ELLIOTT 900 SERIES SIMULATOR

WRITING SIR CODE PROCEDURES FOR ALGOL.

SIR procedures can be linked to ALGOL programs by following the conventions for ALGOL code procedures. In the ALGOL source code, the code procedure should be declared using the code ... ALGOL syntax, for example:

```
     code
          integer procedure example (x);
          integer x; value x;
     algol
```

A SIR program with a global label corresponding to the procedure name can be loaded by the Interpreter (use entry at 11).

The SIR program should take the following form and be assembled to relocatable binary format:

```
*1
((TITLE)
[PROC]
PROC /14  n         (where n is number of formal parameters)
     >1             (space for return link)
     ...            (code)
     ...
     0    PROC+1    (pick up return link)
     /8   1         (return)
```

Note the program must only have one global label and no subglobals. Patches and restores are not allowed.

Important Addresses.

	Issue <= 5	Issue >= 6
QACODL (constants area)	32	132
EP (entry pointer)	37	137
FP (formal pointer)	38	138
PBA (primitive base address)	40	140
W (workspace)	80	180

On procedure entry, FP contains address of parameters, each parameter occupying 3 words, numbered from 0. (Parameter 0 is used for the return value from a function procedure, the first actual parameter is numbered 1.)

PBA points to an area containing runtime routines for ALGOL interpreter arithmetic routines.

N.B., The following text assumes the conventions for ALGOL issue 5 and earlier.

Parameter formats.

Scalar parameters

Type	Mode	3n+[FP]	Contents of 3n+1+[FP]	3n+2+[FP]
integer/ boolean	Name	Address	Undefined	Undefined
	Value	Value	Undefined	Undefined
real	Value	Mantissa*	Mantissa*	Exponent
	Name	Address**	Flag***	Undefined

true is encoded as +1, false as 0.

*   Mantissa consists of two words.
** Bit 18 of the address of a real passed by name is set to 1.
*** Flag is >0 if the real is packed (i.e., occupies two words),
    <0 if unpacked (i.e., occupies 3 words).

Reals passed by value are always unpacked, reals passed by name are packed, unless the actual parameter to the code procedure is itself a formal parameter called by value.

Array Parameters.

Array Type	3n+[FP]	Contents of 3n+1+[FP]	3n+2+[FP]
integer/boolean	Address	Undefined	Undefined

real	Address	Undefined	Undefined

Arrays are stored in column order. The address points to an array descriptor. The first word of the descriptor points to the address of the first element of the array (with bit 18 set if it is a real array). The second word of the descriptor points to an array map. An array map consists of 2d+2 words for an array with d dimensions. The first word of the map contains the number of dimensions as an integer, the second the total size of the array in words, the third is an offset and the fourth the lower bound of the first dimension of the array. These words are followed by pairs of words for each further dimension of the array. The first word of each pair is the length of the previous dimension (in words), the second is the lower bound of the further dimension.

The address of an element x[i, j, k, ...] is

```
     Address of first element + offset +
          F * i +    (where F is 2 for real arrays and
                                 1 for integer arrays)
               Range1 * j +
                    Range2 * k
```

Thus for a [1:10, 1:20] integer array, the map consists of the following values:

+1	Dimensions
+100	Total Size
-1	Offset (first element is [1,1])
+1	Lower bound for first dimension
+10	Length of first dimension
+1	Lower bound for second dimension.

Label Parameters.

If the n-th parameter is a label, 3n+[FP] contains the address of the label. To branch to an actual label parameter, use the following code to set up a goto as the return from the procedure.

```
          4    132       (QACODL)
          0    138       (FP)
          /2   (n*3)     (relative address of label to QACODL)
          6    +8191     (remove function bits)
L1        1    =10 0     (convert to goto instruction)
          5    VALUE     (store instruction)
          4    ADDRESS   (address of instruction)
          0    137        (EP)
          /5   1          (Save at EP+1)
          ...
          0    PROC+1     (pick up return link)
          /8   1          (return)
VALUE     +0
ADDRESS   0    VALUE
```

Switch parameters.

If the n-th parameter is a switch, 3n+[FP] contains the address of a sequence of locations, the first is the size of the switch parameter in words, followed by two words for each entry in the switch, the first of each pair being the address of the corresponding label.

To exit via the M-th label in the switch, use the following:

```
          4    132        (QACODL)
          0    138        (FP)
          /2   3*n        (switch parameter [3n+FP]-QACODL)
          1    M
          1    M          (2*M)
          1    -1         (first element is size)
          8    L1         (see example above)
```

The following code can be used to verify the switch index is within range:

```
          4    M
          7    FAIL       (M = 0)
          9    FAIL       (M < 0)
          0    138        (FP)
          /0   3*n
          4    M
          /2   0
          9    FAIL       (M too large)
```

String parameters.

The contents of 3n+[FP] is the starting address of the string, stored with opening and closing quotes, packed in SIR internal code, left justified.

Result

Result Type	Contents of [FP]	1+[FP]	2+[FP]
Integer/Boolean	Result	Undefined	Undefined
Real	Mantissa	Mantissa	Exponent
None	Undefined	Undefined	Undefined

ALGOL Interpreter Routines.

