) beast, a much misunderstood animal, gets its name from the action it performs for Translatus Uggligr. He points out the edible Stringere to the Translatus Uggligr for his consumption, avoiding the tasteless type.* The Lugen must select the most active edible Stringere first so Translatus Uggligr devours them in the proper order. If the order is not followed Translatus Uggligr goes on a rampage and becomes drastically ill.

* The tasteless type is any material between protection symbols.



[illegible]



Characters Codes and Conventions

Introduction

One of the problems currently encountered in the use of home computers, is the seemingly helter skelter assignment of symbols (both printing and non printing) for various software or system control functions.

The author has been involved for many years in the standardizing activity for communication and computer control coding, starting with the review of the first version of the "ASCII" code, now more properly known as "USASCII".

It is for this reason that the implementation of the SAM76 language (for the 8080 and Z80 microprocessors at least) has been done in such a manner as to conform in its use of symbols and control codes to U.S. national and international standards.

Defined in the standards are techniques for changing various modes of operation, and a multiplicity of optional system control functions; for obvious reasons these are not all included, and the options - permissible under the standards - most suitable for the personal computer user were applied in the SAM76 language implementation.

It is hoped that future systems and software created for the personal computer community will conform more closely to the standards than they have in the past.

Contrary to the general belief, the USASCII code is a true "eight" bit code, and not "seven" bits plus a "parity" bit; in the early application of the code various manufacturers of communications equipment found it convenient to use the then undefined set of symbols represented by the presence of the eighth bit, to furnish a means of checking the accuracy of transmission of each "seven" bit character.

USASCII

The proper way of viewing the USASCII code is as a "tableau" of 16 columns and 16 lines; this establishes 256 different code combinations for an eight bit character.

It is then appropriate to split this tableau into two halves, and thus the first eight columns of sixteen lines becomes the "seven" bit subset of the code that is familiar to most people.

The 128 code combinations within the "seven" bit subset are generally available on the so called "ASCII" keyboards, through the depression of the various keys, either singly or in combination with "shift" or "control" keys; the first two columns (that is to say columns 0 and 1) of the seven bit subset are specified to contain "control" codes, and these codes may be generated either by the use of single specially designated keys - "line feed", "tab", &c., or by the simultaneous depression of the "control" key and one of the keys which generates a character to be found in columns 4 and 5: thus the simultaneous depression of the control key and the letter "M" should generate the control code that does a "return" of a printing element (or cursor) to the beginning of the same line.

Most keyboards available today do not contain the means to generate the codes in the right hand half of the USASCII tableau; these keyboards are then defined to operate in a "seven bit" environment; keyboards that provide means to generate the full 256 codes are known to operate in an eight bit environment. This may be achieved either by providing an additional "shift" key which merely causes the eighth bit to be generated, as well as by providing additional keys which do this directly; typical of these might be the keys that control the movement of a "cursor" on a display device.

The standards provide the users of a "seven bit environment" device several means of accessing and using the full eight bit code structure; these make use of the "shift out - SO" and "shift in - SI" as well as the combination of the "escape - ESC" code with codes to be found in columns 2 through 5 of the "seven" bit subset.

Consequently it behooves the designer and implementer of both hardware and software to consider very carefully the implications of their designs with respect to the potential for user confusion should they decide for no reason, other than ignorance, to deviate from known and published ways of using the USASCII code.

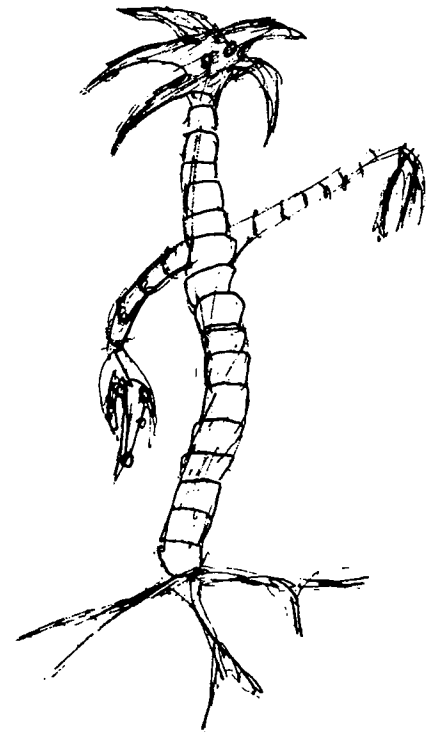
One frequently hears a designer ask "who needs these extra characters anyway?" when challenged as to his usage of the USASCII system; the most classical example of this is well known to most personal computer users - namely the ubiquitous model 33 teleprinter made by the Teletype Corporation; careful examination of the machine's design will reveal that the printing mechanism was designed from the beginning to be able to print both upper and lower case character sets and that the ability to print the lower case symbols was disabled by inhibiting the vertical travel of the printing cylinder the additional two levels required. Several people were actually startled at a National Bureau of Standards conference when they heard a Teletype Corporation Chief Engineer, now retired, ask in all sincerity the question as to need.

A present day example familiar to users of display control boards is exemplified by the Polymorphic and Processor Technology modules; these contain a character generator ROM capable of producing 32 symbols outside of those normally used - these may be the standardized pictorial representations for the USASCII control codes, or alternate alphabets such as "greek". Designers of these two units did not provide means for logically accessing these additional graphical symbols, and the means should be different for "control symbols" as for the "greek alphabet".

Another ubiquitous unit, namely the Lear Siegler ADM3 contains within its guts the capability of displaying 64 graphical symbols in addition to those normally seen by the proud owners of this fine machine; a trivial hardware modification, and the addition of an eight bit key will serve to make the owners prouder yet.

The "greek" character set should be viewed as a set of symbols in the right hand half of the full USASCII tableau, accessed either through the use of the "SO - SI" conventions, or through the use of a particular "ESC + " character sequence, or through the use of a keyboard which operates in the "eight" bit environment; note the need for software to make a reasonable match between keys struck and greek characters displayed.

The "pictorial control character" set should be accessed through the use of a software controlled switch operated by the proper "ESC + " character sequence.



The USASCII tableau - Seven Bit environment

Columns																
	0	1	2	3	4	5	6	7								
0	NUL	^@	DLE	^P	20..	b	30..	0	40..	@	DCS.	P	60..	`	70..	p
1	SOH	^A	DC1	^Q	21..	!	31..	1	41..	A	PU1.	Q	61..	a	71..	q
2	STX	^B	DC2	^R	22..	"	32..	2	42..	B	PU2.	R	62..	b	72..	r
3	ETX	^C	DC3	^S	23..	#	33..	3	43..	C	STS.	S	63..	c	73..	s
4	EOT	^D	DC4	^T	24..	\$	34..	4	INDfe	D	CCH.	T	64..	d	74..	t
5	ENQ	^E	NAK	^U	25..	%	35..	5	NELfe	E	MW..	U	65..	e	75..	u
6	ACK	^F	SYN	^V	26..	&	36..	6	SSA.	F	SPA.	V	66..	f	76..	v
7	BEL	^G	ETB	^W	27..	'	37..	7	ESA.	G	EPA.	W	67..	g	77..	w
8	BS.	^H	CAN	^X	28..	(38..	8	HTSfe	H	58..	X	68..	h	78..	x
9	HT.	^I	EM.	^Y	29..)	39..	9	HTJfe	I	59..	Y	69..	i	79..	y
A 10	NL.	^J	SUB	^Z	2A..	*	3A..	:	VTSfe	J	5A..	Z	6A..	j	7A..	z
B 11	VT.	^K	ESC	^[2B..	+	3B..	;	PLDfe	K	CSI.	[6B..	k	7B..	{
C 12	NP.	^L	FS.	^\	2C..	,	3C..	<	PLUfe	L	ST..	\	6C..	l	7C..	
D 13	RET	^M	GS.	^]	2D..	-	3D..	=	RI.fe	M	OSC.]	6D..	m	7D..	}
E 14	SO.	^N	RS.	^^	2E..	.	3E..	>	SS2.	N	PM..	^	6E..	n	7E..	~
F 15	SI.	^O	US.	^_	2F..	/	3F..	?	SS3.	O	APC.	_	6F..	o	DEL	d

In columns 0 and 1, ^X means "control X"

Functions shown in columns 4 & 5 achieved by ESC followed by character.

The USASCII tableau - Eight Bit environment extension

	Columns 8	9	10 A	11 B	12 C	13 D	14 E	15 F
0	80. ^@	DCS ^P	A0.. b	B0.. 0	C0.. @	D0.. P	E0.. `	F0.. p
1	81. ^A	PU1 ^Q	A1.. !	B1.. 1	C1.. A	D1.. Q	E1.. a	F1.. q
2	82. ^B	PU2 ^R	A2.. "	B2.. 2	C2.. B	D2.. R	E2.. b	F2.. r
3	83. ^C	STS ^S	A3.. #	B3.. 3	C3.. C	D3.. S	E3.. c	F3.. s
4	INDfe^D	CCH ^T	A4.. \$	B4.. 4	C4.. D	D4.. T	E4.. d	F4.. t
5	NELfe^E	MW. ^U	A5.. %	B5.. 5	C5.. E	D5.. U	E5.. e	F5.. u
6	SSA ^F	SPA ^V	A6.. &	B6.. 6	C6.. F	D6.. V	E6.. f	F6.. v
7	ESA ^G	EPA ^W	A7.. ^	B7.. 7	C7.. G	D7.. W	E7.. g	F7.. w
8	HTSfe^H	98. ^X	A8.. (B8.. 8	C8.. H	D8.. X	E8.. h	F8.. x
9	HTJfe^I	99. ^Y	A9..)	B9.. 9	C9.. I	D9.. Y	E9.. i	F9.. y
A 10	VTSfe^J	9A. ^Z	AA.. *	BA.. :	CA.. J	DA.. Z	EA.. j	FA.. z
B 11	PLDfe^K	CSI ^[AB.. +	BB.. ;	CB.. K	DB.. [EB.. k	FB.. {
C 12	PLUfe^L	ST. ^\	AC.. ,	BC.. <	CC.. L	DC.. \	EC.. l	FC..
D 13	RI.fe^M	OSC ^]	AD.. -	BD.. =	CD.. M	DD..]	ED.. m	FD.. }
E 14	SS2 ^N	PM. ^^	AE.. .	BE.. >	CE.. N	DE.. ^	EE.. n	FE.. ~
F 15	SS3 ^O	APC ^_	AF.. /	BF.. ?	CF.. O	DF.. _	EF.. o	DEL d

Functions in columns 8 and 9 identical to those attained in the seven bit environment through use of two symbol sequence "ESC + character" where the character is from columns 4 or 5.

Possible Extended Graphic Set - Bit 8 on

columns	8	9	A	B	C	D	E	F
row 0			0					pppp
			000					pppp
			0 0 0	e e e	pppp			pppp
			00000	e e e	pppp			pppp
			0000	e e e	pppp			pppp
			00000	e e e	pppp			pppp
row 1			1	AAA	0000	aaa	qqqq	
			111	A A A	0000	a a a	q q q	
			111	A A A	0000	a a a	q q q	
			11111	AAAA	0000	a a a	q q q	
row 2			2	BB	RR	bbbb	rrrr	
			222	B B B	R R R	b b b	r r r	
			222	B B B	R R R	b b b	r r r	
			222	B B B	R R R	b b b	r r r	
			22222	BBB	RRR	bbb	rrr	
row 3			3333			cccc	ssss	
			33333			c c c c	s s s s	
			3333	CCCC	SSSS	c c c c	s s s s	
			3333	CCCC	SSSS	c c c c	s s s s	
			33333	CCCC	SSSS	c c c c	s s s s	
row 4			4 4 4			ddd	tttt	
			4444	DDD	TTTT	d d d	t t t	
			44444	D D D	T T T	d d d	t t t	
			444	D D D	T T T	d d d	t t t	
			4444	D D D	T T T	d d d	t t t	
			44444	D D D	T T T	d d d	t t t	
row 5			5 5 5			eee	uuu	
			555	E E E	U U U	e e e	u u u	
			555	E E E	U U U	e e e	u u u	
			555	E E E	U U U	e e e	u u u	
			55555	EEEE	UUU	eee	uuu	
row 6			6			f	v v v	
			666	F F F	V V V	f f f	v v v	
			66666	F F F	V V V	f f f	v v v	
			666	F F F	V V V	f f f	v v v	
			666	F F F	V V V	f f f	v v v	
row 7			777			ggggg	www	
			7777	GGGG	WWWW	g g g	w w w	
			77777	G G G	W W W	g g g	w w w	
			777	G G G	W W W	g g g	w w w	
			7777	G G G	W W W	g g g	w w w	
row 8			8888			h h h	x x x	
			88888	H H H	X X X	h h h	x x x	
			88888	H H H	X X X	h h h	x x x	
			88888	H H H	X X X	h h h	x x x	
			888	H H H	X X X	h h h	x x x	
row 9			9			i i i	y y y	
			999	I I I	Y Y Y	i i i	y y y	
			99999	I I I	Y Y Y	i i i	y y y	
			99999	I I I	Y Y Y	i i i	y y y	
			9999	I I I	Y Y Y	i i i	y y y	
row A			J J J			j j j	z z z	
			J J J	J J J	Z Z Z	j j j	z z z	
			J J J	J J J	Z Z Z	j j j	z z z	
			J J J	J J J	Z Z Z	j j j	z z z	
row B			K K K			k k k		
			K K K	K K K	k k k	k k k		
			K K K	K K K	k k k	k k k		
			K K K	K K K	k k k	k k k		
row C			L L L			l l l		
			L L L	L L L	l l l	l l l		
			L L L	L L L	l l l	l l l		
			L L L	L L L	l l l	l l l		
row D			M M M			m m m		
			M M M	M M M	m m m	m m m		
			M M M	M M M	m m m	m m m		
			M M M	M M M	m m m	m m m		
row E			N N N			n n n		
			N N N	N N N	n n n	n n n		
			N N N	N N N	n n n	n n n		
			N N N	N N N	n n n	n n n		
row F			O O O			o o o		
			O O O	O O O	o o o	o o o		
			O O O	O O O	o o o	o o o		
			O O O	O O O	o o o	o o o		
row								

USASCII Graphic Assignments - normal set - bit 8 off

columns	0	1	2	3	4	5	6	7
row 0	eeee	pppp	000	eee	pppp			
	eeee	pppp	000	eee	pppp			
	eeee	pppp	000	eee	pppp			
	eeee	pppp	000	eee	pppp			
	eeee	pppp	000	eee	pppp			
row 1	AAAA	000	111	A A A	000	aaa	qqq	
	A A A	000	111	A A A	000	aaa	qqq	
	A A A	000	111	A A A	000	aaa	qqq	
	A A A	000	111	A A A	000	aaa	qqq	
	A A A	000	111	A A A	000	aaa	qqq	
row 2	B B B	RRR	222	B B B	RRR	bbb	rrr	
	B B B	RRR	222	B B B	RRR	bbb	rrr	
	B B B	RRR	222	B B B	RRR	bbb	rrr	
	B B B	RRR	222	B B B	RRR	bbb	rrr	
	B B B	RRR	222	B B B	RRR	bbb	rrr	
row 3	C C C	SSS	333	C C C	SSS	ccc	sss	
	C C C	SSS	333	C C C	SSS	ccc	sss	
	C C C	SSS	333	C C C	SSS	ccc	sss	
	C C C	SSS	333	C C C	SSS	ccc	sss	
	C C C	SSS	333	C C C	SSS	ccc	sss	
row 4	D D D	TTT	444	D D D	TTT	ddd	ttt	
	D D D	TTT	444	D D D	TTT	ddd	ttt	
	D D D	TTT	444	D D D	TTT	ddd	ttt	
	D D D	TTT	444	D D D	TTT	ddd	ttt	
	D D D	TTT	444	D D D	TTT	ddd	ttt	
row 5	E E E	U U U	555	E E E	U U U	eee	uuu	
	E E E	U U U	555	E E E	U U U	eee	uuu	
	E E E	U U U	555	E E E	U U U	eee	uuu	
	E E E	U U U	555	E E E	U U U	eee	uuu	
	E E E	U U U	555	E E E	U U U	eee	uuu	
row 6	F F F	V V V	666	F F F	V V V	fff	v v v	
	F F F	V V V	666	F F F	V V V	fff	v v v	
	F F F	V V V	666	F F F	V V V	fff	v v v	
	F F F	V V V	666	F F F	V V V	fff	v v v	
	F F F	V V V	666	F F F	V V V	fff	v v v	
row 7	G G G	W W W	777	G G G	W W W	ggg	www	
	G G G	W W W	777	G G G	W W W	ggg	www	
	G G G	W W W	777	G G G	W W W	ggg	www	
	G G G	W W W	777	G G G	W W W	ggg	www	
	G G G	W W W	777	G G G	W W W	ggg	www	
row 8	H H H	X X X	888	H H H	X X X	hhh	xxx	
	H H H	X X X	888	H H H	X X X	hhh	xxx	
	H H H	X X X	888	H H H	X X X	hhh	xxx	
	H H H	X X X	888	H H H	X X X	hhh	xxx	
	H H H	X X X	888	H H H	X X X	hhh	xxx	
row 9	I I I	Y Y Y	999	I I I	Y Y Y	iii	yyy	
	I I I	Y Y Y	999	I I I	Y Y Y	iii	yyy	
	I I I	Y Y Y	999	I I I	Y Y Y	iii	yyy	
	I I I	Y Y Y	999	I I I	Y Y Y	iii	yyy	
	I I I	Y Y Y	999	I I I	Y Y Y	iii	yyy	
row A	J J J	Z Z Z		J J J	Z Z Z			
	J J J	Z Z Z		J J J	Z Z Z			
	J J J	Z Z Z		J J J	Z Z Z			
	J J J	Z Z Z		J J J	Z Z Z			
	J J J	Z Z Z		J J J	Z Z Z			
row B	K K K			K K K				
	K K K			K K K				
	K K K			K K K				
	K K K			K K K				
	K K K			K K K				
row C	L L L			L L L				
	L L L			L L L				
	L L L			L L L				
	L L L			L L L				
	L L L			L L L				
row D	M M M			M M M				
	M M M			M M M				
	M M M			M M M				
	M M M			M M M				
	M M M			M M M				
row E	N N N			N N N				
	N N N			N N N				
	N N N			N N N				
	N N N			N N N				
	N N N			N N N				
row F	O O O			O O O				
	O O O			O O O				
	O O O			O O O				
	O O O			O O O				
	O O O			O O O				

[illegible]

This section which provides a Description of the SAM76 language is not covered by copyright. SAM76 Inc. would, however, appreciate receiving one copy of any work in which all, or a substantial part of this description might be incorporated.

