AJH Algol Interpreter Notes

Andrew Herbert
2nd February 2015

These notes document the AJH version of the Elliott 903 Algol interpreter. It is derived from the Hunter Algol interpreter itself is an extension of the Elliott Issue 6 interpreter with additional facilities for handling recursive procedures and a wider range of actual argument types for call-by-name procedure parameters.

Introduction

Note: since starting this document I have obtained copies of original Elliott design documents for the Algol system. These describe the 2-pass 8K system up to issues 4 and 5, from which issue 6 was derived. It is therefore useful to read these as background:

> *Elliott 903 Algol: Object Code Manual. (June 1966).*
>
> This document, written by Don Hunter, defines the overall architecture, data structures and function of the Algol interpreter, including a specification of the Algol intermediate code (called *parameter words* or *pords*).
>
> *The 903 Algol Interpreter. (Undated).*
>
> This document gives a routine by routine description of the interpreter with both narrative and flow charts. There are some handwritten annotations showing corrections and revisions.

As this document is further update I will update the terminology herein to match that in the Elliott documents listed above.

The Algol interpreter has a stack based architecture: it follows the broad principles of the Whetstone Algol system described in the classic text "Algol 60 Implementation", by Randell and Russell (Prentice Hall, 1964). The interpreter stores intermediate code instructions as 903 words, called *parameter words* or *pords* for short. Pords can either be single or no-address. There are 30 single address prords in the form of 5 bits of function code and 13 bits of address. There are just over 60 zero address pords: these have the value 31 in the top 5 bits and the remaining 13 bits specify the particular operation. The function codes are tabulated below. It is convenient to represent the pords as if they were

machine function codes, using the modifier bit to extend from 16 to 31, thus the zero address instructions all have function code /15 .
Note: AJH and Hunter Algol use a slightly modified set of interpreter operations compared to 903 Algol Issue 6 although much in this document still applies to the earlier system. This is noted in the function table below.

In the description following we first identify a set of key global variables that point to, for example, the top of the stack. Most operations affect these in one way and another.

We then show the standard representations of the various data types used by the interpreter.

This is followed by the stack representations used: all data items on the stack consist of three word entries. Note the number of different kinds of "address" representation available and the various flag bits that signal what kind of address is being represented.

Finally a large number of example programs are shown in original source and intermediate code form with annotation.

Blocks, Procedures and Functions

Blocks (e.g. the main program), procedures, functions and for loops are all handled the same way on the stack. Each block is allocated a unique identifying "block number" by the translator. Block numbers are always a multiple of 16. When a block is entered by the PE (block/procedure/function entry) or FOR (for loop entry) instruction, the block number of is stored the variable BN.

Note: the translator collapses inner blocks that do not contain array declarations, sine such blocks do not require any dynamic storage to be allocated.

When a block or procedure is started a new "entry" is created on the stack and the address of the start of the entry is stored in the variable EP. All entries record the address of the previously active entry, i.e., the calling block, (word 0, denoted EP'), the interpreter program pointer for the return from the newly entered block (word 1, PP'), the previous (top of stack) pointer (word 2, SP') and the previous entry's block number (word 3, BN').

Hunter Algol introduces an "environment pointer" (word 4, EVN) that relates to the handling of formal procedure parameters in recursive calls. The environment of a procedure is the set of the names accessible at the point the procedure was declared. The nesting of procedures within one another therefore sets up a "static chain" of such environments and it is important that names of formal parameters are correctly resolved with respect to the static structure. However, since procedures can be passed as parameters, the dynamic

call nesting can be different to the static nesting. This complicates the resolution of parameter names, especially when there is also recursion involved.

When a procedure/function is called, a pointer to the environment of the current procedure/function is stacked along with the entry pointer EP. If the called procedure is referenced statically EP' will match EVN'. However if the called procedure is referenced dynamically (i.e., via a procedure parameter) EP' will not be equal to ENV' if the procedure parameter was defined in an outer scope.

Operations that access formal parameters take an address field consisting of a 9 bit block number and a 4 bit argument number. The algorithm for locating the formal is as follows:

1. If the current block number (BN) matches the formal block number, take the argument from the current entry.
2. If the current entry is a for loop then, if BN' equals the formal block number, take the argument from the previous entry on the stack, otherwise got to step 5.
3. If the current entry is a procedure call, EP' equals ENV' and BN equals the formal block number, take the argument from the previous entry on the stack, otherwise go to step 5.
4. If the current entry is a procedure call and EP' is not equal to ENV', then save EP' as LOC and follow down the stack until an entry is found with EP' equal to LOC. If BN' of this entry equals the formal block number take the argument from the previous entry on the stack, otherwise go to step 5.
5. Make the previous entry on the stack the current entry and go to step 2.

When a "goto" is executed a similar algorithm must be used to unwind the stack to the environment in which the label is defined:

1. If the current block number (BN) matches the label block number, jump to the label address.
2. If the current entry is a for loop then, if BN' equals the label block number, make the previous entry current and jump to the label address, otherwise got to step 5.
3. If the current entry is a procedure call, EP' equals ENV' and BN equals the formal block number, make the previous entry current and jump to the label address. If a previous step 5 has noted a match, make the associated entry current and jump to the label address, otherwise go to step 5.
4. If the current entry is a procedure call and EP' is not equal to ENV', then save EP' as LOC and follow down the stack until an entry is found with EP' equal to LOC. If BN' of this entry equals the label block number make the previous entry current and jump to the label address, otherwise go to step 5.
5. If BN' in the current entry matches the label block number, note the address of the entry. Make the previous entry on the stack the current entry and go to step 2.

