SIMULATION OF NERVE CELL KINETICS USING INTERACTIVE SIMULATION LANGUAGE

R. D. Benham
Interactive Mini Systems, Inc.
5312 W Tucannon
Kennewick, Washington 99336

Daniel K. Hartline
Department of Biology
University of California, San Diego
La Jolla, California 92037

INTRODUCTION

Interactive Simulation Language (ISL) can be used by scientists and engineers without prior programming experience for the solution of nonlinear algebraic and differential equations. ISL permits hands-on interactive operation in which the user can monitor the course of a computation (via scope, printers, etc) and change parameter values during execution (via knobs, switches, keyboard). ISL functions well in either an all digital system or in a system utilizing an analog interface. Since ISL is modular computing functions can be added or deleted for tailoring the language to many applications in data collection, data analysis, curve fitting, hybrid computer simulation, etc.

ISL processes simulations quite rapidly by using a single word mantissa and a single word exponent (minipoint). In addition extremely large problems can be solved on a small computer. ISL is currently operational on PDP 8/12, PDP 7/9/15, PDP 11, NOVA, EAI 640/PACER 100, DDP/PRIME, and many hybrid computers. This paper shows application of ISL to neurobiology by:

. presenting an overview of ISL programming
. illustrating the techniques for the simulation of a repetitively firing nerve cell
. presenting comparative results for ISL and SNAX

ISL PROGRAMMING OVERVIEW

It is easiest to consider the ISL system as providing a large number of operational elements such as found on an analog computer. These elements include integrator, add, multiply, and many others (see Table 1). An ISL program is formed by "interconnecting" these elements to satisfy the equation being solved.

Drawing the Block Diagram

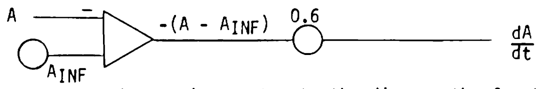
To illustrate the programming technique consider the equation for adaptation (A) of a nerve cell:

$$\frac{dA}{dt} + 0.6\ (A - A_{INF}) = 0 \qquad A_{(0)} = 1.0$$

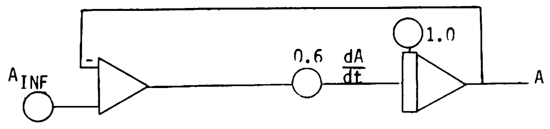
The first step in programming is to equate the highest derivative to the lower order terms. For example:

$$\frac{dA}{dt} = -0.6\ (A - A_{INF})$$

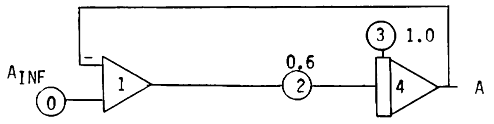
The simulation diagram (using symbols from Table 1) is:



Next by adding an integrator to the diagram the function (A) is available:



Notice the initial condition is shown as an auxiliary input to the integrator. To complete the simulation diagram it is necessary to assign numbers to each operational element. The rule for assigning numbers is analogous to building the simulation diagram. That is, to form the derivatives and then integrate the highest derivative first. Thus:



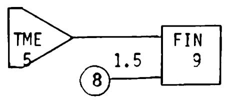
The final step in preparing the simulation diagram is to form operational blocks that:

. label headings (HDE statement)
. identify printing or plotting variables (PRI)
. allow the program to iterate and determine when computation should stop (FIN statement)

To aid in the interpretation of tabulated output data, provision exists for specifying headers above each column. These headings will be the names given to the variables identified by the operational equivalent in the print (PRI) statement. Up to five names (having 6 characters can be included in one heading statement (HDR). The HDR automatically labels the first value as TIME before using the defined labels. Accordingly the print statement gives the value of time. A header and print statement for the example could be:

```
6  HDR  A
7  PRI  4
```

A finish (FIN) statement, as shown below, would be incorporated into each ISL program. This statement allows iteration and also provides the user with a means of manually terminating a run with a keyboard interrupt. Its diagram would be:



When the value of the first input (time) is less than the second input, then for a new iteration, the FIN statement transfers control to statement 0. When the first input is greater than or equal to the second input the next numerical statement is executed. In this example we will use an END statement to terminate the run.