The SAM76 language

The SAM76 language combines into a single interpretive processor characteristics of two different string and general purpose macro generators and one (or more) infix operator mathematical systems.

SAM76 was primarily inspired by the "M6 MACRO PROCESSOR" designed by M. D. Mc Ilroy and R. Morris of the Bell Telephone Laboratories. A description authored by Andrew D. Hall is given in the Bell Lab. Computing Science Technical Report #2, and other places.

The second source of inspiration came from the syntax of "GPM - a GENERAL PURPOSE MACRO GENERATOR" described by its author - C. Strachey.

GPM syntax very significantly improves the interface between the user and the language by eliminating an awkwardness inherent in the design of M6. This awkwardness arises out of the fact that in the M6 language the specification for the action to be performed on the result of an expression evaluation is at the right hand end of the expression to be evaluated; this requirement causes the need for a unique pair of characters to be designated for the purposes of "quotation"; consequently when the writer of procedures in M6 is terminating the writing of an expression he must mentally reconstruct the sequence of alternations of "quoted", "active" and "neutral" expressions which may be nested one within the other.

In GPM, where the nature of the expression is specified at the left end, only one character need be identified to serve to provide the right hand boundary for all three types.

This means that the user, when closing out his writings, need only supply enough such identical characters (plus a few for good measure) to enable correct action to subsequently take place.

An universal lament of users of both M6 and GPM relates to the extremely unnatural method in which arithmetic expressions must be organized. The simplest arithmetic expression in normal "infix" notation becomes a tortuous involved nest of strange symbols when expressed in either M6 or GPM. A simple solution exists, is incorporated in SAM76 and forms the third element of the language.

In addition inspiration for the selection and design of a number of the resident functions came from a variety of sources (see reference list).

Acknowledgement is made of the contribution made by L. Peter Deutsch [U.C. Berkeley] who conceived of nesting an expression calling for input inside an expression which causes output [Ref. 3] thus:

{OUTPUT,{INPUT}}

Finally acknowledgement and great appreciation is expressed for:

The help received from Neil Colvin [M.I.T.] who provided me with an 8080 simulator for the PDP-10 and who coded a preliminary version of the language to run in an 8080 based microprocessor system contributed also materially to the development of this language.

Barry Lubowsky [Rider College] who allowed me to make use of a PDP-10 and who was most helpful with some system software problems.

Roger Amidon [Delta Data Systems], whose actual 8080 based "ALTAIR" and "IMSAI" microcomputers were used to test and debug the software, was also most helpful in developing a family of "people" oriented mnemonics and functions particularly well suited for the "amateur" user.

Other people who contributed to various details and who "people" tested the language are Carl Galletti [T.D.L.], Marty Nichols [Newsweek], and Tom Kirk [N.J. Bell].

References

1. Strachey, C. - "A general purpose macrogenerator", Computer Journal, Vol. 8, No. 3, Oct. 1965 p. 225; |19651000|.
2. Hall, Andrew D. - "The M6 macroprocessor", Bell Telephone Labs; Computer Science Report No. 2; 1971;
3. Kagan, Claude A. R. - "A string language Processor for small machines", Proceedings of the ACM SIGPLAN Symposium on the Pedagogical Applications of small Computers, University of Kansas, Lawrence, Kansas |19711118|.

Notice

This description of the SAM76 language is substantially the same that is available from the IEEE Computer Society Repository, R 76-301 - August 1976.



The three elements of SAM76

The first element is designed around the characteristics described by STRACHEY in his GRM (General Purpose Macro Generator) language.

GRM - S76

Strachey defined its general forms as follows:

Active %function,arguments, :

Neutral %VAL,function,arguments, :

In order to avoid ambiguity and to provide consistency the above is changed for the purposes of SAM76 as follows:

Active %function,arguments, /

Neutral &function,arguments, /

Protected ! . . . protected string . . . /

In addition changes and extensions are made to the language originally described by Strachey. The modified form incorporated in SAM76 is to be known as S76; this is a fully operational proper subset of SAM76.

The syntax used in connection with the S76 system is to be referred to as the "S" syntax of SAM 76.

The second element is substantially as described by Andy HALL for the M6 language attributed to Mc ILROY and MORRIS:

M6 - M76

Active #function,arguments, :

Neutral #function,arguments, ;

Except for different function mnemonics no change was made in the syntax and notation of M6 for the purposes of SAM76. In consequence routines and texts written for M6 processors distributed by the Bell Labs outside the Bell System should be compatible with SAM76.

Compatibility with M6 is insured by enabling user to stipulate whether "resident" or "user defined" functions are to be first searched whenever an expression is to be evaluated.

Expressions written using the M76 syntax will first cause a search of the user defined functions; if no user defined function of that name exists then a search is made of the resident functions. Expressions written using the S76 syntax will first search the resident functions.

Although there is syntactic compatibility with M6, the implementation in SAM76 is designated M76 because of the generally different complement of resident functions, and the significant enhancement of the scan algorithm.

The syntactic form of expressions based on the M76 system is to be referred to as the "M" syntax.

An infix operator system characterized in general by the following example for a simple arithmetic expression:

A76

Example [5 + 5 * (144/12)]

The syntax of the foregoing system is to be referred to as the "A" syntax. The evaluation rules and internal syntax may vary between systems for this sub system; a method of identification is provided within the framework of SAM76.

Input - Activating - Scanning and Output
--

Input and Output

The flow of characters, be they procedures (programs) or just plain text between the SAM76 processor and either the user or other sources or destinations is under the control of two resident SAM76 functions:

Input from user (or source) to SAM76 is under control of the "input string" function whose mnemonics are "IS"; this may be used in either the "S" or "M" syntax either as an active or neutral function thus:

type	active		neutral
S76	%IS/	or	&IS/
M76	#IS:	or	#IS;

Output from the SAM76 processor to the user (or other destination) is controlled by the resident "output string" function, mnemonic of which is "OS"; active or neutral use in both the "S" and "M" syntax is:

type	active		neutral
S76	%OS/	or	&OS/
M76	#OS:	or	#OS;

Initially and whenever its task has been completed, the SAM76 processor automatically loads a "restart" expression which is scanned and provides the means for accepting input and delivering output; this expression is initially defined as the S76 expression:

&os,%is//=

The "restart" expression

The initial or standardized condition makes use of the equal sign "=" to ACTIVATE the processor; this "activating" character may be changed by the user through execution of the appropriate SAM76 resident function.

Activating the processor

Scanning	When expressions are nested one within the other, the SAM76 processor first executes the innermost, proceeding from left to right.
----------	--

The innermost in the "restart" expression is:

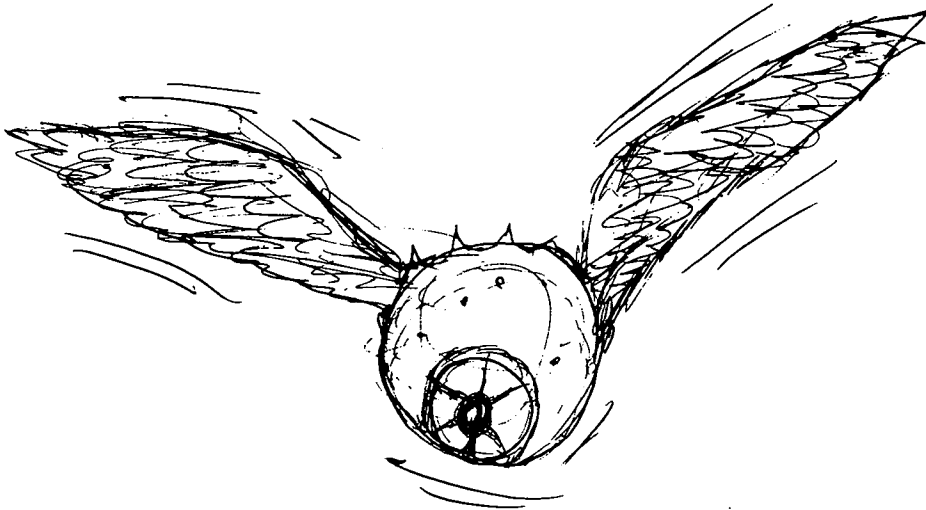
%is/

and therefore input is sought; the user input is terminated with an activating character, the (=) will be shown to remind the user how to terminate his input.

Once the input is terminated, it is in turn examined by the scanner for possible further execution until finally the outermost expression of the "restart" expression is encountered:

&os, ... /

where the series of periods represents the result of processing the user input and in fact will be the output that is to be given the user.



Quoting or Protection

Frequently the user may wish to incorporate in his text certain characters which have special meaning to the scanning system of SAM76. Typical of these are the characters used to indicate the start and end of expressions. These are known as "warning characters".

For standardization purposes a processor for the SAM76 language is initialized with the following assignment of "warning characters":

Abbv.	Char.	Octal	Purpose
SA	%	45	Begin S76 active expression
SN	&	43	Begin S76 neutral expression
SP	!	41	Begin S76 protected string
SE	/	52	End S76 expr. or protected string
MB	#	43	Begin M76 expression
MA	:	72	End M76 active expression
MN	;	73	End M76 neutral expression
AS	,	77	Argument separator
QC	@	100	Quote single character
IC	`	140	Ignore single character
BQ	<	74	Begin quoted string - Pair 1
EQ	>	76	End quoted string - Pair 1
BP	(50	Begin protected string - Pair 2
EP)	51	End protected string - Pair 2
BA	[133	Begin "A - syntax" expression
EA]	135	End "A - syntax" expression

Quoting or protecting a string	In both the "S" and "M" forms of SAM76, the equivalent of string quotes may be provided by any pair of characters designated for that purpose. Initially two such pairs are defined and consist of the following balanced pairs of enclosures:
--------------------------------------	--

< > or (. . . .)

Strings of symbols found between the above illustrated enclosures are said to be "quoted" or "protected" and are completely ignored during the scanning process.

The two rules that follow are applied to the scanning process with respect to the use of "quoting characters".

2

When an unprotected character designated as a "quoting character" is encountered in the left to right scan of an expression, the other character of the pair is then defined as the character which will end the quoted string; this means that for the two initially designated pairs shown above, quoting or protection may also be achieved as follows:

> < or) (

1

In the process of finding the character which is to terminate a particular quoted string any occurrence of the same character as that which identified the beginning of that quoted string must be balanced by a corresponding "end of quoted string" character.

This is to be differentiated from the action which takes place in connection with the "protected string" of the "S" syntactic form:

! . . . protected string . . . /

In the foregoing a tally is made of "unquoted" warning characters in order to find the slant sign "/" which matches the exclamation mark:

The tally count is increased by a count of one - % & ! #

The tally count is decreased by a count of one - / : ;

It should, however, be noted that the "warning characters" (symbols used to delineate bounds of expressions) shown in the foregoing illustrations and examples are changeable by the user and are only representative of initial or standardized conditions.

Any of the foregoing "warning characters" may be changed using the appropriate SAM76 resident function; the abbreviations listed earlier are used to identify the particular warning character it is desired to change.

Quoting or protecting a single character

When an unprotected "character quoting symbol" is encountered, then the immediately following character is considered to be quoted. This is particularly useful when the user wishes to quote single characters, such a warning character, without having to maintain a balance of quoting character pairs.

Initially the "character quoting symbol" is set to be the "commercial at" (@).

Using the SAM76 syntax, functions, and trace

In the examples that follow the | ... | arrangement is used only to delineate output at the user terminal; if for instance the user wished to add two numbers making use of the "A" syntactic form, the following might be observed:

$$[5 + 5] = |10|$$

M76 example	In the following example the M76 syntax is used to create a "macro" which when invoked will square any desired number:
-------------	--

```
#DT,SQ,<#MU,q1,q1:>:=
```

DT means Define Text
 MU means Multiply
 SQ is name given to text

#PT, SQ, q1:= PT means Partition Text

There are four possible actions which may ensue when the macro, or user defined function named SQ is to be invoked:

1. #SQ, 5: =|25| value returned is 25
2. #FT, SQ, 5: =|25| FT - Fetch Text, value 25
3. #SQ, 5: =|25| value returned is 25
4. #FT, SQ, 5: | #MU, 5, 5: | value is Text of SQ

S76 example	The foregoing example is repeated below making use of the S76 syntax; however to illustrate its use, the infix system is used for the actual multiply function:
-------------	---

%DT,SQ,! [q1 * q1] /= q1 is dummy variable

%PT,SQ,q1 /= q1 becomes partition value 1

The four invocations of SQ are shown below:

1. %SQ, 5/=|25| value is 25
2. %FT,SQ, 5/=|25| value is again 25
3. &SQ, 5/=|25| value is still 25
4. &FT,SQ, 5/=|[5 * 5]| value is now "text" of SQ

Resident Functions	In general the mnemonics used in SAM76 to identify the resident functions are composed of two alphabetic characters. This is not a restriction imposed on the language, and three or more alphabetic characters may be used to identify resident functions. In fact the user may wish to employ the "@F" ("wh@" (what) Functions) mnemonic to ascertain the mnemonics in actual being in his system.
--------------------	--

It should be remembered that when the "@" sign serves the purpose of "quoting single character" then the expression to list the resident functions would be:

M76 #@@F, ... :

S76 %@@F, ... /

where " ... " is the string of characters which will precede each resident function name in the result of executing this expression.

Descriptions of the functions and examples in which either S76 or M76 syntax is applicable employ the | (Long Vertical Mark) character to bound the expression. In actual use the | is replaced by the appropriate pair of warning characters.

In the case of "null" valued functions where the neutral form of the expression is to be illustrated, the \ "reverse slash" is used in lieu of the |.

The formal description of the "DT" - "Define Text" function is given below to illustrate the manner of presenting SAM76 resident functions.

|DT,t,s,d| Define Text with id number

This function places in the "text area" the string denoted by "s" in the expression shown above; this text is given the name denoted by "t". An arbitrary identification decimal number denoted by "d" may also be included as part of the "text" record. Previously defined text with same name is erased and replaced with this definition.

Examples:

M76 #DT,NAME,THE SKY IS BLUE:

S76 %DT,NAME,THE SKY IS BLUE/

In the foregoing examples no id. number was desired, and consequently the default value of zero is stored as part of the record created by this function.

Texts in the "text area" may be examined without evaluation using the "vt" function. This examination will display location and value of any "partitions", "multi-partitions", and the text divider(s) when at a location other than the head (or extreme left) end of the text.

The "VT"
View Text
function

Initial conventions for the display within a "text" of partitions, multi-partitions and text divider(s) are:

[1], [2], ... [d]	Partitions
[#1], [#2], ... [#d]	Multi-partitions
[^]	Text Divider

When in the "Trace" condition, active expression about to be evaluated will be shown bounded by the |, neutral expression by \, regardless of the syntax (M or S) of the expression.