For Statement Blocks

A new entry is created when a for statement is executed. As with a basic block or procedure entry, word 0 points to the calling entry, i.e., the surrounding block. Word 1 points to the address of the pords for the next for list element (i.e., first for list element on first cycle, the second for list element on the second cycle and so on). The last for list element is followed by a special pord which causes exit from the entire for block and execution resumes at the pord address given by word 4. Word 2 contains the address of the pords for the controlled statement (with bit 18 set so a for entry can be distinguished from a block/procedure entry). Word 3 contains the block number. Word 5 is always the address of the controlled variable.

A while or step-until for statement continuously executes the controlled statement indicated by word 2 until the terminating condition is met, whereupon control is transferred to the address in word 4.

Call by Name

903 Algol issue 6 only allows identifiers and constants as scalar call by name actual parameters. There are address stack representations for scalars and constant "addresses" to cover these cases. Hunter Algol additionally allows subscripted variables and arbitrary expressions as actual parameters. (Assignment to expressions passed by name generates run-time error 21 – expressions passed by name can only be used as RValues). The basic technique for handling subscripted variables and expressions is to generate a short anonymous procedure or "thunk" that is called to generate a value each time the formal name is referenced, using `GETAD` (if the LValue is required for the left hand side of an assignment) or `TRCN` (if the RValue is required in an expression evaluation). The address of the thunk code is referenced from an address representation which looks much like a procedure address representation – the environment being that of the calling block. When the formal parameter is referenced the thunk is executed and a result returned, initially as an LValue, but in the case of the `TRCN` pord this is then dereferenced to extract the desired RValue. This can be a recursive process if the actual parameter expression references variables that are formal call by name parameters of wider scope.

There is some complication to the generation of thunks by the translator that requires fixing up at runtime. After the translator has emitted a thunk, the intermediate code is followed by a RLB `Halt 15` directive. This runs code in the interpreter that makes the necessary adjustments and then resumes loading further intermediate code.

The general form of a thunk when generated is:

```
UJ          start      (becomes UJ end)
```

```
start     code for expression
          ST
          RETURN
          PE        16        (call thunk)
          RFUN      16        (or IFUN – address of thunk
                               Result)
          TA        start     (becomes UJ start)
end       MKTHK     type      (integer, real or label)
```

As the thunk is read in the `Halt 15` code deals with updating the first and last pords with the correct addresses.

Absolute and Relative Addressing

The LP Algol system has some additional complications since it allows the Algol intermediate code to cross from store module 0 to module 1, up to a maximum program length of 8192 words. (N.B., code procedures cannot extend beyond location 8191.) When the two pass or load-and-go Algol systems load intermediate code they convert addresses to absolute form. When the LP system loads intermediate code it converts addresses to be relative to the start of the program. Thus any pord that references a program location, e.g., `UJ 35;` has to have the program base address added to the address field from the pord to reference the correct location in store. This creates a problem for references to code procedures since these use absolute addressing. (This can only occur with the `CF` (call function) and `TA` (take address) instructions). The LP system loader recognizes when the pord types are used in conjunction with a global label and modifies the instruction.

In the 903 Algol LP system there is a spare pord type which can be used (and only the `CF` case arises). In Hunter Algol this spare pord type is used for `GETAD`, so in the AJH LP system the `PEM` and `INOUT` pords are overloaded with new pords "`CFSIR`" and "`TASIR`".

The maximum address field in an ordinary `PEM` pord is 14, and for `INOUT` 20, so addresses larger than 31 for these instructions are treated as `CFSIR n-32` and `TASIR n-32` respectively. Thus code procedures can still be loaded up to location 8179, but no global names can be loaded above location 8160.

Globals

`BN`	Current block number
`EP`	Current environment frame
`FP`	Arguments pointer (only set on code procedure call)
`PP`	Program pointer (next interpreted instruction to execute)
`SP`	Stack pointer

QACODL Algol Constants Object Data Load: pointer to area of store allocated to hold Algol constants

QAVNDA Algol Variables Notional Data Area: pointer to area of store allocated to hold program variables.

Note: since Elliott Algol treats variables as static they have fixed locations. Only arrays and procedure/function call arguments are allocated dynamic space on the interpreter stack.

Data Representations

Arrays

Descriptor (in QAVNDA)

Word	Name	Notes
0	address	absolute address of data + &400000 if real
1	map	absolute address of map

Map (on stack)

Word	Name	Notes
0	dimensions	
1	total	total length in words
2	offset	
3	lwb 1	
4	size 1	length of previous dimension in words
5	lwb 2	
6	size 2	
7	etc, etc	

Integers

Constants stored in QACODL, variables in QAVNDA as single word.

Labels

Stored in QACODL as two words.

Word	Name	Notes
0	Address	relative to program
1	Block	

Reals

Constants stored in QACODL, variables in QAVNDA as two word packed form.

Word	Name	Notes
0	Sign and mantissa m.s. bits	Bit 18 sign. Bits 17-1 mantissa m.s.
1	Mantissa l.s.bits| Exponent	Bit 18 zero. Bits 17-8 mantissa l.s. Bits 7-1 exponent

Switch

Stored in QACODL as a vector.

Word	Name	Notes
0	size	number of labels in switch.
1+	labels	vector of indices of labels in QACODL.

Stack Representations

Address (integer) [from IFUN] – address of integer formal argument

Word	Name	Notes
0	address	absolute address + &400000
1	type	+1
2		+0

&400000 marker to indicate cannot be assigned to.

Address (integer) [from TIA – address of variable in QAVNDA]

Word	Name	Notes
0	address	absolute address
1	type	+1
2		+0

Address (label) [from TICA – address of constant in QACODL]

Word	Name	Notes
0	address	absolute address + &200000
1	type	+2
2		+0

&200000 flag indicates label.

Address (procedure) [from TA – address of procedure in program]

Word	Name	Notes
0	address	absolute address

1	type	undefined
2	env	Environment

Address (real) [from TRA – address of variable in QAVNDA]

Word	Name	Notes
0	address	absolute address + &400000
1	type	+2
2		+0

&400000 flag to indicate cannot be assigned to.