Preparing the Listing

The program is entered into the computer by using the instructions shown in Table 2. The user area of memory is first erased with the "K" command. Then by typing "A" (for APPEND) the teletype responds by typing (0) for the first line number. The user may then type in the program:

```
0  CON = 1.0E-1
1  ADD  -4, 0
2  POT  1, 6.0E-1
3  CON = 1.0E0
4  INT  3, 2
```

```
S= 1.ØØØE-2     T=1Ø

 TIME          A
 Ø.ØØØE Ø      1.ØØØE Ø
 9.999E-2      9.474E-1
 2.ØØØE-1      8.979E-1
 3.ØØØE-1      8.513E-1
 4.ØØØE-1      8.Ø74E-1
 5.ØØØE-1      7.661E-1
 6.ØØØE-1      7.272E-1
 7.ØØØE-1      6.9Ø6E-1
 8.ØØØE-1      6.561E-1
 9.ØØØE-1      6.236E-1
 1.ØØØE Ø      5.931E-1
 1.1ØØE Ø      5.642E-1
 1.2ØØE Ø      5.371E-1
 1.3ØØE Ø      5.116E-1
 1.4ØØE Ø      4.876E-1
 1.5ØØE Ø      4.649E-1
```

a. Tabular output

```
RANGE =  1.ØØØE Ø
A=A
           Ø....1....2....3....4....5....6....7....8....9....Ø....1....2
 Ø.ØØØE Ø                                                     A
 9.999E-2                                                  A
 2.ØØØE-1                                                A
 3.ØØØE-1                                              A
 4.ØØØE-1                                           A
 5.ØØØE-1                                         A
 6.ØØØE-1                                       A
 7.ØØØE-1                                      A
 8.ØØØE-1                                    A
 9.ØØØE-1                                  A
 1.ØØØE Ø                                 A
 1.1ØØE Ø                               A
 1.2ØØE Ø                              A
 1.3ØØE Ø                             A
 1.4ØØE Ø                           A
 1.5ØØE Ø                          A
```

b. Teletype Plot

Figure 1. ISL Output for Adaptation Example

```
 5  TME  5
 6  HDR  A
 7  PRI  4
 8  CON = 1.5E 0
 9  FIN  5, 8
10  END  10
```

For each line of code the teletype prints a line number, following which the user types a three-character function code, followed by its arguments (separated by commas). Arguments are either "inputs" to the elements (see integrators) or numerical values assigned to them (floating point notition for POTs and CONs).

If a spelling error is detected while entering the program the printer will respond by typing a question mark (?) and retype the statement number. It then waits for the correct information. The computer will not catch "patching errors". For example if a 7 were typed for a 4 in statement 1, the computer would not recognize the error. However the user can correct any numerical error at any time prior to terminating the entry with the "slash" (/). This signals the computer to cancel the entry and accept a new one. Any statement can be modified or re-entered with the MODIFY (M) command shown in Table 2.

Running the Program

The program is run by selecting the integration step size and print interval with the "I" command. For the current example a step size of S = 1.0E-2 and a print interval of T = 10 will allow tabular output as shown in Figure 1: By using the X command of Table 2 the information is plotted. Notice the range for 50 spaces is set to 1.0 as defined by the RANGE = 1.000E 0 comment.

Of much more interest is to output the results to a scope and repetitively run the simulation while varying A_{INF} with an analog pot. This may be done by modifying the following lines with the "M" command:

```
 0  ADC  0            (use pot 0 to change A_INF)

 6  DIS  5, 8, 4, 7   (plot A versus Time on scope)
 7  CON = 2.0E 0      (normalized value of A)

10  CHM  10           (allows repetitive operation)
11  END  11
```

With scope output and parameter variation through an analog pot (through an ADC channel) better interactive communication is established than with just a TTY printer or a line printer. With this form of interaction one can begin to appreciate the speed and power of a simulation using ISL. Solutions will appear about 250 times faster than possible with interactive languages like BASIC. In addition an 8K computer equipped with ISL can solve up to 100 non-linear differential equations. Over 1000 equations can be solved with 32K of memory.