Trace

"id" numbers

The M6 language provides a unique identification number for each of the resident functions of the language, and means for the user to assign a similar type of number to defined functions. This numbering concept is preserved in SAM76; numbers between 1 and 99 inclusive are not assigned to any resident SAM76 function unless its name and operation is absolutely identical to its M6 counterpart. This permits user defined functions to be written to mimic M6 functions if desired. A list of M6 functions and their assigned "id" numbers is given in an appendix.

To illustrate, the M6 function "DEF" might be mimicked in the SAM76 system as follows:

User defined function

1a. %DT,DEF,!%NI,<&>,<%>/DTmpl</> //

2. %MT,DEF,mpl/

examination of the defined text (or function) DEF can be made using the SAM76 "VT - View Text" resident function thus:

3. %VT,DEF/=|%NI,<&>,<%>/DT[#1]</>|

in the foregoing "mpl" is a dummy for a "multi-partition" which will be created by the second expression and visualized by the VT function as [#1].

The function "NI - Neutral Implied" is used to provide the expression using DT with either the active or neutral warning character as used with an expression containing "DEF" as its operative argument.

Alternatively expression (1a) could have been written in one of the following ways, with exactly the same end effect:

1b. %DT,DEF,<%NI,@&,@%/DTmpl/> /

1c. #DT,DEF,<#NI,@&,@%:DTmpl/>:

1d. #DT,DEF,<#DTmpl%NI,@:,@:/>:

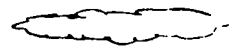
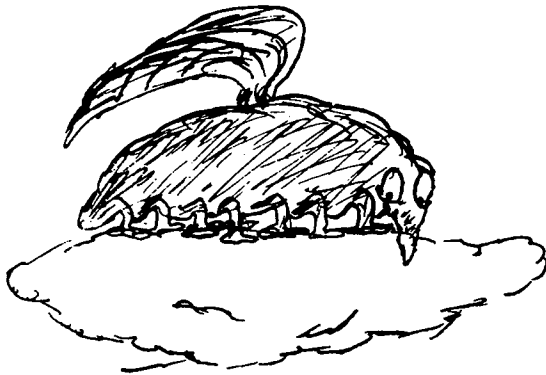
1e. #DT,DEF,<#DTmpl%NI,<;>,<:>:>:

Ignoring next character

Oftentimes the user wishes to arrange his script for convenient legibility using "nulin", "tabs" or other characters; this is strictly for formatting purposes. Under normal circumstances these characters will be part of the expressions, and be returned in the value string unless deleted in the scanner.

One way of achieving this is to place these characters in unused arguments of primitives such as the third argument of an "os" expression. This could be a problem if at some later time the additional unused arguments acquire a purpose.

Under initialized conditions the "" character is used to signify that the immediately next ensuing character is to be deleted in the scanner.



Algorithms in the SAM76 language

introduction

In the algorithms described and illustrated in this section, use is made of the procedure named "print" to display the script of the functions, expressions, and other text strings.

The "print" function is useful in that it augments the capability of the "vt - view text" function by prefixing each individual "text" displayed with its name bounded by angle brackets; assuming that texts have been created as shown:

```
{}-----
{} %dt,a,apple/=
{} %dt,b,bottle/=
{} %pt,b,t/=
{}
{}-----
```

The use of "print" would cause the following display:

```
< a >
apple

< b >
bo[1][1]le
```

The need and use of the "print" expression was described earlier; the expression to accomplish this task uses a multi partition value #2.

print

Typical use to print out name and contents of the text area which in this example only contains the "print" expression would be:

```
{}-----
{} %print%lt,(,)//=
{} < print >
{} %ii,[1],,,!%os,
{} (< [1] >)/%vt,[1]/%os,
{} /%print[#2]///
{}
{}-----
```

A simple way of doing some procedure "proc" some desired "n" number of times is exemplified by the use of "ds - duplicate string" in the expression named "dnt":

do "n" Times

```
< dnt >
%ds,[1],!%proc///
```

The simple recursive function named "fac" generates the factorial of any number which replaces partition value 1 in its use.

factorial

```
< fac >
%ig,1,[1],1,!%mu,[1],%fac,%su,[1],1/////
{}-----
{} %fac,5/=120
{}
{}-----
```

This recursive function named "exp" is used to generate the exponential of "m" to the "n"th. power; in use "m" replaces partition value 1, and "n" partition value 2.

exponential

```
< exp >
%ig,[2],,!%mu,[1],%exp,[1],%su,[2],1////,1/
{}-----
{} %exp,3,7/=2187
{}
{}-----
```

The recursive expression named "mrn" extracts the "m"th. root of any number "n"; it should be noted that this function makes use of the expression named "exp", which generates an exponential.

"m"th. root of "n"

```
< mrn >
%ig,%exp,[3],[2]/,[1],
!%su,[3],1//,
!%mrn,[1],[2],%ad,[3],1////
{}-----
{} %mrn,2187,7/=3
{}
{}-----
```

Expression named "tensor" is used to generate a list of successive numbers separated by "."; the size of the list is specified by the integer number which replaces partition value 1 in the expression.

tensor

```
< tensor >
%ig,[1],1,!%tensor,%su,[1],1//.[1],1/
```

A frequently needed capability is that of generating "roman" numbers, as for text section and chapter headings; when function "DTR" is invoked with a decimal number as its argument, the value returned is the same in roman notation.

Decimal to Roman

Thanks are extended to Jim Gimpel, of the Bell Telephone Laboratories for suggesting this as a most useful adjunct to the SAM76 language, as it is to the SNOBOL language.

```
< DTR >
%dt,x,[1]/%rom1/%y/%et,x,y,z/

< rom1 >
%ig,%crd,x/,!,!%rom2,%fc,x//%rom1///

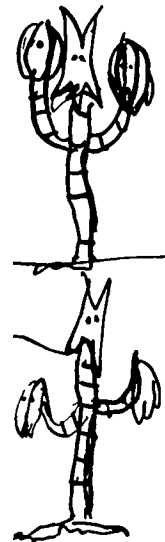
< rom2 >
%dt,z,[1]/%pt,z%d//%dt,z,%z%r///%pt,y%u//%dt,y,%y%t//%z//

< d >
,0,1,2,3,4,5,6,7,8,9

< r >
,,I,II,III,IV,V,VI,VII,VIII,IX

< u >
,I,V,X,L,C,D,M

< t >
,X,L,C,D,M,*, $
```



The "hex" freak can fulfill his desires by defining the "HTR" expression to achieve "hex to roman" conversion:

hex to roman

```
< HTR >
%DTR,%xd,[1]//
```

The user often finds it necessary to derive a series of answers resulting from the execution of some script for a range of numbers "n1" to "n2"; the script loop provides this capability by enabling the user to specify for partition 1 the name of the function, the lower of the two numbers "n1" as partition 2 and the higher number "n2" as partition 3.

loop

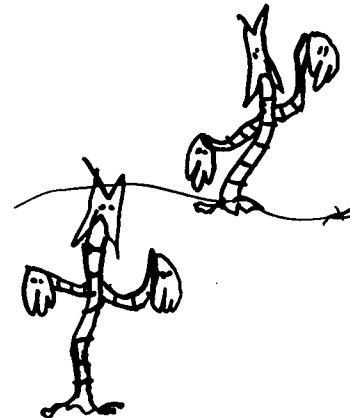
```
< loop >
%zs,6,[2]/%zs,7,%su,[3],[2]//%zi,7/%lp1,[1]/
```

```
< lp1 >
%lp2,[1],%zq,6//%zi,6/%zd,7,,,%lp1,[1]///
```

```
< lp2 >
!
/%ps,-4, ,[2]/    %[1],[2]/
```

For example the user wishes to ascertain the "roman" number equivalent of the decimal numbers from 105 to 116:

```
{ } -----
{ } %loop,DTR,105,116/=
{ } 105    CV
{ } 106    CVI
{ } 107    CVII
{ } 108    CVIII
{ } 109    CIX
{ } 110    CX
{ } 111    CXI
{ } 112    CXII
{ } 113    CXIII
{ } 114    CXIV
{ } 115    CXV
{ } 116    CXVI
{ } -----
```



Often times it is desired to execute a procedure on each and every one of some arbitrary number of "texts" in the storage area; as an example a script designed to provide a list of text names and the count of characters in each text will be illustrated; the key to this script was the use of "mt - multi part text"; it was used to create the multi partition value "#2" seen in the expression named "repeat".

repeat

```
< repeat >
%ii,[1],,,!%procl,[1]/%repeat[#2]///
```

```
< procl >
%os,
%ps,10, ,[1]/%ps,-5, ,%crd,[1]///
```

If, for instance, the only texts in the text area were "repeat" and "procl", then the use of repeat with an "lt - list text function" would appear as follows:

```
{ }-----
{ } %repeat%lt,(,)//=
{ } repeat      26
{ } procl       32
{ }-----
```

The "skim" expression "skims off" the first appearance of each different character of the string which is placed as an argument to the expression; the result is returned as the value.

skim

```
< skim >
%dt,,([1])/%rs,%i//%et,/
```

```
< i >
%ig,%crd,,,!%k,&fc,////
```

```
< k >
%pt,,[1]/%i/([1])
```

A simple example of the use of "skim" follows:

```
{ }-----
{ } %skim,MISSISSIPPI/=MISP
{ }-----
```

In his book "Algorithms in SNOBOL4" Jim Gimpel, the author, offers SKIM as an example of string manipulation; it is interesting to note the typing effort required for the same function in each language; it is assumed that a speed of ten words per minute (or one character per second) is reasonable for intellectual typing. At that rate the SAM76 script would take approximately 100 seconds, while the SNOBOL4 example would require approximately 143 seconds to type up.

"pi" - I. Newton, J. Raamot and J. Gorman

Everybody at one time or another wishes to calculate the value of "pi" to any number of places; the procedure named "p" does just that, and it only uses the "ad", "su", "ig", "ii", and "mu" functions.

< p >

```
%ig,%ig,%ad,[1],%ad,[3],%ad,[3],1///,0,`
!%ad,[1],%ad,[3],%ad,[3],1////,`
!%su,0,%ad,[1],%ad,[3],%ad,[3],1/////,`
%ig,%ad,%su,%su,[1],[2],/, [2]/,1/,0,`
!%ad,%su,%su,[1],[2]/, [2]/,1//,`
!%su,%ad,[2],[2]/,%ad,[1],1////,`
!%ii,[2],0,`
!%ad,[4],[4]//,`
!%p,%ad,%su,%su,[1],[2]/, [2]/,1/, %su,[2],1/, [3],%ad,[4],[3]////,`
!%p,%ad,[1],%ad,[3],%ad,[3],1///, [2],%ad,[3],1/, %ad,[4],[2]////
```

The expression named "pi" is used to pass into "p" a number, the larger that number, the more accurately "pi" will be computed - but more time is required.

< pi >

```
%p,[1],[1],[1],[1]/
```

Two examples illustrating the above are shown below:

```
{ }-----
{ } %pi,10/=324
{ } %pi,100/=31468
{ }-----
```

It is obvious that if "pi" is available in this manner, then it becomes relatively simple to determine all kinds of other functions such as for trigonometry.

The concepts from which this procedure stems are attributable to Newton, and Jaan Raamot; an early implementation of this concept was written in "basic" by Johnny Gorman.

Hanoi - by Richard A. Stone

Hanoi is a game derived from the age old riddle called the Towers of Brahma. There are given three towers or posts and a number of rings of different sizes with no two the same size. The rings are placed on one of the posts in order of size from the largest at the bottom to the smallest at the top. The object of the game is to move the rings one at a time until all the rings are on one of the other posts without having set a larger ring on a smaller one. Supposedly some Buddhist monks near Hanoi are playing this game on large posts with 64 gold rings and believe that when all the rings have been transferred the world will end.

The Hanoi algorithm was originally designed in SNOBOL, another computer language dealing with text and described in a computer manual titled "SNOBOL 4 Programming Language" by Griswold, Poage, and Polonsky, published by Prentice Hall. The technique that this algorithm uses is known as a binary tree.

It is impossible from looking at the Hanoi script to determine just how it works because the script is so recursive. Therefore, to see how Hanoi works it is necessary to have its path diagrammed.

Hanoi is fetched with four values to be inserted: the number of rings; the starting post name; the ending post name; and the intermediate post name. If the number of rings is zero the value of Hanoi is the Null string; Hanoi fetches itself twice. The important thing to notice at this point is that when Hanoi fetches itself, it rearranges the order of the posts. The first fetch switches the second and third posts, and the second fetch switches the first and third posts. Also notice that the number of rings is decreased by one with each new fetch. To diagram this a dot will be drawn for each Hanoi fetched and that dot will be labeled with the order of the four values next to it.

Another interesting item is that the number of HANOI's executed, which is the same as the number of rings moved is 2^n (two to the nth. power) where n is the number of rings. With 64 rings 2^{64} (two to the 64th. power) be equal to 18,446,744,073,709,551,616 moves, which {would} take until the end of the world to do by hand or even by computer. So you see the Buddhist monks are right about when the world will end!

The script which accomplishes this, despite its complexity, is very short. Here it is:

```
< hanoi >
%i, [1], 0, , !%hanoi, %su, [1], 1/, [2], [4], [3]/!
Move Ring [1] from [2] to [3]/
%hanoi, %su, [1], 1/, [4], [3], [2]///
```

And here is some output it gives if the posts are called respectively "hither", "thither", and "yon", and five rings are involved.

```
{ }-----
{ } %hanoi, 5, hither, thither, yon/=
{ } Move Ring 1 from hither to thither
{ } Move Ring 2 from hither to yon
{ } Move Ring 1 from thither to yon
{ } Move Ring 3 from hither to thither
{ } Move Ring 1 from yon to hither
{ } Move Ring 2 from yon to thither
{ } Move Ring 1 from hither to thither
{ } Move Ring 4 from hither to yon
{ } Move Ring 1 from thither to yon
{ } Move Ring 2 from thither to hither
{ } Move Ring 1 from yon to hither
{ } Move Ring 3 from thither to yon
{ } Move Ring 1 from hither to thither
{ } Move Ring 2 from hither to yon
{ } Move Ring 1 from thither to yon
{ } Move Ring 5 from hither to thither
{ } Move Ring 1 from yon to hither
{ } Move Ring 2 from yon to thither
{ } Move Ring 1 from hither to thither
{ } Move Ring 3 from yon to hither
{ } Move Ring 1 from thither to yon
{ } Move Ring 2 from thither to hither
{ } Move Ring 1 from yon to hither
{ } Move Ring 4 from yon to thither
{ } Move Ring 1 from hither to thither
{ } Move Ring 2 from hither to yon
{ } Move Ring 1 from thither to yon
{ } Move Ring 3 from hither to thither
{ } Move Ring 1 from yon to hither
{ } Move Ring 2 from yon to thither
{ } Move Ring 1 from hither to thither
{ }-----
```

KNIKKER - by W. L. Van der Poel

Editor's Note

This most interesting group of scripts should prove to be as much a challenge to you as it was "dutch" to us; the first object of the game is to figure out what it is supposed to do, then the second object is to figure out how it does it.

If you need help, ask a Dutch friend, or write the author, Dr. W. L. Van der Poel, Delft Institute of Technology, Delft, the Netherlands.