Address (real) [from RFUN] – address of real formal argument

Word	Name	Notes
0	address	absolute address + &400000
1	type	/0 2
2		+0

&400000 flag to indicate cannot be assigned to.

Address (real) [from TRCA – address of constant in QACODL]

Word	Name	Notes
0	address	absolute address + &400000
1	type	/8 0
2		+0

&400000 flag to indicate in unpacked form.
&200000 flag to indicate cannot be assigned to.

Address (string) [from TA – address of string in program]

Word	Name	Notes
0	address	absolute address
1	type	undefined
2	EP*	EP of containing block

Address (switch) [from TICA – address of switch in QACODL]

Word	Name	Notes
0	address	absolute address + &200000
1	type	+2
2		+0

&200000 flag indicates label.

Address (thunk) [from MKTHK – address of PE in thunk in program]

Word	Name	Notes
0	address	absolute address of thunk
1	type	+1 integer array

		+2 real array +4 integer scalar +5 real scalar +9 label
2	environment	stack entry current when thunk created. Will have bit 18 set in the case of a partially evaluated array thunk.

Integer/Boolean value

Word	Name	Notes
0	value	
1		undefined
2		undefined

Real (unpacked)

Word	Name	Notes
0	Sign | Mantissa m.s.	Bit 18: sign Bits 17-1: mantissa m.s.
1	Exponent	Bit 18: zero Bits 17-1: mantissa l.s.
2		Exponent

This format is used for reals during expression evaluation.

Stack Frame Structure

Procedure / Block Entry Frame

Created by CBL, CF, CFSIR and CFF/PE. Removed by RETURN

Word	Name	Notes
-3n	FP	3 words for result
	FP+3n	3 words argument n
0	EP'	previous entry pointer
1	PP'	previous program pointer
2	SP'	previous stack pointer
3	BN'	previous block number
4	EVN'	previous EVN
5+		working stack for procedure

FOR Statement Entry

Created by FOR, removed by FSE.

Word	Name	Notes
0	EP'	previous entry pointer
1	Next	address of next for list element
2	Body	address of body ! &400000
3	BN'	previous block number
4	Exit	address of next instruction after loop
5	Var	controlled variable address (bit18 set for real, unset for integer)
6	Packing	<0 unpacked, >=0 packed
7	Flag	0 first time round <>0 otherwise
8	Step1	(3 words)
9	Step2	
10	Step3	
11	Limit1	(3 words)
12	Limit2	
13	Limit3	

Note: on first entry "next" for step element is the initial step element.

Parameter Block

Because formal procedure parameters do not have their formal parameters specified, these have to be checked at runtime. Similarly for array dimensions. Therefore each procedure entry (PE) is followed by a block of descriptors, one for each parameter. The descriptors are coded as follows:

1 0: integer
2 0: real
3 n: integer or Boolean array of dimension n
4 n: real array of dimension n
5 n: integer or boolean function with n arguments
6 n: real function with n arguments
7 n: procedure with n arguments
8 0: switch
9 0: label
10 0: string

/ indicates value parameter (integer, integer array, real, real array, label)

For types 1 and 2 a check is made that the second word of the actual argument matches the opcode, (i.e., type check) or that the type is 5 (integer function) or 6 (real function) as appropriate.

For types 3 and 4 a check is made that the actual argument has the correct number of dimensions (by inspecting its array map).

For types 5-10 a check is made that a corresponding CONn is stacked upon procedure entry.

For types 5, 6, 7 a check is made that the actual argument address points to a procedure with the correct number of parameters.

In the case of /3 (integer array) and /4 (real array), a local copy of the actual argument array is made on the stack.

Intermediate Code

The 2-Pass 903 and Hunter Algol translators both output intermediate code in a subset of the standard Elliott "relocatable binary" (RLB) format. (The Hunter Load and Go system can also do so optionally).

The code consists of 8 bit triples:

```
ABCDE.FGH
IJKLM.NOP
QRSTU.VWX
```

A, I, Q are always zero.
BCD is a loader code.
EFGHJKLMNOPRSTUV are an 18 bit data word.

The following loader facilities are used:

`Code 1, word`	Load word as it stands
`Code 2, word`	Load word after adding base address. N.B. LP Algol uses separate base addresses for own code and machine code.

`Code 3, location`
`3 Blanks`
`3 Blanks`
	Update an implicit jump such as needed around a procedure body or along a conditional instruction. The data word specifies the relative address of the word to be updated. This word is updated with the address of the location being loaded. The code 3 word is followed by 6 blanks.

`Code 3, location`
`offset`
`3 blanks`
	Update an array or procedure checking word.

`Code 4,` `£ABC`	Left hand global labels QACODL and QAVNDA.

Sub Code 1, £DEF	
Code 4, £ABC Sub Code 2, £DEF Pord word 3 blanks	Reference to library procedure, either in a call (CF) or as a parameter (TA).
Code 5	Skip n locations – used to reserve data space following the program.
Code 6	Checksum.
Code 7	Stop loading and print FIRST NEXT message.