SIMULATING A REPETITIVELY FIRING NERVE CELL

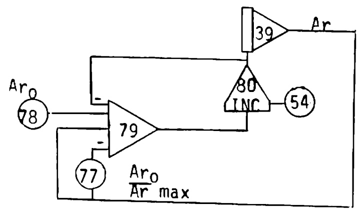
This section illustrates the use of ISL on the more complex problem of a repetitively firing nerve cell. The model is summarized with the simulation diagram of Figure 2 and the listing of Figure 3. Notice that the ISL simulation diagram combines both the mathematical and logical flow diagrams for the nerve cell. As such it is an aid (ie. mathematical analog) in visualizing the operation of the cell. The model is composed of 8 non-linear differential equations that account for:

. Conductance associated with each spike	(gs)
. Current injected with a microelectrode	(I)
. Resting potential adaptation	(Ar)
. Short and long term active response adaptation	(A1,A2)
. Active pacemaker response	(P)
. Threshold	(θ)
. Perturbation variable	(W)

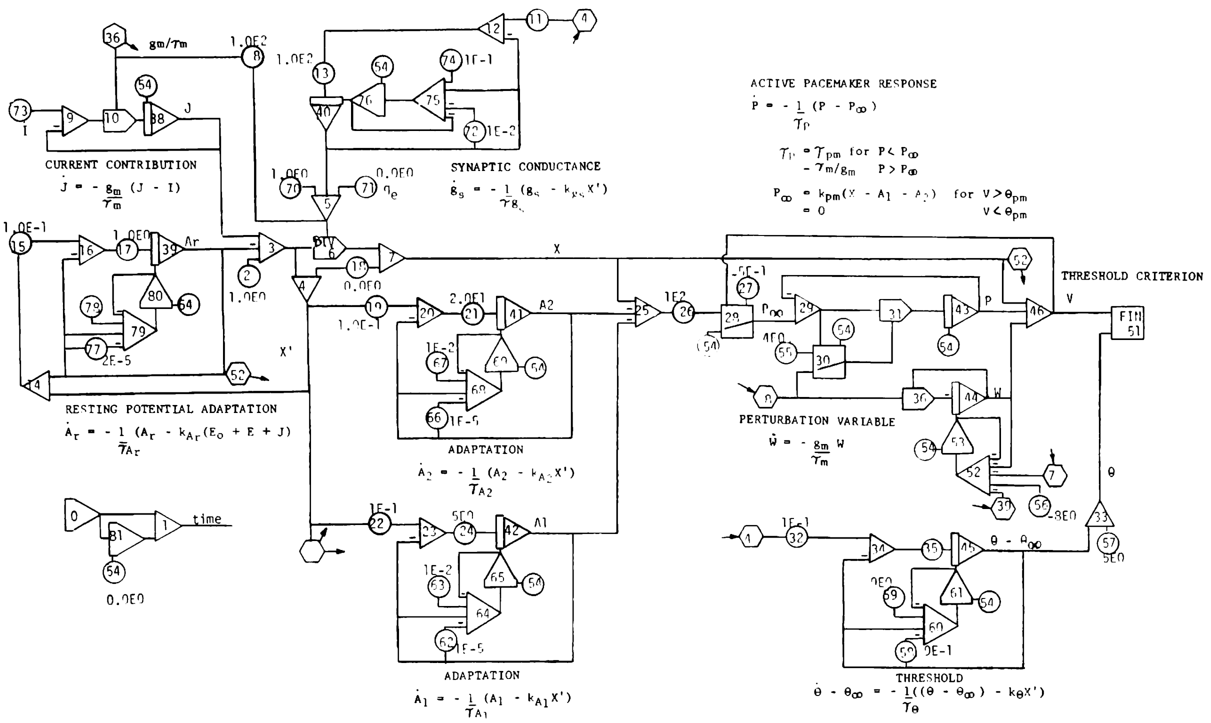
The basic equation for each term is similar to the equation for adaptation used for describing ISL programming in the last section. However there are additional complexities as can be seen by close inspection of Figure 2. A Cell will fire (spike) when the membrane potential (V) becomes equal to or greater than the threshold (θ). When this happens new initial conditions are automatically calculated and imposed for the terms gs, Ar, A1, A2, W, and θ. The form of the IC up-date equation is given for the adaptations:

$$A_{new} = A_{old} + \frac{A_o(A_{max} - A_{old})}{A_{max}} = A_{old}\left(1 - \frac{A_o}{A_{max}}\right) + A_o$$

The new value is a fixed fraction of the distance from the old value to a maximum value. The ISL implementation for this equation is:



The initial condition calculations appear after the FIN statement (no. 51) so that they all are calculated when V = θ and t = time of the spike. Since ISL includes a