< KNIKKER >

```
%os,
BEGINTOESTAND? /%dt,T1,%id,8//%os,
EINDTOESTAND? /%dt,T2,%id,8//%os,
TUSSENRESULTATEN? <<TYPE J VOOR JA>> /%dt,TRACE,%ic//%os,
/%ii,%TRACE/,J,!%os,
DE GEGOOIDE KNIKKER@: ///
%ii,%T1/,%T2/,!%os,NIETS/%KNIKKER//,!%SP2///
```

< CHANGE >

```
%dt,N,[1]/%dt,M,[2]/%dt,RY1,%fdc,M,3//%dt,RY2,%fdc,M,2,!%FOUT/////`
%dt,RY3,%fdc,M,3,!%FOUT/////`
%et,M/%FRY1,RY1,N/%SRY,RY2,N/%FRY1,RY3,N/%RY1/%RY2/%RY3/
```

< PFRY >

```
%ii,%[1]/,[2],!%dt,[3],%fdc,[3],[4]/%HFRY,[3],[5],[6]////
```

< HFRY >

```
%ii,%fc,[1]/,1,!0%[1]/%dt,N,[2]//,!1%[1]/%dt,N,[3]///
```

< FRY1 >

```
%PFRY,[2],1,[1],0,1,5/%PFRY,[2],2,[1],1,5,6/%PFRY,[2],3,[1],2,6,3/
```

< SRY >

```
%PFRY,[2],5,[1],0,1,2/%PFRY,[2],6,[1],1,2,3/
```

< FOUT >

```
%os,FOUT GEMAAKT!//%RR/
```

< TEST >

```
%dt,PP,[1]/%dt,QQ,[2]/%PRENT,PP,QQ/
```

< NOTFOUND >

```
%os,
NIET MOGELIJK!!!!/%KNIKKER/
```

< HDR2 >

%ii,%dt,HDR3,%fdc,[1],-1//%HDR3/,,,!%HDR3/%HDR2,[1]///

< DRAAI >

%dt,HDR1,%fdc,[1],20//%HDR2,[1]/

< FINISH >

%os,

DE BENODIGDE KNIKKERWORPEN@: %dt,F2,%DRAAI,%dt,WOPL,%WORP/%WOPL/,
/WOPL//%F2//%os,

DE DOORLOPEN TOESTANDEN ZIJN@: /%PRENT,F2,T1/%KNIKKER/

< SP2 >

%dt,RESLYS,/,%dt,TEL,1/%dt,WORP,1/%dt,WOPL,/,%dt,SRES,%T1//%SP3/

< SP3 >

%dt,HRES,%CHANGE,%WORP/,%SRES///%ii,%HRES/,%T2/,!%FINISH///`

%dt,TEL,%ad,%TEL/,1/%dt,RESLYS,(<%SRES/>)%RESLYS//`

%dt,WOPL,%WORP/%WOPL//`

%yt,RESLYS,(<%HRES/>),!%BACKTRACK/%SP3//,!%SP4///

< SP4 >

%ii,%TRACE/,J,!%os,

%WOPL////%dt,SRES,%HRES/,
/%ii,%TEL/,10,!%BACKTRACK///%SP3/

< BACKTRACK >

%dt,TEL,%su,%TEL/,1//`

%ii,%TEL/,0,!%NOTFOUND///%dt,WORP,%fc,WOPL,//`

%dt,SRES,%dt,HULP,%fr,RESLYS,@>/%dt,HULP2,%fc,HULP//%HULP//`

%ii,%WORP/,3,!%BACKTRACK//,!%dt,WORP,%ad,%WORP/,1////

< CONV2 >

%dt,CS1,[1]/%CONV3,CS1/

< CONV3 >

%ii,%dt,CS2,%fc,[1]/%CS2/,,,!%ii,%CS2/,1,L,\/@,%CONV3,[1]///

< PRENT >

%os,

%dt,[2],%CHANGE,%fc,[1]/,%[2]///%[2]//%TEKEN,%[2]//`

%ii,%[1]/,,,!%PRENT,[1],[2]///

< TEKEN >

%dt,Q1Q,%CONV2,[1]//%pt,Q1Q,L/%dt,Q1Q,&ft,Q1Q,(@/)//%PLAAT,%Q1Q//

< PLAAT >


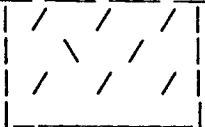
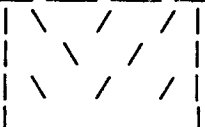
%os,

[1]	[2]	[3]
[4]	[5]	
[6]	[7]	[8]

Clues

An old toyshop item called the "Thinkadot", and the following example of the use of this group of scripts might be helpful.

```

{} -----
{} %KNIKKER/=
{} BEGINTOESTAND? 11111111
{} EINDTOESTAND? 01101011
{} TUSSENRESULTATEN? (TYPE J VOOR JA)    N
{}
{} DE BENODIGDE KNIKKERWORPEN: 111
{} DE DOORLOPEN TOESTANDEN ZIJN:
{} 01111011
{}
{} 
{}
{} 11101111
{}
{} 
{}
{} 01101011
{}
{} 
{}
{} BEGINTOESTAND?
{} -----

```

Arithmetic Evaluator - Karl Nicholas

This example demonstrates two basic algorithms; substitution and a bit of systematic juggling. What this example does is enable the user to express equations in a format approaching that of APL however still completely compatible with SAM76 of course. The user invokes "EXP" (expression), arithmetic expression = value. One can get the expression scanned in either direction by using neutral or active fetches. A normal active fetch will give a normal left to right scan. For an example of the left to right scanner:

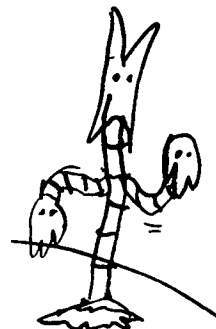
```
{}-----
{} %EXP,2+10\2/=6
{}
{}-----
```

or an example of the right to left scanner:

```
{}-----
{} &EXP,2+10\2/=7
{}
{}-----
```

or to get what you want from either scanner:

```
{}-----
{} %EXP,2+{10\2}/=7
{}
{}-----
```



As you can see, hierarchy is established by use of curly braces, they can be infinitely nested, of course. The user defines all other functions by use of the "define" function. This function creates two simultaneous lists where in the first is the symbol to be used in the evaluator and in the second is the SAM76 function or user defined function to be substituted in its place. The user invokes "define", first list append, second list append. For example:

```
{}-----
{} %define,+,ad/=
{} %define,\,di/=
{} %vt,list1/=
{} ,+,\
{} %vt,list2/=
{} ,(ad),(di,)
{}
{}-----
```

This will cover the functions used in the previous example. A view text on define reveals:

```
{ }-----
{ } %vt,define/=
{ } %dt,list1,&ft,list1/([1])/'
{ } %dt,list2,&ft,list2/((,[2]),)/
{ }
```

This one is simple; the extra commas in list2 are there to separate numbers from functions for the evaluator.

The procedure that tests for the neutral and does the substituting is "EXP". Looking at it we see:

```
{ }-----
{ } %vt,EXP/=
{ } %ni,'
{ } (%dt,arith,([1])/%pt,arith,{,}%ft,list1/'
{ } %EXP11,%arith,<%EXP11,>,</>%ft,list2///),'
{ } (%dt,arith,([1])/%pt,arith,{,}%ft,list1/'
{ } %EXP1,%arith,<%EXP1,>,</>%ft,list2///)/
{ }
```

The first thing "EXP" does is test to see if the fetch to "EXP" was neutral or not. In both cases it defines text arith as what ever was after "EXP" then partitions out of arith the curly braces and all list1 symbols. After that it fetches which ever evaluator it's supposed to depending on the neutral implied, substituting arith with commas in between all numbers and list2 replacement functions. The procedure that does all the evaluating is either "EXP1" or "EXP11". "EXP11" evaluates from right to left and to see what it does we will do a view text on it:

```
{ }-----
{ } %vt,EXP11/=
{ } %ii,[2],,[1],(%[2],[1],%EXP11[#3]//)/
{ }
```

This is very similar to the recursive algorithms except that the numbers and functions are user defined in each case. "EXP1" then evaluates from left to right. A view text on it gives:

```
{ }-----
{ } %vt,EXP1/=
{ } %ii,[2],,[1],(%EXP1,%[2],[1],[3]/[#4]//)/
{ }
```

This time it takes the first set of numbers and function and evaluates it then loops back on itself until only a number is left. If you want to use an user defined functions make sure you follow the same format as always. (numberFUNCTIONnumber) If the function only requires one number it must be enclosed in curly braces because no other form of hierarchy is programmed in.

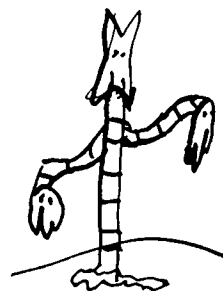
Examples

In this section I would like to just go to the teleprinter and show some examples using this evaluator. Remember everything that has been defined in this section is still here and valid. My main example will be simulating APL sort of. Anyway on with the show:

```

{}-----
{} %dt,FAC,!%ii,q2,,1, (%mu,q2,%FAC,,%su,q2,1//%)////////=
{} %pt,FAC,,q2////=
{} %define,^,exp/=
{} %define,(@!),FAC/=
{} %define,(@#),mrn/=
{} %define,*,mu/=
{} %define,-,su/=
{} %dt,APL,!%ca,%xc,0D//%ut,**/%apl///=
{} %dt,**,!%os,
{} EXIT/%ri///=
{} %dt,apl,!%os,
{}      /%os,
{} %EXP,&IS///%apl///=
{} %APL/=
{}      3+3
{}      6
{}      5+3-2*3\9
{}      2
{}      3#27
{}      3
{}      2^8
{}      256
{}      !100
{}      933262154439441 5268169923885626670049071 59682
{}      6438162146859296389521759999322991 56089414639
{}      761565182862 536979208272237582 511852109168640
{}      000000000000000000000000000000
{}      !200
{}      EXIT
{}
{}-----

```



To make clear how I exited, because it is a little tricky, I defined a user trap that means that when a delete code was hit during a multiply function the text "***" was executed. With a little imagination one can get closer and closer to real APL but never all the way there because functions don't come before multiply and divide and so on. However let me see you simulate APL with a different kind of language, especially one that gets a factorial of one hundred with ALL significant figures.

[illegible]

@cn	240	ep	151	lw	214	ra	215	vt	118
@f	238	et	104	mc	110	rep	263	we	181
@n	239	etb	249	md	146	rf	243	wf	244
@t	237	ex	112	mt	109	rfr	235	wfr	236
ab	159	fb	226	mu	130	ri	166	wi	175
ad	128	fc	137	ni	111	rj	247	wl	179
ai	160	fde	138	not	188	rn	252	wr	178
and	187	fde	139	nu	209	rot	189	ws	180
as	161	fdm	140	nud	268	rp	248	wx	176
bf	220	fe	141	or	186	rr	165	wy	177
ca	113	ff	142	os	101	rs	163	xal	156
cfc	195	fl	143	ot	154	sar	158	xc	170
cin	193	fp	145	pc	108	sda	260	xcf	273
cld	148	fr	144	pl	174	sdu	223	xd	172
cll	191	ft	106	ps	162	sem	199	xi	255
cnb	133	ftb	210	pt	107	sf	222	xj	123
cpc	266	fts	211	qcs	227	sfd	157	xll	155
crd	147	hc	212	qdu	225	sfe	221	xo	256
cro	203	hm	150	qfc	196	sh	190	xqf	272
ct	132	hp	149	qfe	219	sic	245	xqs	254
cwc	250	ht	114	qfs	217	sio	265	xr	119
cws	261	ic	115	qin	194	soc	246	xrp	121
cx	171	id	116	qio	264	srn	253	xrs	229
cxb	200	idt	153	qld	197	sti	258	xrs	269
da	259	ig	136	qll	192	su	129	xu	271
di	131	ii	135	qnb	134	sw	231	xw	120
dif	233	im	117	qof	202	sy	232	xwp	122
dof	234	is	102	qp	167	tb	127	xws	230
dq	208	it	152	qpc	267	ti	257	xws	270
ds	164	iw	213	qrd	198	tm	125	yt	126
dt	103	lf	216	qro	204	tma	124	zd	182
dx	173	lff	228	qta	205	tr	168	zi	183
ea	206	lic	241	qwc	251	trs	274	zq	184
ed	207	loc	242	qws	262	uf	218	zs	185
ef	224	lt	105	qxb	201	ut	169		

Copyright (c) 1978, SAM76 Inc. - Pennington, N. J., U.S.A.

Permission is herewith granted to copy all or part of this section, on Function definitions, for any purpose providing that:

1. This notice be included with each copy made, and
2. If said copy is incorporated in a larger work, then two copies of such work be furnished gratis to SAM76 Inc.

SAM76 Language Functions

The SAM76 language functions described in the following pages represent the second draft for test and evaluation purposes, and include all of the functions available in processors for that language coded for the INTEL 8080 and Zilog Z-80 microprocessors.

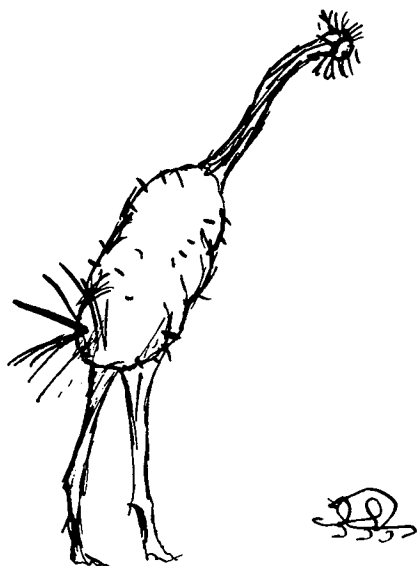
Not all functions are available in all SAM76 language processors, a number of functions relate to disk systems, or are particular for specific machines. Generally this is done through the use of "sub-functions" of a defined function; subfunction definitions are given in documents associated with particular systems.

All functions defined herein properly represent the action expected for processors which have the <ncra> identification; see function [n].

The references listed below should be used to provide information as to language syntax, method of use, and applications:

References

1. "SAM76 - A language based on "Strachey's GPM and Mc Ilroy's M6", by Claude A. R. Kagan, dated august 1976, and available as report R 76-301 from the IEEE Computer Society, 5555 Naples Plaza, Long Beach, CA, 90803.
2. "SAM76 Language Manual"
3. Various notices and updates published in "Dr. Dobb's Journal of Computer Calisthenics and Orthodontia".



Notation used in list of functions

function,arg1,arg2,...,argn	Active form of execution
\function,arg1,arg2,...,argn\	Neutral form of execution
@	"at" (wh@ means "what")
c2,c1, ... ,c	Single characters
s2,s1, ... ,s	Character strings
s0	Prefixing character string
t2,t1, ... ,t	Names of Texts
f2,f1, ... ,f	Names of Files
d2,d1, ... ,d	Decimal Numbers
n2,n1, ... ,n	Numbers in current radix
x2,x1, ... ,x	Binary numbers (octal/hex)
vz	Value if function fails
vt	Value if function True
vf	Value if function False
v0	Value if function zero
v+	Value if function positive
v-	Value if function negative
dev	Device
f or filename	Filename.Extension
a2,a1, ... ,a	Abbreviations
xn1,yn1, ... xn,yn	Increments in X and Y
con	Condition
EQ	Equal
GT	Greater Than
LT	Less Than
ZE	Zero
GE	Greater than or Equal
LE	Less than or Equal
UC	Upper Case
LC	Lower Case

The number of characters in "Filename" and "Extension" is a function of the operating system. Usually the Filename is 6 to 8 characters, and Extension allows for three characters.

When a radix greater than 10 is used upper case alphabetic symbols represent the ensuing integers starting with "A" being equivalent to 10 in base 10, "B" to 11, and so on. Lower case characters are used when the radix exceeds 36.

Function Definitions

101 - |os,s| Output String

This function causes the selected output device to output the string of characters symbolized by "s".

102 - |is,dev| Input String

The value of this function is a stream of characters coming from the currently selected input device; termination occurs as soon as the currently specified "activating character" is encountered; this activating character does not become part of the value string of the function.

103 - |dt,t,s,d1,d2| Define Text (superseding)
 \dt,t,s,d1,d2\ Define Text (save current)

This function is used to create a "text" whose name is denoted by "t", containing the string of characters symbolized by "s"; arguments denoted by "d1", "d2" are for undefined purposes.

The active form of the expression serves also to erase the text whose name is "t" if currently in the text area; the neutral form will not cause any text deletion.

104 - |et,t1,t2,...,t| Erase Text
 \et,t1,t2,...,t\ Erase all occurrences of Text

This function is used to erase from the text area texts whose names are denoted by the arguments of the expression; the active form of the expression erases only the latest version of the named texts, should more than one version exists in the text area.

105 - |lt,s0,d1,d2,...,d| List Texts

The value of this function is a list of the names of the texts to be found in the text area; each name is preceded by the string symbolized by "s0".

Arguments symbolized by "d1", "d2", ... have not been defined.

106 - |ft,t,s1,s2,...,s| Fetch Text

The value of this function is the contents to be found in the text named "t" to the right of its internal text divider.

Any partitions found in that portion of the text are replaced with the strings denoted "s1", "s2", ... "s", in such a manner that partitions with value "1" are replaced by "s1", value 2 with "s2" and so forth.

107 - |pt,t,s1,s2,...,s| Partition Text all matches
 \pt,t,s1,s2,...,s\ Partition Text next match

The purpose of this function is to replace in the text whose name is denoted by "t" the strings symbolized by "s1", "s2", ... "s" with partitions identified by numbers which correspond to the position of the strings in the expression; that is to say "s1" is replaced by a partition value "1", "s2" by partition value "2" &c.

In its active form, this function serves to replace all occurrences of the desired strings; in its neutral form only the first occurrence found is so replaced for each of the specified strings.

Action takes place starting at the current location of the internal text divider which is not moved by this function.

A particular example of interest might be:

&pt,animal,dog,dog,dog,dog/=

This example would in the text whose name was animal, replace the first match with "dog" with partition value "1", the second, "2", the third with "3" and the fourth with "4".

108 - |pc,d| Partition Character

The value of this function is a "partition character" of value symbolized by the decimal number "d".