Parameter Words (Hunter/AJH Algol)

Function	Meaning	Stack	Action
CBL /15 1	Call Block		SP!1 := PP EP := SP PP := PP+2 SP +:= 2 Followed by a UJ instruction to next statement.
CHECKB /15 2	Check Boolean	value	Algol checkb function. punch * newline value.
CHECKI /15 3	Check Integer	value	Algol checki function. Punch * newline value.
CHECKR /15 4	Check Real	value	Algol checkr function. Punch * newline value.
CHECKS /15 5	Check String	string address 10	Algol checks procedure. Punch * newline string. SP -:= 3

CF n /5 n	Call Function		Create entry for procedure/function call. ENV := EP SP!0 := EP SP!1 := PP EP := SP SP +:= 2 Jump to n in program (will be PE/PEM of procedure/function). PP := n + base address
CFF BN+n /6 n	Call Formal Function n[12-5]: block no B n[4-1]: arg no A		Locate FP using B+A. Then as CF but with jump to address of formal procedure (contents of 3A+FP) and ENV set to contents of 3A+2+FP.
CFSIR n /14 n	Call Function (for code procedure)		32 <= n < 8160 As for CFSIR but note n is relative to code procedures not own code.
CON+n /15 63-70	Take Immediate		Stack integer value n-60. Used to set up parameter descriptors, e.g., for strings.
DIV /15 12	Divide (Integer)	i j	Algol “div” operator i/j. i := i “div” j; SP -:= 3
DO /15 6	For Statement Body	address of controlled variable next value for controlled variable	Store next value in controlled variable. ASSIGN(); Set next for list element. EP!1 := PP Jump to loop body. PP := EP!2
FINISH /15 8	Finish program		Algol stop procedure. Punch FINISH and halt execution.

FOR body block exit /15 9	For Statement FOR is followed by 4 words: body: address of loop body block: loop block number exit: statement following loop		Create and initialize new for loop entry. SP!0 := EP SP!1 := PP := PP+4 (first for list element) SP!2 := body + &400000 SP!3 := BN SP!4 := exit EP := SP SP +:= 5 BN := block Then execute following TIA or TRA instruction to stack controlled variable address.
FR /15 10	For Return		Execute next element of for loop. PP := EP!1
FSE /15 11	For Statement End		Exit from for for statement. BN := EP!3 PP := EP!4 (exit) SP := EP := EP!0
GETAD n /9 n	Get Address (of Call by Name Argument) n[12-5]: block no B n[4-1]: arg no A		Find FP using B+A If parameter at FP+3A is a thunk (word 2 <> 0) then call procedure at address FP+3A using CFF to calculate address result. Set environment of call to contents of word 2. Otherwise treat as in TF. Note: in 903 Algol this is never generated.
GT n 10 n	Go To		n is offset of label descriptor in QACODL, containing BN, label address. Unwind stack to innermost occurrence of label block number. PP := address of label.

GTF n 11 n	Go To Formal n[12-5]: block no B n[4-1]: arg No A		Find FP using B+A. Then as for GT with content of FP+3A as offset of label descriptor in QACODL.
GTFS n 14 n	Go To Formal Switch n[12-5]: block no B n[4-1]: arg no A	index	Find FP using B+A. Switch address := contents of 3n+FP. Then as for GTS.
GTS n 9 n	Go To Switch	index	n is offset of switch in QACODL. Select index-th label from switch, checking bounds. If index <= 0 or > contents of QACODL+n then fail. Label address := QACODL+n+index. Unstack index. SP -:= 3 Then as for GT.
IFJ n 7 n	If Jump False	condition (Boolean)	Unstack condition. SP =:= 3. if condition = 0 (false) then goto n+base address
IFUN n /12 n	Integer Function n[12-5]: block no B n[4-1]: arg no A		Get address of function result or parameter. Find FP using B+A. Set up read only address SP!0 := FP+3A + &400000 SP!1 := 2 SP!2 := 0 SP +:= 3

INDA n 12 n	Index Address n = number of indices on stack, in words	array address index1 index2 ...	Check array bounds. Unstack indices. SP -:= 3*n+3 Stack address of indexed element of array. SP!0 := address SP!1 := 2 for integer, /0 2 for real SP!3 := +0 SP +:= 3
INDFS n 5 n	Index Formal Switch n[12-5]: block no B n[4-1]: arg no A	index	Find FP using B+A, then as for INDS.
INDR n 13 n	Index Result n = number of indices on stack, in words	array address index1 index2 ...	Check array bounds. Unstack indices. SP -:= 3*n+3 Stack value of indexed element of array. SP +:= 3
INDS n /11 n	Index Switch	index	n is offset of switch in QACODL. Check index in bounds. Unstack index Stack address of indexed label.

INOUT n 15 n	Input / Output	Parameters as required	1: read integer 2: real real 3: output integer 4: output real 5: aligned - global 6: punch – global 7: digits - global 8: freepoint – global 9: spare 10: spare 11 prefix – global 12: sameline - global 13: scaled - global 14: reader - global 15: output string 17: punch - local 16: aligned – local 17: punch – local 18: digits - local 19: freepoint - local 20: restore global settings 22: prefix - local 23: sameline - local 24: scaled - local 25 reader - local
ITOR1 /15 13	Integer to Real	value to convert	Convert value on top of stack.
ITOR2 /15 14	Integer to Real	value to convert some other value	Convert value under top of stack.

MAMPS n Address 6 n	Make Array Maps n = 64 * dimensions + number of arrays required. Pord followed by word giving offset of array descriptor in QAVNDA with bit 18 set if real.	SP[0]: lwb1 SP[3]: upb1 SP[6]: lwb2 SP[9]: upb2 etc etc	Allocate stack for descriptor, map and data of each array. One word per element for integer arrays, two for real. Array map: dims total size offset l1 c1 l2 c2 etc offset is address of nominal element [0, 0, ...]. li is lower bound i. ci is range of subscript I * 1 if integer, 2 if real. Descriptor (in QAVNDA) address of map address of array. Unstack bounds pairs. SP -:= 3*n
NEGI /15 15	Negate Integer	value	Negate value on top of stack.
NEGR /15 16	Negate Real	value	Negate value on top of stack.