CURRENT CONTRIBUTION
$J = -\frac{g_m}{\tau_m}(J - I)$
SYNAPTIC CONDUCTANCE
$\dot{g}_s = -\frac{1}{\tau_{g_s}}(g_s - k_{gs}X')$
ACTIVE PACEMAKER RESPONSE
$\dot{P} = -\frac{1}{\tau_P}(P - P_\infty)$
$\tau_P = \tau_{pm}$ for $P < P_\infty$
τ_m/g_m $P > P_\infty$
$P_\infty = k_{pm}(X - A_1 - A_2)$ for $V > \theta_{pm}$
$= 0$ $V < \theta_{pm}$
THRESHOLD CRITERION
RESTING POTENTIAL ADAPTATION
$\dot{A}_r = -\frac{1}{\tau_{A_r}}(A_r - k_{A_r}(E_o + E + J))$
ADAPTATION
$\dot{A}_2 = -\frac{1}{\tau_{A2}}(A_2 - k_{A_2}X')$
PERTURBATION VARIABLE
$\dot{W} = -\frac{g_m}{\tau_m}W$
ADAPTATION
$\dot{A}_1 = -\frac{1}{\tau_{A_1}}(A_1 - k_{A_1}X')$
THRESHOLD
$\dot{\theta} - \theta_\infty = -\frac{1}{\tau_\theta}((\theta - \theta_\infty) - k_\theta X')$
time
FIN