109 - |mt,t,s1,s2,...,s| Multi-part Text all matches
 \mt,t,s1,s2,...,s\ Multi-part Text next match

This function serves to replace in the text whose name is denoted by "t" the strings symbolized by "s1", "s2", "s" with multi-partitions of values respectively "1", "2", ...; the neutral form of this function only replaces the first occurrence of these strings found in the text.

110 - |mc,d| Multi-partition Character

The value of this function is the "multi-partition character" whose value is denoted by the decimal number "d".

111 - {ni,vt,vf} Neutral Implied

This function returns the string symbolized by "vt" if the last previously executed implied "fetch text" was neutral. If that last previously executed implied "fetch" was active, then the value of this function is that string symbolized by "vf".

112 - {ex,f} Exit

This function is used to exit from the processor back to a system monitor; if the expression contains any arguments, such as "f" then a file is created in auxiliary storage which contains all of the user work space including variables, working area and text area.

This file is given the name denoted by "f", and may be reloaded on any other system and the action interrupted by the execution of this function is then resumed at the same point.

Successful resumption of action is subject to compatibility as determined by verification of the system authorship and version number.

113 - {ca,s} Change Activator (current)
 \ca,s\ Change Activator (initial)

The purpose of this function is to change the symbol which terminates input from its current value to that denoted by the first character of the string denoted by "s"; active form is used to change the operational activating character whereas the neutral form is used to change the initial table.

114 - {ht,t} Hide Text
 \ht\ Hide all Texts

The purpose of this function is to put a screen or fence in the text area such that only those texts created after that whose name is denoted by "t" are available.

This fence is removed by execution of this function with no arguments; if it is desired to conceal all of the text area, the neutral form is to be executed.

115 - {ic} Input Character

The value of this function is one character the source of which is the currently selected input device; this may be absolutely any one single character in the system character set.

116 - |id,d| Input "D" characters

The value of this function is a stream of characters from the current selected input device; the action is terminated when that number symbolized by "d" has been received; in the case of certain sources, an End of File will also terminate.

117 - |im,s1,s2,...,s| Input to Match

The value of this expression is a stream of characters coming from the currently selected input device; termination of this stream occurs as soon as one of the strings symbolized by "s1", "s2", ... "s" has been encountered; the value includes the string that caused termination.

118 - |vt,t1,t2,...,t| View Texts

This function enables examination of defined texts, indicating visually the location of internal text divider, partitions and their values, as well as multipartitions.

119 - |xr,x| eXamine Register

The value of this function is the contents of the memory location symbolized in the current "X" base by "x".

120 - |xw,x1,x2| eXperimental Write in reg.

Use of this function causes writing the value "x1" into the memory location "x2"; both of these are in the current "X" base.

121 - |xrp,x| eXamine Register Pair

The value of this function is the combined contents of the two memory locations, whose first location is symbolized in the current "X" base by "x".

122 - |xwp,x1,x2| eXperimental Write reg. Pair

Use of this function causes writing of the double word data symbolized in the "X" base by "x1" in the memory location whose first word is identified by "x2".

123 - |xj,x| eXperimental Jump

Execution of this function cause a jump to the memory location symbolized in the "X" base by "x"; a return from programs encountered at that location will cause normal reentry into the system scanner.

124 - |tma| Trace Mode All activated
 \tma\ Trace Mode All deactivated

Active execution enables full trace action; neutral execution allows display only of expressions about to be executed.

125 - |tm,d| Trace Mode activated
 \tm\ Trace Mode deactivated

This function is used to enable step by step execution of the scripts and functions of the language; once activated each step is initiated by depression of the "new line" key ("line feed") or a specified number of steps may be specified by the number symbolized by "d".

126 - |yt,t,s,vt,vf| Ys There

This function searches the text whose name is denoted by "t" for an exact match with the string symbolized by "s"; if such a match is found, then the value of the expression is that string symbolized by "vt", else it is that string symbolized by "vf".

127 - |tb,t,vt,vf| Text Branch

If the text whose name is denoted by "t" is to be found in the text area, then the value of the function is that string symbolized by "vt", else the value is string symbolized by "vf".

128 - |ad,n1,n2,n3,...,n| Add

The value of this expression is the arithmetic sum of the integer numbers denoted by "n1", "n2", ... "n". Each of the arguments is examined from right to left until a symbol not representing a valid number in the current number base is encountered; non numeric symbols prefixing the arguments are ignored with the exception of those in argument "n1" which is carried into the value string of the expression. For example:

```
%ad,dog5,cat6/={dog11}
```

Negative numbers are indicated in the usual manner with a "-" sign which must immediately precede the string of integers it applies to thus:

```
%ad,dog-5,cat6/={dog1}
```

Ommitted arguments (such as adjacent commas) have the value of the "null string" and a numeric value of zero (0) as illustrated in the following example:

```
%ad,dog,/={dog0}
```

[Note: multiple arguments not supported in all processors]

129 - |su,n1,n2,...,n| Subtract

The value of this function is the result of subtracting the number symbolized by "n2" from that symbolized by "n1"; all rules indicated for sign and non numeric matter in the "ad" function apply.

130 - |mu,n1,n2,vz| Multiply

The value of this function is the arithmetic product of the numbers symbolized by "n1", and "n2". The same rules as to sign and non numeric matter apply for this function as they do for the "ad" function.

Should an overflow condition arise as a result of the execution of this function, then the value of the expression is that string symbolized by "vz".

131 - |di,n1,n2,vz| Divide

The value of this function is the integer result of dividing the number denoted by "n1", by the second number denoted by "n2"; if the result of the division would be indeterminate - "n2" = zero - then the value of the expression is that string denoted by "vz".

Treatment of non numeric, signs, and null strings is identical to that described in the "ad" function.

132 - |ct,t1,t2,t3,...,t| Combine Texts (superseding)
 \ct,t1,t2,t3,...,t\ Combine Texts (save current)

The purpose of this function is to create a text, whose name is denoted by "t1" which will contain the "concatenation" of the texts whose names are denoted by "t2", "t3", ... "t".

The active form of the expression will also delete the current text named "t1" if any; the neutral form of the expression will not cause deletion of any pre-existing texts.

133 - |cnb,d| Change Number Base (active)
 \cnb,d\ Change Number Base (initial)

The purpose of this function is to change the radix of the processor's arithmetic functions; this is always expressed as the decimal number symbolized by "d" in the expression; see [qnb]; active form of the expression changes the value of the radix used during the actual operations while the neutral form changes the initial table. Note that the neutral change can only be effective while the processor is in unprotected memory, and will have no effect if the processor is executed from read only or protected memory.

134 - |qnb| Query Number Base

The value of this function is the current value of the radix for arithmetic operations and functions.

135 - |ii,s1,s2,vt,vf| If Identical

Strings symbolized by "s1", and "s2" are compared, if found to be absolutely identical, then the value of the expression is that string symbolized by "vt"; if not identical the value is the string symbolized by "vf".

136 - |ig,d1,d2,vt,vf| If Greater

This expression compares in an arithmetic sense the two strings denoted by "d1" and "d2"; if the value of "d1" is greater than that of "d2", then the value of the expression is that string symbolized by "vt", else the value is that string symbolized by "vf".

137 - |fc,t,vz| Fetch Character

The value of this function is the character to be found immediately to the right of the internal text divider of the text whose name is denoted by "t"; if such a character exists then the text divider is advanced to point to the next available character - or alternatively to the end of the "text".

If no character is available then the value of the expression is that string denoted by "vz" which is always treated actively.

138 - |fdc,t,d,vz| Fetch "D" Characters

The value of this function is that decimal number of characters denoted by "d" which is to be found to the right or left of the internal divider of the text whose name is denoted by "t"; a positive number specifies to the right, and a negative number to the left; the text divider is then relocated to point to the next character available if to the right, or to the first character returned in the value string if to the left.

Should there be absolutely no characters (not even one) available for the value, then "vz" represents the string which will be returned as the value of the expression, and this value is always treated actively.

139 - |fde,t,d,vz| Fetch "D" Elements

The value of this function is that string of characters to the right or left of the text divider of the text whose name is denoted by "t" which comprises the number of elements specified by the decimal number "d"; an element is that which is found between partitions.

The divider is then moved either to the right or left to point either to the next character to be read (if to the right) or that character which corresponds to the first character of the value string (if to the left).

If absolutely no characters are returned, then the divider is not moved and the value of the expression is that string symbolized by "vz" which is always treated actively.

140 - |fdm,t,d,s,vz| Fetch "D" Matches

The value of this function is a character string taken to the right of the text divider of the text whose name is denoted by "t"; the contents of the text "t" is scanned searching for occurrences of the string denoted by "s"; when that number of occurrences equal to the number "d" is found then the divider is relocated to point to the next immediately following character, and the value string is that which lies between the old and the new divider locations.

If the required number of matches is not found, then the divider is not moved, and the value of the expression is that string denoted by "vz", which is always treated actively.

If the desired number is negative, then the action takes place to the left of the divider instead of to the right. If "d" is a negative number, then the search is made to the left of the divider.

141 - |fe,t,vz| Fetch Element

The value of this function is that string of characters which is to be found immediately to the right of the current location of the divider in the text named "t" up to the next encountered partition - or end of text.

The text divider is moved to point to the immediately available next character.

If absolutely no characters are to be found, then the value is the string denoted by "vz", which is always treated actively.

142 - |ff,t,d,vz| Fetch Field

The value of this expression is that string of characters to be found to the right of the first occurrence of a partition of value symbolized by "d" until the next partition is encountered, or to the end of the text named "t"; the divider is not moved.

If absolutely no characters are returned the value of the expression is the string symbolized by "vz" which is always treated actively.

143 - |fl,t,s,vz| Fetch Left match

The text named "t" is searched to the left of its internal divider for a string identical to that denoted by "s"; if such a string is found then the divider is moved to point to the first character of that string and the value of the expression is that group of characters to be found between the new location of the divider and its old location.

If no such matching string is found, then the divider is not moved, and the value of the expression is the string "vz" which is always treated actively.

144 - |fr,t,s,vz| Fetch Right match

The text named "t" is examined starting at the current location of the text divider for a string identical to that denoted by "s"; if such a string is found, then the divider is moved to the next character which follows this string (or the end of the text) and the value returned is that string of characters which lies between the old and the new locations of the text divider.

If no such match is found, then the value of the expression is the string denoted by "vz", always treated actively, and the divider is not moved.

145 - |fp,t,x1,...,x| Fetch Partition

The value of this function is the next partition to be found in the text whose name is denoted by "t"; the text divider is moved to point to the next ensuing character.

146 - |md,t,d| Move Divider to pos. "d"
 \md,t,d\ Move Divider "d" increments

This function is used to move the text divider in the text whose name is denoted by "t". Active form serves to move the divider the number of positions from the left end of the text if "d" is positive, or from the right end if negative; the neutral form is used to advance or retreat the divider from its current location that number of positions specified by the number "d".

It should be noted that these moves are positional and do not distinguish any difference between partitions and characters; this is the only function available to position the text divider other than immediately preceding a character.

147 - |crd,t| Characters Right of Divider

The value of this function is the (decimal base) number of characters located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

148 - |cld,t| Characters Left of Divider

The value of this function is the (decimal base) number of characters located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

149 - |hp,t,d| How many Partitions

The value of this function is the number of partitions to be found to the right of the text divider in the text whose name is "t"; if there is no explicit third argument, then the value represents the total number of partitions; if argument three is specified, then the value is that number of partitions whose value is equal to the number symbolized by "d".

150 - |hm,t,s| How many Matches

The value of this function is that number of occurrences of the string of characters symbolized by "s" in the text whose name is denoted by "t".

151 - |ep,t,p1,p2,...,p| Erase Partitions

This function serves to delete from the text named "t" any partitions that may be found in it to the right of the internal text divider.

If arguments "p1", "p2", and so on are specified, then only those partitions with values denoted by these arguments are deleted.

The value of this function is a list of internal system labels available for examination by name; these labels may be actual values, identify locations of program entry points or tables, or give addresses of memory locations in which system pointers may be found. Each label name is preceded in the value string by the string symbolized by "s0".

Typical system labels as determined by the "xll" function are given below with their characterizations:

AST	Address System subroutine Table
ASV	Address System subroutine Vectors
ENC	Entry Continue ("hot start")
ENR	Entry Restart ("warm start")
ESL	Entry Subroutine Links
INP	Address Input Vector
OPV	Offset Permanent/Variable tables
OUT	Address Output Vector
POA	Pointer Overlay A
POB	Pointer Overlay B
POC	Pointer Overlay C
POD	Pointer Overlay D
PSE	Pointer Service Entry
PSW	Pointer Function "sw"
PSY	Pointer Function "sy"
PUI	Pointer User Input routine
PUL	Pointer User Link table
PUO	Pointer User Output routine
PWA	Pointer Work Area start
TAE	Text Area End
TAH	Text Area High
TAL	Text Area Low
TAM	Text Area Maximum

156 - |xal,label,offset| Xamine Address of Label

The value of this function is the value of the "label" to which the value of "offset" has been added; both the value returned and the "offset" are in the current "X" base.

157 - |sfd,fun,dev| Specify Function Device

To be defined and coded.

158 - |sar| "Auto Return" on line feed
 \sar\ no Auto Return on line feed

Execution of this function serves to enable (if active) or disable (if neutral) the automatic generation of a "carriage return" code when a "new line" code is output.

159 - |ab,s1,s2,vt,vf| Alphabetic Branch

This function compares the character strings denoted by "s1" and "s2" in terms of their "ASCII" binary value; if the string "s1" has a greater value than the string "s2" then the value of the expression is the string denoted by "vt", else the value is the string denoted by "vf"; if the value strings are expected to contain executable or syntactically meaningful symbols they should be protected within the expression.

160 - |ai,s0,s1,s2,...,s| Alphabetic Insertion

The value of this function is a character string in which the string denoted by "s1" has been inserted ahead of the first of the subsequent argument string which has a greater alphabetic value; each of the strings which form the arguments of the expression are returned with a copy of the string symbolized by "s0" preceding it.

161 - |as,s0,s1,s2,...,s| Alphabetic Sort

The value of this function is a character string in which the sequence of the strings denoted by "s1", "s2", ... "s" is rearranged to be in proper alphabetic sequence; each of these strings is preceded by the string denoted by "s0" in the result:

%as,(,),horse,cat,dog/={,dog,cat,horse}

Note the use of "(,)" for "s0"; this is typical of lists that the user may wish to use in a variety of functions which have arbitrary number of arguments.

162 - |ps,d,s1,s2| Pad String

The value of this function is the string "s2" to which is added either to its right or to its left enough duplications of the first character of string "s1" so that the total size of the value string becomes equal to the number symbolized by "d"; positive values of "d" pad to its right, and negative values to its left.

163 - |rs,s| Reverse String

The value of this function is the string of characters symbolized by "s" reversed, end for end.

164 - |ds,d,s| Duplicate String

The value of this expression is a concatenation of that number of copies of the string "s" which is denoted by the decimal number "d".

This function may be quite properly used to "DO" something a desired number of times as illustrated below:

```
%ds,5,!%dt,x,%ad,3,%x/////=
```

This example shows a most awkward way of creating a text named "x" which contains the product of 5 and 3.

165 - |rr,s| Return to Restart

This function causes immediate cessation of execution of any unexecuted script, forces a return to the idling level of the processor and then returns the value of the string symbolized by "s1"; if that string is an executable expression, then said expression is executed.

166 - |ri| Restart Initialized

Execution of this function causes immediate termination of any unexecuted scripts, and causes return to the idling condition, reinitializing all normally changed user specifications.

167 - |qp,t| Query Partition

The value of this function is the decimal identification of the next partition to be found in the text whose name is denoted by "t"; the text divider is not moved by this function.

168 - |tr,t,s| Trim

This function is used to replace multiple adjacent occurrences of the character symbolized by "s" with a single such occurrence in the text named "t".

169 - |ut,cc| User Trap active
 \ut\ User Trap inactive

Activating of this trap through use of this function will cause an automatic execution of a text (or function) whose name is denoted by "cc"; this automatic execution takes place whenever the user attempts to make use of system interrupt capabilities to escape out of executing "scripts" when it is desired that such escape be inhibited.

170 - |xc,x1,x2,...,x| "X" to Character

The value of this function is a string of characters whose value is represented by the numbers in "X" base symbolized by "x1", "x2", ... "x".

171 - |cx,s0,s| Character to "X"

This function returns as a value a sequence of binary numbers, each preceded by the string denoted by "s0" which result from converting the character string denoted by "s"; the conversion is in the current specification for the "X" base; see [cxb, qxb, xc].

172 - |xd,x| "X" to Decimal

The value of this function is the decimal equivalent of the number in the current value of the "X" base symbolized by "x".