The primitive base address (PBA) points to an area containing the addresses of subroutines to carry out the ALGOL arithmetic operators. In the following table I1, I2, R1, R2 are assumed to be a workspace, declared as followed:

```
I1 R1     >+3
I2 R2     >+3
```

The address of this workspace should be stored in W location 180.
Reals should be stored unpacked in the workspace.

Routine	Address Relative to PBA
I1:=I1+I2	30
R1:=R1+R2	31
I1:=I1-I2	32
R1:=R1-R2	33
I1:=I1*I2	34
R1:=R1*R2	35
R1:=I1/I2	36
R1:=R1/R2	37
I1:=I1^I2	38
R1:=I1^I2	39
R1:=R1^R2	40

(Note Elliott manuals also list 28 R1 := R1^I2 28 but it does not work and interpreter code shows PBA+28 does not have a link address to support a call from a code procedure.)

Normal overflow checks are included. Results in all cases except 38 and 39 are also stored in locations 183, 184, 185.

Thus, to multiply two reals:

```
          4    WSADD
          5    180        (store address of workspace)
          0    140        (PBA)
          /0   35         (R1*R2 routine)
          /11  0          (store link)
          /8   1          (jump to routine)
          ...
WSADD
I1 R1     >3
I2 R2     >3
```

Other useful routines in the Interpreter can be used to pack and unpack reals and to convert between integers and reals.

<u>Pack.</u>

```
          ...             (store unpacked real in 183, 4, 5)
          0    174
          /11  0
          /8   24
          ...             (result in 103, 104)
```

<u>Unpack.</u>

```
          0    ADD        (address of packed real)
          /4   0
          5    UNPACK
          /4   1
          6    &377600    (eliminate packed exponent)
          5    UNPACK+1
          /4   1
          14   11         (eliminate fractional part)
          14   8181       (shift back, regenerating sign bit)
          5    UNPACK+2
     ...
UNPACK    >3
ADD       0    VALUE
VALUE     >2              (packed real)
```

Real to Integer conversion (ENTIER).

(Includes overflow check)

```
          ...            (store real in 183, 4, 5)
          0    157       (address of ENTIER routine)
          /11  0
          8    158
          ...            (result in 83)
```

Integer to real conversion.

```
          4    I
          5    83        (store as mantissa)
          4    +0
          5    84
          4    +17       (integer = 0.xxx * 2^17)
          5    85
          0    150       (address of standardize routine)
          /11  0
          /8   1
          ...            (unpacked result in 83, 4, 5)
```

SIR procedures written according to the above rules should be compiled using a non-load-and-go SIR assembler to produce an intermediate relocatable binary tape suitable for loading at run time.

ADDING FURTHER I/O DEVICES TO THE ALGOL INTERPRETER.

The 903 ALGOL interpreter has an extensible i/o system that permits additional devices to be added. The device should be assigned a device number N in the range 1-10, not assigned to any other device.

The conventional Elliott assignment is:

1	Paper tape reader / punch
2	Reserved
3	Teleprinter
4	Line printer
5	Plotter
6	Card reader/punch
7	Not allocated
8	Not allocated
9	Not allocated.

The user must write a device routine for the device together with a machine code set up procedure to load the address of the device routine into the appropriate table of the ALGOL interpreter. To address the device, an ALGOL program must firstly make a call to the appropriate setup procedure. Thereafter the device may be selected by the standard statements reader(N) and punch(N).

Setup procedure for an input device.

The set up procedure for input device, number N must:

1. Load +0 into the location (BUFFER+N-1) if location (INSLOT+N-1) does not already contain the address of the device routine.
2. Load the address of the device routine into location (INSLOT+N-1)

Setup procedure for an output device.

The set up procedure for output device, number N must:

1. Load the address of the device routine into location (OUTSLT+N-1)

Device routine entry and exit.

Output routines will be entered with the character to be output either as SIR internal code or as a negative number with an 8 bit binary code for output in the l.s. digits of the accumulator.

(The second option is not documented in the Elliott ALGOL manual, however it will be generated by a binary code in an inner string, e.g., ''B24@@, or a multiple newline code, such as ''L3@@).

Input routine should exit with the character input, in SIR internal code in the accumulator.

While not documented in the Elliott ALGOL manual it also appears necessary to store the character location (BUFFER+N-1) and (if instring is to be used) also in location 73 (issue 5), 173 (issue 6).

The addresses of the locations in the interpreter are documented in the Elliott ALGOL manual as follows:

Issue 5 (903 and MASD ALGOL 16K load-and-go systems)

```
ICHLNK    = 129
PCHLNK    = 130
BUFFER    = 131
INSLOT    = 141
OUTSLT    = 151
```

(Empirically however it appears to be the case that ICHLNK = PCHLNK = 130).

Issue 6 903, MASD and HUNTER systems)

```
ICHLNK    = 229
PCHLNK    = 230
BUFFER    = 231
INSLOT    = 241
OUTSLT    = 251.
```

For examples of using these facilities see HUNTER/DEMO13, HUNTERP/DEMO13, 903ALGOL/DEMO11A, 903ALGOL/DEMO11B, MASDALGOL/DEMO11A, MASD/DEMO11B which are all variants of a the same program illustrating use of an Elliott 4100 card reader and an Elliott 4100 line printer from ALGOL.