Figure 2. ISL Simulation Diagram for Repetitively Firing Nerve Cell

```
Ø    TIME   Ø                   = Ø.ØØØE Ø    Ø
1    ADD    Ø, 81               = Ø.ØØØE Ø    1
2    CON    2                   = 1.ØØØE Ø    2
3    ADD    38, -39, 2          = Ø.ØØØE Ø    3
4    ADD    3, 18               = Ø.ØØØE Ø    4
5    ADD    7Ø, 4Ø, 71          = Ø.ØØØE Ø    5
6    DIV    3, 5                = Ø.ØØØE Ø    6
7    ADD    6, 18               = Ø.ØØØE Ø    7
8    POT    5,  1.ØØØE 2        = Ø.ØØØE Ø    8
9    ADD    73, -38             = Ø.ØØØE Ø    9
1Ø   MUL    9, 8                = Ø.ØØØE Ø    1Ø
11   POT    4,  1.ØØØE-1        = Ø.ØØØE Ø    11
12   ADD    11, -4Ø             = Ø.ØØØE Ø    12
13   POT    12,  1.ØØØE 2       = Ø.ØØØE Ø    13
14   ADD    4, 39               = Ø.ØØØE Ø    14
15   POT    14,  1.ØØØE-1       = Ø.ØØØE Ø    15
16   ADD    -39, 15             = Ø.ØØØE Ø    16
17   POT    16,  1.ØØØE Ø       = Ø.ØØØE Ø    17
18   CON    18                  = Ø.ØØØE Ø    18
19   POT    4,  1.ØØØE-1        = Ø.ØØØE Ø    19
2Ø   ADD    19, -41             = Ø.ØØØE Ø    2Ø
21   POT    2Ø,  2.ØØØE 1       = Ø.ØØØE Ø    21
22   POT    4,  1.ØØØE-1        = Ø.ØØØE Ø    22
23   ADD    22, -42             = Ø.ØØØE Ø    23
24   POT    23,  5.ØØØE Ø       = Ø.ØØØE Ø    24
25   ADD    7, -41, -42         = Ø.ØØØE Ø    25
26   POT    25,  1.ØØØE 2       = Ø.ØØØE Ø    26
27   CON    27                  =-5.ØØØE-1    27
28   IPL    46, 27, 26, 54      = Ø.ØØØE Ø    28
29   ADD    28, -43             = Ø.ØØØE Ø    29
3Ø   IPL    54, 29, 8, 55       = Ø.ØØØE Ø    3Ø
31   MUL    29, 3Ø              = Ø.ØØØE Ø    31
32   POT    4,  1.ØØØE-1        = Ø.ØØØE Ø    32
33   ADD    45, 57              = Ø.ØØØE Ø    33
34   ADD    32, -45             = Ø.ØØØE Ø    34
35   POT    34,  2.5ØØE 2       = Ø.ØØØE Ø    35
36   MUL    44, 8               = Ø.ØØØE Ø    36
37   CON    37                  = 1.ØØ2E Ø    37
38   INT    54, 1Ø              = Ø.ØØØE Ø    38
39   INT    8Ø, 17              = Ø.ØØØE Ø    39
4Ø   INT    76, 13              = Ø.ØØØE Ø    4Ø
41   INT    69, 21              = Ø.ØØØE Ø    41
42   INT    65, 24              = Ø.ØØØE Ø    42
```

```
43   INT    54, 31                  = Ø.ØØØE Ø    43
44   INT    53, -36                 = Ø.ØØØE Ø    44
45   INT    61, 35                  = Ø.ØØØE Ø    45
46   ADD    7, 43, 44               = Ø.ØØØE Ø    46
47   DAC    1, 37, 1                = Ø.ØØØE Ø    47
48   DAC    33, 82, 2               = Ø.ØØØE Ø    48
49   DAC    46, 82, 2               = Ø.ØØØE Ø    49
5Ø   DAC    56, 84, 1               = Ø.ØØØE Ø    5Ø
51   FIN    46, 33                  = Ø.ØØØE Ø    51
52   ADD    -44, -53, -7, 56, -39     = Ø.ØØØE Ø
53   INC    54, 52                  = Ø.ØØØE Ø    53
54   CON    54                      = Ø.ØØØE Ø    54
55   CON    55                      = 4.ØØØE Ø    55
56   CON    56                      =-8.ØØØE Ø    56
57   CON    57                      = 5.ØØØE Ø    57
58   POT    45,  9.ØØØE-1           = Ø.ØØØE Ø    58
59   CON    59                      = 9.ØØØE Ø    59
6Ø   ADD    -61, 59, 45, -58          = Ø.ØØØE Ø     6Ø
61   INC    54, 6Ø                  = Ø.ØØØE Ø    61
62   POT    42,  1.ØØØE-5           = Ø.ØØØE Ø    62
63   CON    63                      = 1.ØØØE-2    63
64   ADD    -65, 63, 42, -62          = Ø.ØØØE Ø     64
65   INC    54, 64                  = Ø.ØØØE Ø    65
66   POT    41,  1.ØØØE-5           = Ø.ØØØE Ø    66
67   CON    67                      = 1.ØØØE-2    67
68   ADD    -69, 67, 41, -66          = Ø.ØØØE Ø     68
69   INC    54, 68                  = Ø.ØØØE Ø    69
7Ø   CON    7Ø                      = 1.ØØØE Ø    7Ø
71   CON    71                      = Ø.ØØØE Ø    71
72   POT    4Ø,  1.ØØØE-2           = Ø.ØØØE Ø    72
73   CON    73                      = Ø.ØØØE Ø    73
74   CON    74                      = 1.ØØØE-1    74
75   ADD    -76, -72, 4Ø, 74          = Ø.ØØØE Ø     75
76   INC    54, 75                  = Ø.ØØØE Ø    76
77   POT    39,  2.ØØØE-5           = Ø.ØØØE Ø    77
78   CON    78                      = 1.ØØØE-1    78
79   ADD    -8Ø, 78, 39, -77          = Ø.ØØØE Ø     79
8Ø   INC    54, 79                  = Ø.ØØØE Ø    8Ø
81   INC    54, Ø                   = Ø.ØØØE Ø    81
82   CON    82                      = 5.ØØØE 1    82
83   CHM    1                       = Ø.ØØØE Ø    83
84   CON    84                      =-1.ØØØE Ø    84
85   END    Ø                       = Ø.ØØØE Ø    85
```

Figure 3. ISL Listing for a Repetitively Firing Nerve Cell

second IC iteration the calculation must not be up-dated during the second pass. The solution to this is the INC block which allows correction during the compute pass but not the second IC pass through the program. Since the INC block adds the new input to the old output, the INC output must be subtracted from the input to satisfy the above equation..

The Change Mode (CHM) instruction (block 82) allows simulation control for repetitive operation. ISL has two modes: "compute" in which normal program execution occurs and "initial conditions" (IC), in which initial conditions are set as shown in the proceding example. When the CHM is executed and ISL is in IC, the mode is changed to compute. Control is then transfered to block 0 (zero). Conversely when CHM is executed and ISL is in compute mode the mode is changed to IC for an initial condition pass. The FIN statement then allows for the next instruction to be executed when in IC mode. This allows the CHM block to again change the mode back to compute from IC mode.