PE n /7 n	Procedure Entry n[12-5]: block no B n[4-1]: no of args A followed by one word per argument		Finalize content of procedure/function entry. SP!0 := EP-3n SP!1 := BN SP!2 := ENV SP +:= 3 BN := B Perform parameter checking for block of A argument codewords following. PP += A+1
PEM /14 n	Procedure Entry Machine n (<15): no of args		Create frame for call of machine code procedure, as for PE, but with SP!3 set to current BN. Enter machine code via 0 PP /11 0 /8 1 Fall into RETURN on exit
RETURN /15 17	Return from Procedure		Unstack function / procedure entry. BN := EP!3 SP := EP!2 EP := EP!0
RFUN n /13 n	Real Function n[12-5]: block number B n[4-1]: arg no A		Find formal argument B+N. Stack address contained in FP+3A as if from TRA but with read only marker set. SP!0 := FP+3n + &400000 SP!1 := /0 2 SP!3 := +0 SP+:= 3
RTOI /15 18	Convert real to integer	value to convert	
ST /15 20	Store	address value	Store value in address, according to flag and indicator in address. ASSIGN () SP -:= 3

STA /15 21	Store Also	As ST	ASSIGN () SP!-3 := SP SP!-2 := SP!1 SP!-1 := SP!2
STEP /15 22	For Statement First Step	controlled variable address	Store value in controlled variable and unstack it. ASSIGN () Set next to following instruction. EP!1 := PP Set first time flag in controlled variable address. EP!7 := 0
STEP2 /15 25	For Statement Next Step	value	if not first time store value in controlled variable. if EP!7 =0 then ASSIGN() clear first time flag. EP!7 +:= 1
STW /15 07	Store While	address	As ST but only unstack value. ASSIGN ()
TA n 0 n	Take Address		Stack address of string or procedure at offset n in program. SP!0 := n+base address SP!2 := EP SP +:= 3
TASIR n 15 n	Take Address (of code procedure)		32 <= n < 8160 Stack address of code procedure at address n. SP!1 := n + base address SP!2 := EP SP +:= 3
TF n /8 n	Take Formal n[12-5]: block no B n[4-1]: arg no A		Find FP using B+A. Stack a copy of address from FP+3A. SP!0 := contents of FP+3A SP!1 := contents of FP+3A+1 SP!2 := contents of FP+3A+2 SP +:= 3

TIA n 1 n	Take Integer Address		Stack address of integer variable at offset n in QAVNDA. SP!0 := QAVNDA+n SP!1 := +1 SP!2 := 0 SP +:= 3
TIC n /2 n	Take Integer Constant		Stack value of integer constant at address n in QACODL. SP!0 := contents of QACODL+n SP +:= 3
TICA n /1 n	Take Integer Constant Address		Stack address of integer constant at offset n in QACODL and set read-only bit. SP!0 := QACODL+n + &400000 SP!1 := +1 SP!2 := 0 SP +:= 3
TIR n 2 n	Take Integer Result		Stack integer value at offset n in QAVNDA. SP!0 := contents of QAVNDA+n SP +:= 3
TLA n /1 n	Take Label Address		Synonym for TICA.
TRA n /3 n	Take Real Address		Stack address of real variable at offset n in QAVNDA SP!0 := QAVNDA+n+400000 SP!1 := 2 SP!2 := 0 SP +:= 3

MKTHK n /0 n	Create Thunk n = 1: integer expression n = 2: real expression n = 9: label expression		Create thunk out of preceding sequence of instructions. Instruction prior to TA will be the offset of the first instruction in the thunk within the program. Stack thunk address. SP!0 := address SP!1 := type if pord followed by INDA then 1 or 2, else 3 or 4 for integer or real respectively. SP!2 := EP Note: in 903 Algol Issue 6 this is never generated.
TRC n /4 n	Take Real Constant		Stack value of real constant at offset n in QACODL. SP!0, SP!1, SP!2 := unpack contents of QACODL+n, QACODL+n+1 SP +:= 3
TRCA n /10 n	Take Real Constant Address		Stack address of constant at offset n in QACODL and set read-only bit. (Used to pass constant by name). SP!0 := QACODL+n+&600000 SP!1 := +2 SP +:= 3

TRCN n /10 n	Take Result Call Name Argument n[12-5]: block no B n[4-1]: arg no A		Find FP using B+A. If address at FP+3A is a thunk (FP+3A+2 <> 0), call it. Otherwise store contents of address at FP+3A at SP. If address < 0 then transfer contents of address+1 and address+2 to SP!1, SP!2, else unpack contents of address+1 to SP!1, SP!2. SP +:= 3
UJ n	Uncond-itional Jump		PP := n + base address
UNTIL /15 26	For Statement Until Condition	step limit	Unstack step (EP!8) and limit (EP!11). SP -:= 6 Test controlled variable (EP!5) against limit (EP!11). If work to do, jump to loop body. (Will fall into FSE if complete). PP := EP!2
UP /15 27	Extend Stack		SP +:= 3
WAIT /15 24	Wait		Algol wait procedure. Halt but prepare to resume after a start at 9.
WHILE /15 29	For Statement While Condition	condition	Unstack condition. SP -:= 3 If true jump to next. PP := EP!2 (Will fall into FSE if false).

ASSIGN	Subroutine	Flags:
		&400000 =>
		real value
		(vs
		integer
			Unstack value
			SP -:= 3
			address := SP!-3
			type := SP!-2
			
			Type:
			&400000 => real value (vs integer)
			&200000 => read only
			
			If read only FAIL.
			
			address!0 := SP!0
			
			if address < 0 then begin
			  if type < 0 then
			  address!1 := SP!1
			  address!2 := SP!2
			  else
			  address!2 :=
			  pack SP!1 and SP!2
			  with roundoff