173 - |dx,d,x| Decimal to "X"

The value of this function is that number which results from converting the decimal number denoted by "d" to its equivalent in the current "X" base; the argument symbolized by "x" is undefined; see [xd, cxb, qxb].

174 - |pl,sub,s1,...,s| Plot

This is a general purpose plotter or display control function; details may vary in its use as a function of the system in which it is used. Certain subfunctions as described below have been used:

-01 pl,lf,s0	List of Plotter subfunctions
-02 pl,as,port,time	Assign output port and time delay
-03 pl,nm	Output plotter control bytes
-04 pl,vm	Return as value control bytes
-05 pl,cal	Set up for "Calcomp" plotter
-06 pl,pol	Set up for "Polymorphic" display
-07 pl,trs	Set up for "TRS-80" display
-08 pl,mq,x	Rotate to quadrant "x".

175 - |wi,xn1,yn1| Write Initialize

This null valued function serves to create a reference value for the current position of the display pointer; the "x" and "y" coordinates of this position are set equal to the decimal integer strings found in the second and third arguments respectively.

176 - |wx| Write "X" displacement

The value of this function is equal to the algebraic difference between the initial value "X" of the display pointer position and the sum of any subsequent incremental displacement values.

177 - |wy| Write "Y" displacement

The value of this function is equal to the algebraic difference between the initial value "Y" of the display pointer position and the sum of any subsequent incremental displacement values.

An effective way to return the display pointer to its original location is illustrated as follows:

```
{ } ~~~~~
{ } %WS,U%WX/,%WY//
{ }
```

178 - |wr| Width Right

The value of this function is the incremental value of the width of a character being plotted to the right of its center of gravity.

179 - |w1| Width Left

The value of this function is the incremental value of the width of a character being plotted to the left of its center of gravity.

180 - |ws,xn1,yn1,...,xn,yn| Write Straight Lines

This function causes the generation of the appropriate control information to cause the display of one or more line segments starting at the current position of the display pointer. Successive points are referenced by consecutive incremental x and y argument pairs. The x and y components are expressed as decimal integer strings. The sign of the number indicates direction.

When in visual display mode, the neutral form of this function serves to erase any line or points which coincide with the path of the lines drawn.

Non numeric matter preceding the "x" value causes certain auxiliary functions to take place as follows:

U	"pen up" or invisible vector
S	Scale change for "x" and "y"
A	Absolute position vector pair
I	Incremental vector pair
C	Clear display screen and initialize absolute X & Y locations
Q	Quadrant rotation 0 - 3 clockwise, 4 - 7 counter clockwise
W	Character width information "x" left and "y" right of center

```
181 - |wc,s1,s|      Write Characters
```

This plotter or display function generates a graphic display from a font of vectors named "s1"; the string symbolized by "s" is the text string to be plotted or displayed.

182 - |zd,r,v-,v0,v+| "Z" reg. Decrement and branch

Execution of this function causes the contents of a special register identified by "r" to be decreased by a count of one; if the resulting contents of that register are greater than zero, then the expression value is that string symbolized by "v+"; if equal to zero, then the value is symbolized by "v0", and if less than zero the value becomes the string symbolized by "v-".

183 - |zi,r,v-,v0,v+| "Z" reg. Increment and branch

The action of this function is identical to that described for "zd", except that the designated register "r" is increased by a count of one prior to testing.

184 - |zq,r| "Z" reg. Query

The value of this function is the current contents of the special register designated by "r"; execution of this function does not cause any change in said register.

185 - |zs,r,n| "Z" reg. Set

This function is used to preset designated special register "r" to the number symbolized by "n".

186 - |or,x1,x2| Or the bits

The value of this function is the logical or of the two binary number strings "x1" and "x2" expressed in the current "X" base.

187 - |and,x1,x2| And the bits

The value of this function is the result of performing an anding of the two binary numbers represented in the current "X" base by "x1" and "x2".

188 - |not,x| Not (complement) the bits

The value of this function is the complement of the binary number (in the current "X" base) symbolized by "x".

189 - |rot,d,x| Rotate the bits

The value of this function is that binary number which results from a rotation of the binary number "x" expressed in the current "X" base; the number of bits rotated is specified by the decimal number "d"; the direction is clockwise if the number is positive, and counterclockwise if negative.

190 - |sh,d,x| Shift the bits

The value of this function is that binary number string which results from a logical shift to the right or left of the string whose binary value, expressed in the current "X" base is symbolized by "x"; the number of shifts is specified by the decimal number "d" and the direction by its sign, positive to the right and negative to the left.

191 - |cll,d| Change Line Length (active)
 \cll,d\ Change Line Length (initial)

The purpose of this function is to change the number of characters that the processor will output before automatically inserting a "new line" code; the desired value is the decimal number denoted by "d" in the expression; the active form changes the actively used constant, while the neutral form changes the initial condition table; see [cnb].

192 - |qll| Query Line Length

The value of this function is the current specification for the line length; this specification is changed through use of the "cll" function.

193 - |cin,t1,d1,...,t,d| Change Id Number

To be defined and coded.

194 - |qin,s0,t1,t2,...,t| Query Id Number

To be defined and coded.

195 - |cfc,d1,s| Change Fill Character schema
 \cfc,d1,s\ Change Fill Char. (initial)

The purpose of this function is to enable the user to specify the number of that character which is the first one of the string denoted by "s" that will be automatically output at the beginning of each "new line"; usually these will be "nulls" but they may be any other character the user wishes to use. Active form of the expression changes the table in active use, while the neutral form changes the initial table; see [cnb].

196 - |qfc,s0| Query Fill Character schema

The value of this function is the current specification for the optional fill character output immediately after each "new line"; the parameters are each preceded by the string symbolized by "s0"; see [cfc].

197 - |qlld,t| Query Left of Divider

The value of this function is the (decimal base) number of positions - characters and partitions - located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

198 - |qrd,t| Query Right of Divider

The value of this function is the (decimal base) number of positions - characters and partitions - located between the beginning of the "text" whose name is denoted by "t" and the current location of the internal "text divider" of said text.

199 - |sem,dev| Set "Echoplex" Mode active
 \sem,dev\ "Echoplex" Mode inactive

This function enables character echo from the system if executed actively; if neutrally, then echo generation is suppressed.

200 - |cxb,d| Change "X" Base (active)
 \cxb,d\ Change "X" Base (initial)

Through use of this function, the user may change the specification of the radix (or base) for those functions which operate in the "X" base; "d" symbolizes the decimal value of the desired new radix, and this should normally be either "2" or "4" or "8" or "16"; active and neutral forms operate with respect to the "X" base in the same manner as described for the "cnb - change number base" function.

201 - |qxb| Query "X" Base

The value of this function is the currently set radix for the functions which operate in a "binary" or "X" base.

202 - |qof| Query Over Flow conditions

The value of this function are characters which may be used to identify the cause of an interrupt or overflow condition.

203 - |cro,s1| Change Rub Out char. schema
 \cro,s1\ Change Rub Out (initial)

The purpose of this function is to permit the user to specify as the first character of the string denoted by "s1" that symbol desired to serve the character delete function; in addition the user may specify the next two characters of that string to be output before and after, respectively, the deleted string of characters. The active form changes the user table, while the neutral form of this function changes the initial table; see [cnb].

204 - |qro| Query Rub Out char. schema

The value of this function describes the current specification for the use of the "rub out"; see [cro].

205 - |qta| Query Text Area used

The value of this function is representative of the amount of space consumed in the text area by user defined texts.

206 - |ea,t1,t2,...,t| Erase All excepting

This function is used to erase from the text area all defined texts; excepted from this erasure are the texts whose names are denoted by "t1", "t2", ... "t"; full erasure is accomplished as shown below:

%ea/=

[Note: Multiple arguments not supported in all systems]

207 - |ed,t,d1,d2,vz| Extract "D" characters

The value of this function is a string of characters extracted from the text whose name is denoted by "t"; the first character returned is that which is found distant to the right of the text divider by the number denoted by "d1", and the number of characters returned is denoted by the number "d2".

Note that "0" represents the location of the divider itself, and that "d1" may be a negative number.

If there are absolutely no such characters available then the value of the expression is that string denoted by "vz" which is always treated actively; this is regardless of whether the original expression was used actively or neutrally.

208 - |dq,s| Define Quote

The purpose of this function is to enable the user to specify the character that is to be recognized by the scan algorithm as an unconditional protecting character. To be further defined.

209 - |nu,s1,s2,...,s| Null

The purpose of this function is to cause execution of the function whose name is denoted by "s1", with appropriate arguments "s2", ... "s"; that function is executed but any resulting value string is suppressed.

210 - |ftb,t,s,vz| Fetch To Break character

The value of this function is a string of characters taken from the current location of the internal text divider of the text whose name is denoted by "t" to the first ensuing character in text "t" which is found in the string symbolized by "s"; the text divider is moved to the first character which follows that which terminated the scanning action.

If no character is returned, then the divider is not moved and the value of the expression is that string symbolized by "vz" which is always treated actively.

211 - |fts,t,s,vz| Fetch To Span character

The value of this function is a string of characters taken from the current location of the internal text divider of the text whose name is denoted by "t" to the first ensuing character in text "t" which is not found in the string symbolized by "s"; the text divider is moved to point to the first character which follows that which terminated the scanning action.

If no character is returned, then the divider is not moved and the value of the expression is that string symbolized by "vz" which is always treated actively.

212 - |hc,s| How many Characters

The value of this function is the actual number of characters comprising the string symbolized by "s".

213 - |iw,n| Input Wait

This function sets up an internal timer which becomes effective for the next ensuing input function, such as "is", "ic", "idc", "im"; automatic termination of the input process takes place at the end of the specified time interval even though normal terminating conditions specified for these functions may not have been met.

The duration of the time interval is specified by "n" which is in seconds.

214 - |lw,s0,s1,s2,...,sn| List Where

The value of this function is a list of the names of the texts in the text area which contain at least one occurrence of each of the strings symbolized by s1,s2, . . . sn; each name returned in the value string is preceded by the string symbolized by s0.

It should be noted that the name of the text itself is part of the text and that the search through the text area by this function will also return text names even if the only match is in the name itself.

215 - |ra,d,s1,s2,s3,...,s| Return Argument

The value of this function is that string symbolized by one of s1", s2", "s3", ... "s" whose position corresponds to the value of the decimal number symbolized by d".

216 - |lf,s0,d1,...,d| List Files

The value of this function is a list of the names of the files in auxiliary storage; each file name is preceded by the string symbolized by "s0".

Additional arguments are to be defined.

217 - |qfs,filename| Query File Size

The value of this function is the file size as a decimal number of records.

218 - |uf,f,t1 t2,...,t| Update File

This function combines the action of "erase file" and "store file" by first storing a file successfully before erasing the old file version.

219 - |qfe| Query File Extension

The value of this function is the current setting of the "extension" to the file name; the extension is that part of the name to the right of a period embedded in the file name, where this period is not part of said name. Sometimes the extension is known as a "file type".

220 - |bf,f,vz| Bring File

Execution of this function serves to bring from auxiliary storage the file whose name is denoted by "f"; if this file does not exist, then the value of the function becomes the string symbolized by "vz".

221 - |sfe,extension| Set File Extension

This null valued function is used to preset a desired file extension so that the extension need not be used in any of the file functions as part of the file name. In effect this may be used as a classification means. The extension may be set to any desired three (or less) symbols, including spaces and ???, or to the null string (viz: %sfe/) with varying effects.

222 - |sf,f,t1,t2,...,t| Store File

This function is used to place into auxiliary storage under the file name denoted "f", those texts whose names are denoted by "t1", "t2", ... "t"; on completion of this action, the named texts are erased from the text area.

If no texts are named, then the assumption is made that the entire text area is to be placed into auxiliary storage.

223 - |sdu,dir| Select Directory Unit

This null valued function allows the user to designate the "Directory" unit to be accessed by the various file functions. A "directory" unit is any device which has a directory structure and could be cassette as well as disk.

In a CP/M system, the "dir" is a number in "X" base.

224 - |ef,f1,f2,...,f| Erase Files

This null valued function is used to erase from auxiliary storage the files whose names are denoted by "f1", "f2", ... "f".

225 - |qdu| Query Directory Unit

The value of this function is the currently selected "directory" unit.

226 - |fb,f,vt,vf| File Branch

The value of this function is the string symbolized by "vt" if the file whose name is denoted by "f" is to be found in external storage; if not the value of this function is that string symbolized by "vf".

227 - |qcs| Query Command String

The value of this function is that string of characters entered after the name which brought the SAM76 language interpreter into action from the operating system.

228 - |lff,s0| List File Functions

The value of this expression is a list of the available file functions; each function mnemonic is preceded in the value string by the string symbolized by "s0".

229 - |xrs,unit,track,sector,s0| X Read Sector

The value of this function is the contents, in the current "X" base of the identified sector. Each "byte" in the resulting value string is preceded by the string symbolized by "s0".

230 - |xws,unit,track,sector,X| X Write Sector

This null valued function places the string symbolized by X in the designated sector. If that string is greater in length than one sector, then only as much as will fit will be stored.

In the foregoing expression "X" represents a list of numbers in "X" base where each number in the list is preceded by the current argument separator (usually a comma).

231 - |sw,s1,s2,s3,...,s| Switches

Special system dependent function for user definition.

232 - |sy,s1,s2,...,s| System functions

Special system dependent function for user definition.

233 - |dif,filename| Designate Input File

This function serves to open a file from the currently selected "directory" unit for access through use of the "rfr" function, or via channel control assignment of "FIL" to any of the SAM76 language functions that are designed to accept input.

234 - |dof,filename| Designate Output File

This null valued function is used to open a file for output. Such output is attained through use of the "wfr" write file record function. If the "soc,FIL" function is used, then any SAM76 language functions which delivers output will go to the file.

A file is closed through execution of %dof/ with no arguments. Exit using "control C" or "ex" function will not close output files.

235 - |rfr| Read File Record

The value of this function is one record from a file previously opened using the "dif" function. Typically this will be 128 characters, except for the last record of the file which may be shorter.

236 - |wfr,s| Write File Record

This null valued function will write out to a previously properly opened file the string symbolized by "s". There is no limit as to the number of characters in string "s".

237 - |@t| wh@ is processor Title

The value of this function provides information as to the authorship of the processor being used; a typical example is:

&@@t/=<ncra> indicating that the authors are:

"Neil Colvin Claude Roichel Roger Amidon.

238 - |@f,s0| wh@ are Functions

The value of this function is a list of built-in functions available within the system. Each function mnemonic is preceded by the string of characters symbolized by "s0". It is advisable to use the "neutral" form of expression as shown below:

&@@f, /=(Value will be list of functions)

Note use of two "@"; since that symbol is a warning character it must be protected, and in the above example it is used as the single character protector to protect itself.

239 - |@n| wh@ is processor ser. Number

The value returned through execution of this function is the version number of the processor being used. This number is tested by some of the functions - and may also be so tested by the user - to ascertain compatibility between scripts and run time load modules that may be moved between systems and users.

240 - |@cn,current,new| Change function Name

This function allows the user to rename any of the resident function "current" names to any desired "new" name; these new names must consist of two or three alphabetic characters; in changing names the user should be particularly careful in his command sequence. Usually "os" and "is" should not be changed.

241 - |lic,s0| List Input Channels

The value of this function is a list of channel names that may be used in the "sic" select input channel function; each name in the value returned is preceded by the string symbolized by "s0". Typical of these channel names are the following:

CON Console, normally the user keyboard
RDR Reader, paper tape, cassette &c.
FIL A disk file opened with the "dif" function
USR Device, user software driver address at "PUI"

242 - |loc,s0| List Output Channels

The value of this function is a list of the channel names that may be used in the "soc" select output channel function. Each name is preceded with the string symbolized by "s0". Typical symbol names returned by this function are:

CON Normally the user display unit
PUN Punch, Cassette. &c.
FIL A disk file properly opened with the "dof" function
USR Device whose user software address is at "PUO"
LST a Listing device

243 - |rf,filename| Read File

The value of this function is the contents of the file, whose name is denoted by "filename". This file is assumed to be in the normal format used by the host system for USASCII files.

244 - |wf,filename,s| Write File

This function writes out in the normal format used by the host system for USASCII files, a file whose name is denoted by "filename" containing the string symbolized by "s".

245 - |sic,sym| Select Input Channel

This null valued function assigns to the channel whose name is symbolized by "sym" one of the two groups of functions; if this function is executed actively the assignment is of the Group 1 functions if neutrally the assignment is of the Group 2 functions. [see 246 for definition of Group 1 and Group 2 functions].

246 - |soc,sym| Select Output Channel

This null valued function assigns to the channel whose name is symbolized by "sym" one of the two groups of output functions. If this function is executed actively, then the assignment is of the Group 1 functions, if neutrally then the assignment is of the Group 2 functions.