Notice that blocks 0. 81, and 1 are used for generating time. Each time there is a spike ($V = \theta$), time as generated by the TME block (0) is automatically reset to 0. Therefore by using the INC (block 81) the value of time (when $V = \theta$) is added to the last INC value and remembered. Thus by adding the output of TME and INC the true accumulative time is made available.

Notice also that the input relays (IRL) as shown in blocks 28 and 30 are used to select values for Υ_p and P_{∞} for the active response equation (P) which are dependent on condition requirements on certain variables.

Statements 45, 46, and 47 show the ISL commands for outputting results to digital to analog converters which are in turn used to drive a scope (see Figure 4).

COMPARISON OF ISL AND SNAX

The SNAX language was developed for the PDP 11/45 computer and is currently used for math model development and testing of the nerve cell simulation. The purpose of the ISL implementation is to provide comparative information for the two languages.

Approximately 4 hours were required to study the math model and draw the preliminary ISL flow diagram. This experience generated several questions that required consultation for correction. An additional seven hours were needed to enter, debug, test, and compare the model with the SNAX version. The same PDP 11/45 computer was used. ISL required 15 seconds of computer time to simulate the 14 firings shown in Figure 4. The ISL program required about 520 memory locations to hold the 85 lines of code. Although provision exists for using hardware multiply/divide within ISL, this feature was not used for the comparison run. ISL could have processed the information substantialy faster if this additional hardware would have been used.

The SNAX simulation required 230 lines of code. SNAX processed the simulation in 14 seconds by using the floating point processor available on the PDP 11/45. The results for ISL and SNAX were found to be identical as can be seen by comparing Figure 4. To verify the comparison of ISL and SNAX a run was made where SNAX results were written over the ISL results (saved on a memory scope) and no appreciable difference could be noticed.

Figure 3A and 3B show two different time scales for a simulation of an abrupt depolarization to the cell which is responded to by an initially high but exponentially declining firing rate. For each simulation the upper line represents threshold (θ) and the lower line membrane potential (V). Impulse time-course is not simulated, but impulses occur at the discontinuities of θ and V, representing resets of these quantities by the spike.

CONCLUSIONS

ISL is an efficient general purpose language suitable for simulation of analog computer functions on minicomputers. Its block diagram representation scheme aids in visualizing the processes being simulated. For an on-line interactive language it is rapid and conservative of memory. Its availability on a number of different minicomputers make ISL programs "exportable" for many of the computer systems currently employed in neurobiology research.

ACKNOWLEDGEMENTS

Support by Interactive Mini Systems, Inc. and grants from NSF (#GJ-43177), the Alfred P. Sloan Foundation, and the University of California Academic Senate to DKH.

REFERENCES

1. ISL Interactive Simulation Language Programing Manual, Interactive Mini Systems, Inc., 5312 W Tucannon, Kennewick, Washington 99336 (1974)

2. Hartline, D. K., "SNAX: A language for interactive neuronal modeling and data processing", Department of Biology, University of California, San Diego, La Jolla, California 92037

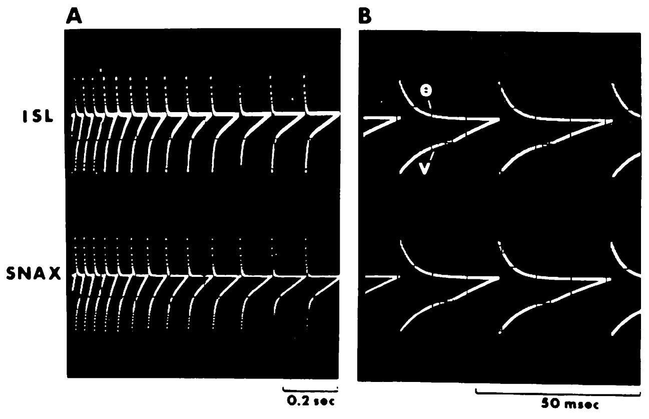


Figure 4. Comparison of ISL and SNAX for a Repetitively Firing Nerve Cell

TABLE 1

MATHEMATICAL INSTRUCTIONS

ARITHMETIC

function	instruction		remarks
absolute value	ABS	N1	R = |N1|
add subtract	ADD	±N1, ±N2,... ±N9	R = ±N1 ± N2 ± ... N9
constant	CON	Nk	R = Nk
divide	DIV	N1, N2	R = N1/N2
equate	EQT	N1, N2	R = O : N2 = N1
multiply	MUL	N1, N2	R = N1 x N2
pot	POT	N1, y.yyyE y	R = N1 x y.yyyE y