/15 28	R := R^I	R, I	SP -:= 3
/15 30	I := I+I	I, I	SP -:= 3
/15 31	R := R+R	R, R	SP -:= 3
/15 32	I := I-I	I, I	SP -:= 3
/15 33	R := R-R	R, R	SP -:= 3
/15 34	I := I*I	I, I	SP -:= 3
/15 35	R := R*R	R, R	SP -:= 3
/15 36	R := I/I	I1, I2	SP -:= 3
/15 37	R := R/R	R1, R2	SP -:= 3
/15 38	I := I^I	I1, I2	SP -:= 3
/15 39	R := I^I	I1, I2	SP -:= 3
/15 40	R := R^R	R1, R2	SP -:= 3
/15 41	B := I<I	I1, I2	SP -:= 3
/15 42	B := R<R	R1, R2	SP -:= 3
/15 43	B := I<=I	I1, I2	SP -:= 3
/15 44	B := R<=R	R1, R2	SP -:= 3
/15 45	B := I=I	I, I	SP -:= 3
/15 46	B := R=R	R, R	SP -:= 3
/15 47	B := I<>I	I, I	SP -:= 3
/15 48	B := R<>R	R, R	SP -:= 3
/15 49	B := I>I	I1, I2	SP -:= 3
/15 50	B := R>R	R1, R2	SP -:= 3
/15 51	B := I>=I	I1, I2	SP -:= 3
/15 52	B := R>=R	R1, R2	SP -:= 3
/15 53	B := B&B	B, B	SP -:= 3
/15 54	B := B|B	B, B	SP -:= 3
/15 55	B := B=B	B,B	SP -:= 3
/15 56	B := B=>B	B, B	SP -:= 3

/15 57	B := ~B	B	
/15 58	ABS	R	
/15 59	ENTIER	R	
/15 60	EXP	R	
/15 61	LN	R	
/15 62	SIGN	R/I	

Typical Code Sequences

OPCODEnn relates to the files TEST_OPCODEnn in the TESTS/ALGOL/AJH folder.

The notation is roughly that of the output of my simulator's PrintAlgol command. Where it is not confusing to do so, some numerical addresses are replaced by the identifiers or constants to which they correspond.

Later examples are abbreviated to just show the salient points.

OPCODE01 Null Program

```
Opcode;
OPCODE=0;
"begin"
    0;    INOUT     20   (Reset I/O)
    1;    TIC       2    (push +3)
    2;    INOUT     17   (punch[3])
    3;    UJ        9;   (skip over string)
    4;    £{{L
    5;    £3}
    6;    £OPC
    7;    £ODE
    8;    £^^}
    9;    TA        4;   (push address of string)
   10;    INOUT     15   (punch string)
   11;    CBL
   12;    UJ        15;  (on return jump to FINISH)
   13;    PE        816
   14;    RETURN
   15;    FINISH
"end"
QACODL=16;                (constants)
   16;    TA   0    (+0)
   17;    TA   1    (+1)
   18;    TA   3    (+3)
Checksum +2875
QAVNDA=0;                 (variables)
    0; >+1
Checksum +465
```

```
Halt 0
```

OPCODE02 Assignments

```
“integer” i, j; “real” r, s;
i := 0;
   14;    TIA  1    (take integer address QAVANDA+1 [i])
   15;    TIC  0    (take integer constant QACODL+0 [0])
   16;    ST       (store)
i := j := 0.5;
   17;    TIA  1    (i)
   18;    TIA  2    (j)
   19;    TRC  3    (0.5)
   20;    RTOI1    (real to integer)
   21;    STA      (store, leave address)
   22;    ST       (store)
r := 0.5;
   23;    TRA  3    (r)
   24;    TRC  3    (0.5)
   25;    ST       store)
r := s := 1;
   26;    TRA  3    (r)
   27;    TRA  5    (s)
   28;    TIC  1    (i)
   29;    ITOR1    (integer to real)
   30;    STA      (store, leave address)
   31;    ST       (store)
r := r + i;
   32;    TRA  3    (r)
   33;    TRR  3    (r)
   34;    TIR  1    (i)
   35;    ITOR1    (integer to real)
   36;    R:=R+R
   37;    ST       (store)

QACODL=33;
   33;    +0
   34;    +1
   35;    +3
   36;    0.5
   37;    0.5
   38;    1.0
   39;    1.0
Checksum +4365
QAVNDA=0;
    0; >+7
```

OPCODE03 Conditionals

```
“if” “true” “then” “print” {t};
```

```
   14;    TIC        1     (true)
   15;    IFJ        21;
   16;    INOUT      20    (“then” ...)
...
“if” “false” “then” “print” {f} “else” “print” {t}
   21;    TIC        0     (false)
   22;    IFJ        29;
   23;    INOUT      20    (“then” ...)
...
   28;    UJ         34;

   29;    INOUT      20    (“else” ...)
...
   34; ...
QACODL=36;
   36;    +0         (false)
   37;    +1         (true)
```

Loops

OPCODE04 For list

```
“for” i := 1, 2 “do” “print” i;
FOR body block exit
i 1 DO
i 2 DO
FSE
body: ... FR
exit: ...

   14;    FOR
   15;    0          24;   (body)
   16;    +832             (block)
   17;    0          28;   (exit)
   18;    TIA        1     (i)
   19;    TIC        1     (1)
   20;    DO
   21;    TIC        3     (2)
   22;    DO
   23;    FSE              (FOR EXIT)
   24;    INOUT      20    (FOR body)
   25;    TIR  1
   26;    INOUT      3
   27;    FR               (FOR RESUME)
```

OPCODE05 FOR WHILE

```
“for” i := 1, i+1 “while” I < “do” “print” i;
FOR body block exit
```

```
i 1 DO
i 1 + STW
i 3 < WHILE
FSE
body: ... FR
exit: ...

   14;    FOR
   15;    0          30;  (body)
   16;    +832            (block)
   17;    0          34;  (exit)
   18;    TIA        1    (i)
   19;    TIC        1    (1)
   20;    DO
   21;    TIR        1    (i)
   22;    TIC        1    (1)
   23;    I:=I+I
   24;    STW             (store)
   25;    TIR        1    (i)
   26;    TIC        2    (+3)
   27;    B:=I<I
   28;    WHILE
   29;    FSE
   30;    INOUT      20   (body...)
...
   33;    FR
   34;    RETURN
```