Group 1 functions are those normally initialized to interact with the user console; these are: "is", "os", "ic", "id", "im", "vt", and "tm". Group 2 functions are those normally initialized to interact with the "Reader/Punch" devices; these are: "it", "ot", "idt".

247 - |rj,s,s1,d,s2| Return Justified lines

To be defined and coded.

248 - |rp,s,s1,d,s2| Return Padded lines

To be defined and coded.

249 - |etb,s| Erase Trailing Blanks

This function, which Neil Colvin insisted be made available, serves to delete from the character string symbolized by "s" all blanks which immediately precede the occurrence of "new line" codes; particularly useful when reading punched cards with many trailing blank columns. [Note: Not usually provided]

250 - |cwc,s1| Change Warning Character
 \cwc, ... \ Change Warn. Char. (initial)

The purpose of this function is to allow the user to select other symbols for syntactic purposes than those initially defined in the language; see [qwc]. The active form of this function changes the current user tables while the neutral form changes the initial table; see [cnb].

251 - |qwc,a2,a1,...,a| Query Warning Characters

The value of this function is a list of the currently specified and functional warning characters of the language

252 - |rn,n| Random Number

The value of this function is a number ranging between 0 and "n", randomly selected through a computational algorithm; see [srn].

253 - |srn,n| Seed Random Number

Through use of this function the automatic generation of random numbers is initiated; the user seed number is symbolized by "n" in the above expression.

254 - |xqs,s0| X Query work Space

The value of this function is a set of two numbers in the current "X" base which identify the first and last addresses of the user work space. Each number is preceded with the string symbolized by "s0". The work space so indicated includes all of the variables that may have been set by the user. If this work space is saved, then subsequently reloaded and a "hot" entry is made into the SAM76 interpreter, then whatever process was interrupted at the time of exit and save will be resumed. [See 261/cws and 112/ex].

255 - |xi,port| eXperimental Input

The value of this function is that number in the current "X" base which results from an attempted input from the port whose designation in the "X" base is symbolized by "port".

256 - |xo,x,port| eXperimental Output

Execution of this function enable output of the "X" base value of "x" to the port whose "X" base id is symbolized by "port"

257 - |ti,s1,s2| Time

The value of this function is the current time of day derived from a system clock in the format HH MM SS, where HH represent Hours, MM represent Minutes and SS seconds; these elements are separated in the value string by the string symbolized by "s1".

258 - |sti,t1,t2,t3| Set Time

Through use of this function the correct current (or incorrect if desired) time of day is established in the system.

259 - |da,s0| Date

The value of this function is the current calendar date in the sequence of "day month, and year"; each of these elements is preceded by the string of characters denoted by "s0". The date may be either user entered (using the "sda function) or may be automatically generated by a system clock or calendar; see [sda].

260 - |sda,da,mo,yr| Set Date

This function enables the user to set into the system the current (or any other) desired date.

261 - |cws,d| Change Work Space
 \cws,x\

This function allows a respecification of the upper limit of space required for function execution and text storage; the neutral form is used to allow a binary (hex, octal ...) number denoted by "x" to be entered; the active form uses the decimal number system instead; see [qws]. The use of this function without any arguments serves to reduce the work space to its practical minimum size.

262 - |qws| Query Work Space
 \qws\

The value of this function is the upper limit of the user work space; active form of this expression yields the value as a decimal number; neutral form yields the value in the current "X" base.

263 - |rep,d1,d2,s| Return Character Picture

To be coded.

264 - |qio| Query current IO assignments

The value of this function is in the current "X" base the value of IOBYTE" as defined by the monitor - if such a facility is provided.

265 - |sio,iobYTE| Set IObyte

This null valued function is used to reassign the monitor "IOBYTE" if provided for. It should be noted that a "cold start entry into the processor carries with it the current monitor assignment; if this function is used then a "control C" or "ex" exit causes restoration back to the original assignment. A "warm reentry or a "hot continue causes reassignment to the setting in force at the time of exit.

266 - |cpc,x1,t1,...,tn| Change Protection Class

Execution of this function changes the protection class of texts whose names are symbolized by "t1",... to the "X base value symbolized by "x1".

267 - |qpc,s0,t1,t2,...,t| Query Protection Class

The value of this function is a list of the protection classes of the texts named "t1", "t2", ... , each protection class value is preceded by the string symbolized by "s0".

Protection class of "HEX 80" is reserved for texts which contain machine code; Classes 1 through 8 are for user use. Texts are always given a protection class of 8 at time of definition.

268 - |nud,func,arguments| Null Display mode

The "nud primitive serves to modify the scanner to the end that no value is returned by the function whose name or mnemonic is symbolized by "func" (with its appropriate arguments) Instead the currently assigned console device displays the value.

269 - |xrs,unit,trk,sec,s0| X Read Sector

The value of this function is the contents, in the current "X" base of the identified sector. Each "byte is preceded by the string symbolized by "s0".

270 - |xws,unit,trk,sec,X| X Write Sector

This null valued function places the string of arguments symbolized by "X" in the designated sector. (X=x0,x1, ...xn).

271 - |xu,sub,arguments| Xperimental User

This function, through the use of subfunctions whose mnemonics are symbolized by "sub", with appropriate additional arguments, provides a variety of machine dependent debugging software tools. Explicitly defined are the following:

-01 xu,lf,s0	List "xu" functions
-02 xu,dm,from,to,with	Duplicate Memory
-03 xu,fm,from,to,s0	Fetch Memory
-04 xu,tm,from,to	Type Memory
-05 xu,rl,label	Read contents of "label"
-06 xu,wl,label,Xval	Write in "label" "Xval"
-07 xu,mm,from,until,to	Move Memory
-08 xu,vm,from,till,from	Verify Memory
-09 xu,dt,t,x1,x2,..	Define Text to contain x1
-10 xu,ft,t,s0	Fetch text "X" contents
-11 xu,sum,from,to	Check Sum from to incl.

272 - |xqf,s| eXperimental Query Function

The value of this function is the machine adress in "X" base radix of the entry point for the function whose mnemonic is denoted by the string "s".

273 - |xcf,s,x| eXperimental Change Function

This function is used to assign a user defined machine adress symbolized by the "X" base number "x" for the built in function whose mnemonic is denoted by "s".

274 - |trs,sub,arguments| TRS-80 Computer Functions

This function provides access to a number of subfunctions whose mnemonics are symbolized by "sub" above. A list of typical such subfunctions particularized for the TRS-80 is given below:

-01 trs,lf,s0	List Functions
-02 trs,bof,f	Bring Overlay File
-03 trs,idt,d	Input 'd' Texts
-04 trs,it	Input Text
-05 trs,qfl	Query Flags
-06 trs,rcv,f,x1,x2,x3	Receive
-07 trs,run,f	Run program
-08 trs,sfl,x	Set Flags
-09 trs,sds,d	Set Display Speed
-10 trs,sws,f	Save Work Space
-11 trs,sof,f,x1,x2	Save Overlay File
-12 trs,xmt,f,x1,x2,x3	Transmit.

[illegible]

Glossary

The SAM76 glossary

Certain abbreviations and conventions are used in this glossary:

Notice

Numbers as 1. xxx; 2. yyy signify alternate meanings of the word or phrase being explained.

Control codes are shown as an up arrow (or circumflex) followed by an upper case letter corresponding to the code meant, thus ^N and ^O mean control "N" and control "O" or "shift out" and "shift in" respectively.

The use of an up arrow (or circumflex) followed by a lower case letter signifies that the characters immediately following are a number in the appropriate radix; thus ^hFF is equivalent to ^o377 or ^d256 - namely hexadecimal, octal and decimal.

activator

character desired. The character.

The character which terminates a string of input characters giving the value of a Input String expression. The Change Activator (CA) primitive will change the activator character to any equal sign is most often used as the activator

active string

The text which at any stage, is yet to be scanned and evaluated.

active text

with certain character pairs, renders it neutral. (See [Protection]).

Text on which the scanning and evaluation process will cause a transformation. It includes all expressions and syntax characters. A technique called protection, which flanks the active text

argument

arguments. (See [Delimiter]).

The strings bounded by delimiters which make up an expression. For example, in %DI,50,,DEFAULT/, "DI" is one argument; 50 is another; and the null string and "DEFAULT" are the two remaining

arithmetic overflow

An occurrence within the computer when the numbers being handled by the multiply function become too large for the system. In the SAM76 language, when this happens the multiply function use its fourth argument as a default string value. (See [Value and Default argument]).

ascii

In geography, are those inhabitants of the globe, which, at certain times of the year, have no shadow. Such are the inhabitants of the torrid zone; by reason the sun is sometimes vertical to them. To find on what days the people of any parallel are Ascii consult an astronomer.

Obs.: American Standard Code for Information Interchange. (See [USASCII]).

bit

A single binary digit which can be only 1 or 0. The number 1011101 is seven bits long.

bulk storage

A storage device, such as disk, drum, or magnetic tape, capable of storing in files a large number of texts that would otherwise overflow the computer's memory. The SAM76 language utilizes these devices with the bulk storage primitives Store File (SF), Bring File (BF), and Erase File (EF).

carriage return

The USASCII character (Octal-15 or Hex 0D) which causes the teleprinter to move to the beginning (lefthand end) of the line it is on. (See [USASCII]).

character

A unit of text that can be produced on a display or other output device by a single keystroke. Every character has a unique USASCII code. (See [USASCII] and [Text]).

computer

An electronic device which works from a series of instructions stored within its memory, for doing jobs, solving problems, etc. With the SAM76 language, the computer carries out the actions specified by scripts in the language. (See [Script]).

concatenate	To take two strings and abut them one against the other in sequence. For example:
-------------	---

```

{}-----
{} %os,ABC/%os,DEF/=ABCDEF
{}
{}-----

```

The execution of the two concatenated Output String expressions back to back in the manner shown causes the two strings ABC and DEF to be printed out together.

control character	Any character produced by typing on a keyboard while the control key (similar to the shift key) is being held down. These characters do not print and are called control characters because they can be used to control devices; this can be with software or hardware as through a teleprinter mechanism called a stunt box. One control code that almost all teleprinters have is control G which rings a bell inside the teleprinter. These characters are very useful in the SAM76 language as secret names.
-------------------	--

decimal number	Any number in the base 10 arithmetic system. A single character of a decimal number is a decimal digit.
----------------	---

default argument	The argument returned in certain SAM76 primitives if the normal function cannot be carried out. For example the Fetch Character expression will result in its default argument if there are no more characters to be read in that particular text. The default argument is always returned actively.
------------------	--

delimiter	Any special character or set of characters, such as syntax or warning characters, which separates and designates the arguments of an expression by causing a break in a string. (See [Warning characters]).
-----------	---

digit	A numeric character, e.g.: 7.
-------	-------------------------------

erase	A general term meaning delete. It is used primarily with the primitives Erase All (EA), Erase Text (ET), and Erase File (EF), since all three of these primitives are used in erasing. The first two primitives erase texts and the third erases files.
-------	---

escape	The character code (Octal-33 Hex 1B) used to signal the computer or user terminal that there will be immediately following one or more characters which in combination are to be interpreted to mean 1. Do a particular thing, or 2. that subsequent code or code characters are to be interpreted differently than heretofore. For instance an Escape code immediately followed by a character which has an octal value of 40 (Hex 20) through 77 (Hex 3F) in combination means that a function identified as having a value between 200 and 237 inclusive is to be performed. (See [USASCII] [Shift codes]).
--------	--

evaluate	To find the value of some expression after it has been scanned. Also, to perform the actions specified by some expression.
----------	--

execute	To cause a typed-in expression to be scanned and evaluated. This is usually done by typing the expression and then striking the "activating" character. Also, to evaluate an expression.
---------	--

file	A named group of texts on a bulk storage device. A file consists of a file name and one or more texts. Many files may be stored. A file is accessed by its name. (See [Text, Text Store] and [Bulk storage]).
------	---

function	An expression which carries out some defined process. All the SAM76 language primitives are functions. Functions may also be created by the user through procedures. These are called user defined functions. (See [User defined functions]).
----------	---

<code>^Ngost^O</code>	Acronym for <code>^Ngosudarstwennyj standart^O</code> the Soviet Union's standard description of the eight bit communications code. <code>^Ngost^O-13052-67</code> provides Russian computer users with assignments similar to that provided by USASCII to American users. (See [USASCII]).
<code>hex</code>	An adjective meaning that a number is in base 16.
<code>hex bit vectors</code>	The hexadecimal representation of a binary number. (see [Nibble]).
<code>implied fetch</code>	<p>A fetch in which the first argument is the name of the fetched text, instead of the two-letter mnemonic for the primitive Fetch Text (FT). After the name, there may be other arguments. An implied fetch expression always works exactly as an active fetch <code>%FT,...../</code>, whether the implied fetch is preceded by a percent "%" or an ampersand "&" warning character.</p>
<code>initialize</code>	<p>To begin an action. When a procedure is started it is initialized. When the SAM76 language processor is initialized, it is started. All pointers are set to beginning conditions, and the working area is cleared and loaded with <code>&OS,%IS//</code>.</p>
<code>input</code>	As a noun: information communicated into the computer. As a verb: the action of putting information into the computer, such as by typing it in.
<code>integer arithmetic</code>	The arithmetic system which deals only with integers (whole numbers). In the SAM76 language Divide (DI) rounds downward to the nearest whole number, e.g.: <code>%DI,3,2/=1</code> .

interaction	The ability of the user to affect running scripts through the use of Input primitives such as "Input String" (IS) and "Input Character" (IC) contained within the scripts. An example of this would be a question-and-answer script in a teaching machine. The user's input would affect the course of the script, changing the actions which followed.
-------------	---

JISCII	Japanese Industrial Standard Code for Information Interchange which provides Japanese computer users with information as to an eight bit code in which the Katakana characters are shown to have octal values between 240 and 337 inclusive.
--------	--

level	The punch hole position in a frame of punched paper tape. Almost all teleprinters manufactured since the Teletype Model 33 use eight level punched tape. A hole represents a binary 1, and the lack of a hole, a binary 0. The eight levels make it possible to have an eight bit number in each row or frame on the tape.
-------	--

line feed	Archaic designation for a code assignment in USASCII (octal-12 or Hex-0A) which caused the printer or other user terminal to advance the position at which the next character would be printed (or displayed) one line - without restoration to the left end (or right-end if arabic, hebrew &c.) of the paper or other display medium. This very useful capability was summarily changed by redesignating this code as New Line, thus messing up all picture drawing character strings, and requiring all future ones to have lots of extra spaces if one wanted to draw vertical lines. (See [USASCII] [New Line] [Return]).
-----------	--

mnemonic	The abbreviation for the names of the primitives. This abbreviation is always found in the first argument of the expression of a primitive to be evaluated. (See [Primitive]).
----------	--

multiexpression	Two or more expressions concatenated rather than nested. For example, %OS,HE/%OS,LLO/ is a multiexpression.
-----------------	---

multi-partition	<p>A feature of a user defined text created through the use of the "mt" multi-partition text function. The feature is also known as a "multi-partition character" and is in effect a numerically identified pointer which may be replaced by an arbitrary number of argument strings. In this context an argument string is a string of characters preceded by an argument separator (usually a comma). Replacement is accomplished during the execution of the fetching of the text; replacement of the multi-partition is with all of the argument string to the right of the argument whose position counting from the left is equal to the value of the multi-partition.</p>
nesting	<p>The process of placing one expression within another so that when the inner expression is evaluated its final value becomes part of the argument for an outer expression. For example, %AD,3,%SU,4,3/=4 is a nested expression.</p>
neutral	<p>Any text unaffected by the scanning and evaluation process. Also, text strings resulting from the scanning and evaluation process.</p>
neutral string	<p>The set of text strings resulting from any stage of the scanning and evaluation process. For a primitive to be evaluated, all its arguments must be in the neutral string. (See [Active string]).</p>
new line code	<p>An USASCII character (octal-12 or hex-0A) which has the same effect as the combination of carriage return line feed codes. In other words, the printer is moved to the beginning (lefthand end) of the next line.</p>
nibble	<p>Unit of information 4 bits long. Computer word size may be expressed in nibbles, thus an 8 bit word is 2 nibbles &c.</p>

null value	The value of any primitive which is replaced by the null string following execution. For example, the final value of the Output String (OS) primitive is the null value. Null value also is used to describe a primitive, as in the statement, "The OS primitive is a null valued primitive."
------------	---

null string	A string containing no characters. A null string can be used as a name for a text. It has the numeric value of zero in the arithmetic primitives. (See [Argument]).
-------------	---

numeric character	The character - (minus), and the decimal digits, as well as any character that represents a number in number bases greater than 10. Notice that the characters which are numbers change when the number base changes. (For example, 9 is a non-numeric character in base 8).
-------------------	--

octal	An adjective meaning that a number is in base 8.
-------	--

octal bit vectors	The octal representation of a binary number. (see [Type]).
-------------------	--

output	As a noun: information sent from the computer. As a verb: the act of the processor in sending information from the computer, as on the output printer.
--------	--

partition	A feature of a user defined text created through the use of the "pt" partition text function. The feature is also known as a "partition character" and serves two basic purposes; 1 - as a separator of text elements, 2 - as a numerically identified pointer which may be replaced with character strings or other partitions during the execution of a fetch.
-----------	--

primitive	The simplest executable SAM76 language expression.
procedure	An executable expression in a text. When it is actively fetched the expression is executed, but when it is neutrally fetched it is not executed. With a procedure, the user can define functions of his own.
processor	A computer loaded with the special program which enables it to carry out the actions specified in a SAM76 language script. (See [Computer]).
program	A series of instructions which, executed in a specific order, cause a computer to take some desired action. In the SAM76 language there is no such thing as a program. The computer is controlled by a script. The SAM76 language scripts are very different from standard computer programs. (See [Script]).
protection	A technique of rendering active text neutral by enclosing the text in between pairs of certain characters such as parentheses or angle brackets (<>).
recursive	A procedure which recalls itself, and so loops back and uses itself over and over again.
replacement	The process by which an expression, after being evaluated, is replaced by its value.