TRANSCENDENTALS

function	instruction		remarks

sine	SIN
	SND
		N1
		N1
			R = sin N1 radians
			R = sin 2π N1 degrees

cosine	COS
	CSD
		N1
		N1
			R = cos N1 radians
			R = cos 2π N1 degrees

logarithm	LOG
	LNX
		N1
		N1
			R = log(10) N1
			R = log(e) N1

exponential	TXP
	EXP
		N1
		N1
			R = (10) exp N1
			R = (e) exp N1

square root	SQT	N1	R = square root N1

SUBROUTINE

function	instruction		remarks
subroutine	SBR	N1, N2 . . . N8	N1 = block transfer no. N2 . . . N8 are inputs SBR can reference all blocks except 0 to 10 in the main program
return	RTN	K	return to block N + 1 of main program from SBR K is meaningless

INPUT/OUTPUT

function	instruction		remarks
heading	HDR	T1, . . . T5	label TTY/L.P. columns with up to 6 chrs. each
print/plot	PRI	N1, . . . N5	print/plot to TTY/L.P.
display* (scope)	DIS	N1, N2, N3, N4	X - axis = N1/N2 Y - axis = N3/N4

digital* to analog conv	DAC	N1, N2, K	R = N1/N2
			K = channel no.

analog* to digital conv	ADC	K	R = ADC output
			K = channel no.

CONTROL

function	instruction		remarks
change mode	CHM	K	reverses mode IC to C or C to IC
count	CNT	K	when iteration count = K skip next instr and reset count to K
end	END	K	reverse mode when in I and go to block O otherwise terminate comp
finish	FIN	N1, N2	If N1≥N2 take next instr otherwise up-date time and go to block O
go to	GTO	Nk	go to block Nk
transfer	TFR	N1, N2, N3	if N1≥N2 go to N3

LOGICAL

function	instruction		remarks	

input relay	IRL	N1, N2, N3, N4	R = N3
			R = N4
				N1≥N2
				otherwise

limiter	LIM	N1, N2, N3	R = N1 = N3 = N2	N3>N1 N1 ≤ N3≤N2 N3<N2
step size change	SCH	N1, N2, N3, K	if N1≥N2	S = N3 and T = K

SPECIAL FUNCTIONS

function	instruction		remarks	
comment*	CCC	EXAMPLE COMMENT	one line of comment	
function generation	FNG	N1	R = f(N1)	
increment	INC	N1, N2	R = ∑N2 + N1 = N1 in I C mode	
integrate	INT	N1, N2, . . . N9	R = $\int$(N2 + . . . N9)dt + N1	
variable time delay	TDL	N1, N2, N3	R = f(N1)(T - 20N3/N2) = f(N1)(T - 20S)	N3≥S · N2 N3<S · N2
time	TME		each iteration advances time by S	

program preparation and modification	
A	Append program
E	Enter program from external device
F*n*, *m*	edit line *m* of FNG *n*
J	enter Job (program) title
K	Kill (erase) current program
L	List current program
L*n*	List statement *n*
L*n*, *m*	List statements *n* through *m*
M*n*	Modify statement *n*
M*n*, *m*	Modify statements *n* through *m*
R	print Remaining memory locations
W	Write to external device
space	define a CON with current block no. exit append
CR	terminates all program statements

program operation	
B	Back to numerical listing from plot
C	enter Compute mode
I	set Initial conditions
H	go to Hold (sense switch A for EAI computers)
P	Parameter read and set
Q*n*	integrator selection (Quadrature) *n* = 0 EULER *n* = 1 RUNGE-KUTTA 2 *n* = 2 RUNGE-KUTTA 4
S	select integration Step size
T	select Typing frequency
U*n*	select output Unit *n* = 0 teletype *n* = 1 line printer
V	read numerical Value
X	select plotting routine and set range

NOTE: NUMERICAL ENTRIES ARE TERMINATED WITH A SPACE, COMMA, OR CARRIAGE RETURN (CR).

TABLE 2. ISL Monitor Commands