OPCODE06 FOR STEP UNTIL

(OPCODE07 for REAL version)

```
“for” i := 3 “step” -1 “until” 1 “do” “print” i;
FOR body block exit
i 3 STEP
1 NEGII STEP2
1 UNTIL
FSE
body: ...
exit: ...

   14;    FOR
   15;    0          27;  (body)
   16;    +832            (block)
   17;    0          31;  (exit)
   18;    TIA        1    (i)
   19;    TIC        2    (3)
   20;    STEP            (first step)
   21;    TIC        1    (1)
   22;    NEGI            (negate)
   23;    STEP2           (next step)
```

```
   24;    TIC       1    (1)
   25;    UNTIL
   26;    FSE
   27;    INOUT     20   (body)
...
   30;    FR             (FOR RETURN)
```

OPCODE08 Goto

```
“begin” “integer” i;
    “for” i := 1, 2 “do” “begin”
          “if” i = 1 “then” “goto” l1 else “goto” l2;
          stop;
     l1:  “print” {L1}
     end;
     stop;
l2:  “print” {L2}
“end”
```

```
   11;    CBL
   12;    UJ        47;
   13;    PE        816
   14;    FOR
   15;    24;            (exit)
   16;    832            (block)
   17;    39;            (body)
   18;    TIA       1    i
   19;    TIC       1    (+1)
   20;    DO
   21;    TIC       3    (+2)
   22;    DO
   23;    FSE
   24;    TIR       1    (i)
   25;    TIC       1    (+1)
   26;    B:=I=I
   27;    IFJ       30;
   28;    GT        4    (L1)
   29;    UJ        31;
   30;    GT        6    (L2)
   31;    FINISH
   32;    INOUT     20
   33;    UJ        36;
   34;    £{L1
   35;    £}
   36;    TA   34;
   37;    INOUT     15
   38;    FR
   39;    FINISH
   40;    INOUT     20
   41;    UJ        44;
   42;    £{L1
```

```
   43;     £}
   44;     TA   42;
   45;     INOUT     15
   46;     RETURN
   47;     FINISH
QACODL=48;
   48;     +0
   49;     +1
   50;     +3
   51;     +2
   52;     0 32;          (L1)
   53;     +832
   54;     0 40;          (L2)
   55;     +816
```

OPCODE09 Switch

```
“switch” sw := l1, l2;
“goto” sw[1];
   14;     TIC       +1
   15;     GTS       3
l1: “print” {L1};
   16;     INOUT     20
   17;     UJ        20;
   18;     £{L1
   19;     £}
   20;     TA        18;
   21;     INOUT     15
stop;
   22;     FINISH
l2:
   23;     INOUT     20
   24;     UJ        27;
   25;     £{L2
   26;     £}
   27;     TA   25;
   28;     INOUT     15
   29;     RETURN
   30;     FINISH
QACODL=31;
   31;     +0
   32;     +1
   33;     +3
   34;     +2             (sw)
   35;     0 16;          (l1)
   36;     +816
   37;     0 23;          (l2)
   38;     +816
```

OPCODE10 Arrays

```
"integer" "array" m, n[1:2,3:4];
   14;    TIC       +1
   15;    TIC       +2
   16;    TIC       +3
   17;    TIC       +4
   18;    +130           (64 * 2 + 2)
   19;    TA        m
"real" "array" r[0:9];
   20;    TIC       +0
   21;    TIC       +9
   22;    MAMPS     +65  (64 * 1 + 1)
   23;    /0 3
m[2,4] := 99;
   24;    TIA       m
   25;    TIC       +2
   26;    TIC       +4
   27;    INDA      6    (2 indexes = 6 words)
   28;    TIC       +99
   29;    ST
r[7] := 66.0;
   30;    TIA       r
   31;    TIC       +7
   32;    INDA      3    (1 index = 3 words)
   33;    TRC       66.0
   34;    ST
“print” r[7];
   35;    INOUT     20
   36;    TIA       r
   37;    TIC       +7
   38;    INDR      3    (1 index)
   39;    INOUT     4
“print” m[2,4];
   40;    INOUT     20
   41;    TIA       m
   42;    TIC       +2
   43;    TIC       +4
   44;    INDR      6    (2 indexes)
   45;    INOUT     3
QACODL=48;
   48;    +0
   49;    +1
   50;    +3
   51;    +2
   52;    +4
   53;    +9
   54;    +99
   55;    +66.0
   56;    +66.0
   57;    +66.0
```

OPCODE11 Check functions

(OPCODE12 = OPCODE11 with check functions compiled out);

```
checks({string})

    15;     UJ          19;
    16;     £{ST
    17;     £RIN
    18;     £G}
    19;     TA          16;
    20;     CON+10
    21;     CHECKS
b := checkb("true")
    22;     TIA         b
    23;     UJ          24;
    24;     TIC         "true"
    25;     CHECKB
    26;     ST
    27;     TIA         i
    28;     UJ          29;
    29;     TIC         +1
    30;     CHECKI
    31;     ST
    32;     TRA         r
    33;     UJ          34;
    34;     TRC         0.5
    35;     CHECKR
    36;     ST
```

OPCODE13 Type Conversions

```
"real" r; "integer" i;
r := 1;
    14;     TRA         r
    15;     TIC         +1
    16;     ITOR1
    17;     ST
r := r+1;
    18;     TRA         r
    19;     TRR         r
    20;     TIC         +1
    21;     ITOR1
    22;     R:=R+R
    23;     ST
r := 1 + r;
    24;     TRA         r
    25;     TIC         +1
    26;     TRR         r
    27;     ITOR2
    28;     R:=R+R
```

```
   29;     ST
i := r
   30;     TIA         i
   31;     TRR         r
   32;     RTOI1
   33;     ST
```