restart expression

The expression &OS,%IS// which is loaded into the working area before anything else happens. The Input String expression is evaluated first and this is what makes it possible to input strings of indefinite length. When the "activator" character is struck the active expressions typed in are evaluated, and the final value of the typed-in string is typed out by the Output String primitive. Then a new &OS,%IS// is loaded into the working area.

scan

To analyze the active string character by character. The warning characters are marked, while some other characters are deleted.

scanner

A mechanism in the SAM76 language which scans the contents of the active string, marks it for evaluation, and controls the evaluation.

script

A sequence of statements for computer control written in the SAM76 language. A script can be made up of nested expressions, multiexpressions, and fetches to one or more procedures and texts. A script in the SAM76 language corresponds to a program in other languages.

shift codes

Character codes which serve the function of telling the computer and user terminals that a change to some other code set is desired, that is to say that the meaning of each code sent or received subsequently is to be different from what it was prior to the occurrence of the shift code. In USASCII there are three such codes. Escape (Octal-33) which is used with one or more additional characters to signal the desire to change code sets to some particular other set. Shift-Out (Octal-16) changes the meaning of a six bit portion (octal-40 through octal-177) of the code set to the alternate one of the two sets of 96 codes available. Shift-In (Octal-17) restores the meaning of the code set to the basic one of the two available. Interestingly these last two codes have been named by the Russians respectively ^Nrus^O for Shift-Out, and ^Nlat^O for Shift-In thus more precisely describing their function, namely to make available to the Russian user the Cyrillic, or the Latin alphabets. (See [USASCII] [Escape] [^Ngost^O]).

string	A sequence of characters. Its length is measured by the number of characters it contains. A string can be any length. A sequence zero characters in length is a null string.
syntax	The syntax of a language is the way words are put together to form phrases, clauses, and sentences. The syntax of the SAM76 language is the way text strings are put together to form executable expressions.
system	A computer program, or a group of related computer programs that work together as a unit. The program which allows the SAM76 language to function is a system. (See [Processor]).
teledisplay	A communications device like a typewriter in that it has a keyboard, but instead of printing its messages on papers shows them without any permanent record being made. Typically the display medium is like a television set and in fact may use a television set. Other media are plasma panels, or surfaces which contain thousands of light bulbs.
teleprinter	A device like a typewriter, with a keyboard and a printing element, through which a user communicates with a computer.
text	<ol style="list-style-type: none">1. Anything that can be typed on the teleprinter keyboard. A sequence of characters. Text includes letters of the alphabet, decimal integers, punctuation, and other keystroke characters; also2. A string of characters stored in memory under a referencing name. Texts are defined through the use of the Define Text (DT) primitive.
text area	The place in memory in which the texts defined with the Define Text (DT) primitive are stored.

text divider	A pointer to a position in a named text. When a text is initially defined the pointer is implicitly placed to the left of the first character of the defined character string. The pointer is moved as the result of the execution of a variety of text fetching functions, or through the use of the "md" move divider function. The position of the divider in a given text may be ascertained visually through the use of the "vt" function, or numerically through the execution of the "crd", "cld", "qrd" and "qld" functions.
--------------	--

text name	The string by which a text can be referenced, i.e.: the name of the text. It is theoretically of unlimited length and can be made up of any characters, including the null string or no characters.
-----------	---

trace mode	In the SAM76 language, a manner of operation that causes the evaluation process to occur one step at a time, with each step being made after the user hits a specified key. This mode makes it very easy to analyze even the most complex scripts and procedures.
------------	---

tyte	Unit of information three bits long. Computer word size may be expressed in tytes, thus a 12 bit word is 4 tytes, a 21 bit word is seven tytes, and will accomodate three seven bit USASCII characters, or
------	--

a 24 bit word being 8 tytes will accomodate three eight bit USASCII characters; for convenient handling of segment gaps and characters word sizes which are multiples of three tytes are very suitable, but words which are multiples of bytes (eight bit long information unit) make the programming of SAM76 processors very inconvenient if the full eight bit USASCII character set is to be handled. (See [USASCII]).

USASCII	United States of America Standard Code for Information Interchange. It is a standard code which provides the binary representation and the meaning of each combination of bits possible for an eight bit communication code system. This allows codes for 256 different characters: upper case letters, numbers, symbols, control characters, and lower case letters. These codes may be used to represent characters in computers, on communications circuits, and to control the operation of user terminals such as teleprinters or teledisplays. For your convenience we have put copies of charts which show various assignments used elsewhere in this book.
---------	--

user	The human element in the SAM76 language. Hrm* is the person using the language. (YOU!!!).
------	---

The author prefers to use the syllable "pip" as in: "your'e a pip". This syllable may be attached to root words both as a suffix and/or as a prefix; for example: "piphole" or "chairpip" (meaning "manhole" and "chairman" in case you didn't figure it out yet).

*According to the United States Congress Hrm is the ne sexist way of implying nobody in particular.

user defined function	A procedure which may be activated with an implied fetch in such a way that it is sometimes difficult to tell whether it is a SAM76 script or a real primitive. (See [Implied fetch]).
--------------------------	--

value	The text which replaces a primitive when the primitive is evaluated. Also, the text (such as that of an argument) in which all active expressions have been evaluated.
-------	--

vector	A binary number expressed in octal or hexadecimal notation in the SAM76 language. (See [Octal/Hex bit vector]).
--------	---

warning characters	Under initialized conditions the warning characters are as follows: to start the expression a % for an active string or an & for a neutral string, followed by arguments separated by commas and ending with a /. For protection use !.... / or (....) or <.....>. To be tricky you can use a # to start and a : or a ; to end an expression if you want the scanner to search through user defined functions first for the primitive used in that string. If one wants to protect a single character precede it with a @.
--------------------	--

working area	A section of memory where the restart expression is loaded and the scanning and evaluation processes take place. It is called the working area because this is where all the work is done. The active and neutral strings are held in this area during processing.
--------------	--

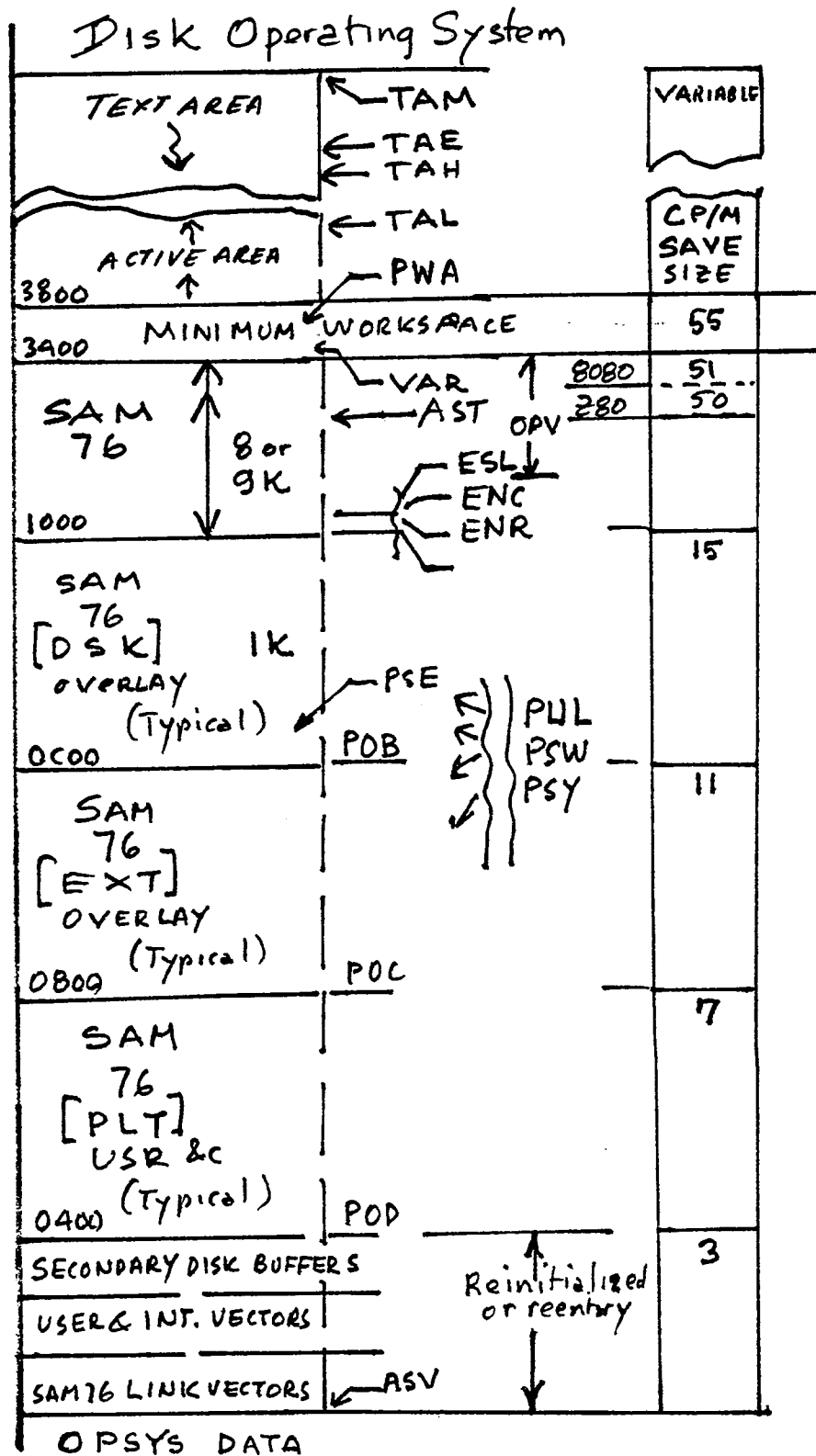
work space

That portion of memory which includes both the "work area" and the table of pointers, switches and other user system management options.



[illegible]

Typical System Layout



eXperimental Functions

This section will discuss a number of the functions that may be found in SAM76 language processors released after July 1978. These functions are located in "overlays" and are linked through the POA, POB, POC and POD locations as described elsewhere.

the XU family

A family of experimental functions exist as subfunctions of the "xu" eXperimental User function; a list of these functions may be obtained by executing &xu,lf, /= which is defined as follows:

|xu,lf,s0| eXperimental User List Functions

The value of this expression is a list of the subfunctions of the "xu" primitive. Each name in the list is preceded by the string symbolized by "s0".

|xu,dm,x1,x2,x| Duplicate Memory

This null valued function fills memory from the location symbolized in the current X base by "x1" to the location symbolized by "x2" inclusive with the "byte" symbolized by "x". {to be used with care as for instance when one wishes to scuttle the system which is about to be intruded by an unauthorized peeker".

|xu,fm,x1,x2,s0| eXperimental User Fetch Memory

The value of this function is the contents of the system memory between locations symbolized in the current X base by "x1" and "x2" inclusive. Each word content is preceded by the string symbolized by "s0".

|xu,mm,x1,x2,x3| eXperimental User Move Memory

This null valued function serves to move the contents of the system memory between "x1" and "x2" inclusive to a location whose starting point is symbolized by "x"; all of these are in the current X base.

|xu,rl,sym| eXperimental User Read Label

This function returns as a value the contents of the memory location identified by the valid label symbolized by "sym". A list of the valid labels is obtained through execution of the %xll, / Xamine Label List function. {Note this differs from the "xal,sym" function which only gives the actual machine adress of the symbol label}.

|xu,sum,x1,x2| eXperimental User SUM

The value of this function is the low order "byte" of the arithmetic sum of all memory cells between the locations symbolized in the current X base by "x1" and "x2" inclusive. {This is useful in verifying portions of the system; overlays are usually checksummed to zero by storing the appropriate value in the last location of the overlay}.

|xu,tm,x1,x2| eXperimental User Type Memory

This valued function contains in "printable" form the contents of the memory cells located between "x1" and "x2" inclusive. Non printing characters are displayed as "dots".

|xu,vm,x1,x2,x3,s0| eXperimental User Verify Memory

This valued function compares the contents of the memory between the locations symbolized in the current X base by "x1" and "x2" inclusive with memory locations which start at the location symbolized by "x3". The value returned provides a list of adresses of the locations which are different. Each adress is preceded with the string symbolized by "s0".

|xu,wl,sym,x| eXperimental User Write Label

This null valued function is the reverse of the "xu,rl,sym" function, in that its execution will cause the storing of the value symbolized by "x" at the actual location identified by the valid label symbolized by "sym".

|xu,xmt,m,0,x1,x2| eXperimental User X{trans}mit

This null valued function is used to generate for transmission purposes a blocked checksummed group of records of the contents of memory locations between those symbolized in the current X base by "x1" and "x2" inclusive. The symbol "m" denotes one of several modes of transmission as defined below:

mode 0	No local display, no error feedback
mode 1	No local display, automatic error control
mode 4	Local progress display, no error control
mode 5	Local progress display, with error control

All of the above transmissions are in the blocked checksummed format to be described later and are through the "group 2" communications channel of the SAM76 language processor.

The argument symbolized by "0" in the above expression is reserved for future control and selection purposes and is undefined at this time.

Note:The format is that which is currently used in the "loadit" software distributed with the paper tape and cassette media.

|xu,rcv,m,0,x1,x2| eXperimental User Receive

This valued function accepts data in the checksummed block format generated by the "xu,xmt" function. The modes are the same as described for the transmit function. The arguments symbolized by "x1" and "x2" are to force loading elsewhere than specified within the received block of data; "x1" is an offsetting value.

This function may be used to load SAM76 language object code distributed in the "loadit" format.

|xu,dt,t,x1,x2,...,xn| eXperimental User Define Text

Execution of this function creates in the text area a text with a protection class of HEX 80. This text contains in successive locations following its name the binary equivalent of the numbers symbolized in the current "X" base by x1,x2,...,xn. This text is intended to contain executable machine code, and execution of %ft,t, ... / or %n, ... / &c. will casue a "CALL" to be made to the location containing "x1". [see "xu,ft"].

|xu,ft,t,s0| eXperimental User Fetch Text

The value of this function is the binary contents of the text whose name is symbolized by "t", each binary value is preceded by the delimiter symbolized by "s0" and is represented in the current "X" base.

		It is evident that use of some of the foregoing value
	use of the	returning functions would fail due to lack of
	"nud"	sufficient user work space for storage prior to
	function	evaluation. It is appropriate to use the "nud" null
		display function to obviate this problem; typical
		expressions making use of "nud" and some of the above

functions are shown below:

%nud,xu,fm,100,200, /= Displays from 100 to 200

%nud,xu,rcv,5,9000/= Accepts the blocked checksummed transmission o
f
object code offsetting the load point by 9000 Hex. Assuming that an 8K object code of the SAM76 language interpreter intended for loading at 8000 is received, the loading will take place at 1000 instead. While loading takes place progress of load is displayed on the user console, and each block correctly received will be acknowledged to the sender; if incorrectly received then a retransmission will be requested.

User may make use of the "sw" and "sy" functions by storing at locations determined by %xal,PSW,offset/ and %xal,PSY,offset/, respectively, the memory addresses which execution of these functions should reach. If the offset is 0, then the temporary table address in the "work space" is given, if the offset is that value returned by executing %xal,OPV/, then the permanent table address is given. The "xrp" and "xwp" (or "xu,rl" and "xu,wl") may be used to read and write in these locations.

"sw" and "sy" linking