OPCODE14 Procedure Call – no arguments

```
“procedure” p; “print” 0;
   14;     UJ          20;
   15;     PE          832   (block 52, args 0)
   16;     INOUT       20
   17;     TIC         0
   18;     INOUT       3
   19;     RETURN
p;
   20;     CF          15;
```

OPCODE15 Procedure with Goto Exit

```
“procedure” p; “goto” l;
   15;     PE          832
   16;     GT          l
   17;     RETURN
   18;     CF          p;
stop;
   19;     FINISH
l: “print” ...
   20;     INOUT       20
...
```

OPCODE16 Procedure Call – Integer Value Parameter

```
“integer” j;
“procedure” p(i); “value” i; “integer” i;
“begin”
   14;     UJ          24;
   15;     PE          833   (block 52, args 1)
   16;     /1 0              (value, integer)
   17;     IFUN        833   (i: block 52, arg 1)
   i := 10;
   18;     TIC         +10
   19;     ST
    “print” i;
   20;     INOUT       20
   21;     TFAI        833
   22;     INOUT       3
“end”;
```

```
   23;    RETURN
j := 20;
   24;    TIA       j
   25;    TIC       +20
   26;    ST
   27;    UJ   28;
   28;    TIR        j
   29;    CF        15;
   30;    INOUT     20
   31;    TIR       j
   32;    INOUT     3
```

OPCODE17 Procedure Call – Real Value Parameter

```
“real” s;
“procedure” p(r); “value” r; “real” r;
“begin”
   14;    UJ        24;
   15;    PE        833  (block 25 args 1)
   16;    /2 0           (value, real)
    r := 10;
   17;    RFUN      833  (r: block 25 arg 1)
   18;    TRC       10.0
   19;    ST
    “print” r
   20;    INOUT     20
   21;    TFAI      833  (r: block 25 arg1)
   22;    INOUT     4
“end”;
   23;    RETURN
s := 20;
   24;    TRA       s
   25;    TRC       +20
   26;    ST
ps(s);
   27;    UJ        28;
   28;    TRR       s
   29;    CF        15;
“print” s;
   30;    INOUT     20
   31;    TRR       s
   32;    INOUT     4
```

OPCODE18 Procedure Call – Label Parameter

```
“procedure” p(l); “label” l;
   14;    UJ        19;
   15;    PE        833  (block 25, 1 arg)
   16;    9 0            (label)
    “goto” l
```

```
   17;    GTF       833  (l: block 25, arg 1)
“end”
   18;    RETURN         (+253969)
   19;    UJ        20;
p(l);
   20;    TICA      3    (l)
   21;    CON+9          (label indicator)
   22;    CF        15;
stop;
   23;    FINISH
l: “print” {l}
   24;    INOUT     20
   25;    UJ        27;
   26;    {L}
   27;    TA        26;
   28;    INOUT     15

QACODL=31;
   31;    +0
   32;    +1
   33;    +3
   34;    24;            (l address)
   35;    +816           (l environment)
```

OPCODE19 Procedure Call – String Argument

```
“procedure” p(s); “string” s;
   14;    UJ        21;
   15;    PE        833  (block 25 args 1)
   16;    10 0           (string)
    “print” s;
   17;    INOUT     20
   18;    TFAI      833  (s: block 25, arg 1)
   19;    INOUT     15
   20;    RETURN
   21;    UJ        22;
   22;    UJ        24;
   23;    £{S}
   24;    TA        23;
   25;    CON+10         (string indicator)
   26;    CF   15;
   27;    RETURN
```

OPCODE20 Procedure Call – Procedure Argument (1)

```
“procedure” pr; “print” {P};
   14;    UJ        22;
   15;    PE        832  (Block 25, 0 args)
   16;    INOUT     20
   17;    UJ        19;
```

```
   18;     £{P}
   19;     TA          18;
   20;     INOUT       15
   21;     RETURN
“procedure” p(pp); “procedure” pp; pp;
   22;     UJ          27;
   23;     PE          849   (block 53, 1 args)
   24;     7 0;              (procedure 0 parameters)
   25;     CFF         849   (block 53, arg 1)
   26;     RETURN
   27;     UJ          28;
   28;     TA          15;   (pr)
   29;     CON+7
   30;     CF          23;   (p)
   31;     RETURN
```

OPCODE21 Procedure Call – Procedure Argument, with value argument

Note use of ((0.5)) to indicate call by value.

```
“procedure” pr(r); “value” r; “real” r; “print” r;
   14;     UJ          21;
   15;     PE          833   (block 25, 1 args)
   16;     /2 0              (value real)
   17;     INOUT       20
   18;     TFAI        833   (block 25, arg 1)
   19;     INOUT       4
   20;     RETURN    14;
“procedure” p(pp); “procedure” pp; pp((0.5));
   21;     UJ          28;
   22;     PE          849   (block 53, 1 args)
   23;     7 1
   24;     UJ          25;
   25;     TRC         0.5
   26;     CFF         849
   27;     RETURN
   28;     UJ          29;
   29;     TA          15;   (pr)
   30;     CON+7
   31;     CF          22;   (p)
   32;     RETURN
```

OPCODE22 Procedure Call – Procedure Argument (3)

Note absence of brackets on 0.5 – generates error 7 (unstandardized real). Same happens in Hunter Algol. 903 Algol gives translation error 108.

```
“procedure” pr(r); “value” r; “real” r; “print” r;
   14;     UJ          21;
```

```
   15;    PE        833
   16;    /2 0
   17;    INOUT     20
   18;    TFAI      833
   19;    INOUT     4
   20;    RETURN
"procedure" p(pp); "procedure" pp; pp(0.5);
   21;    UJ        28;
   22;    PE        849
   23;    /7 1
   24;    UJ        25;
   25;    TRCA      3    (note stacks address not value)
   26;    CFF       849
   27;    RETURN
pp(pr);
   28;    UJ        29;
   29;    TA        15;  (pr)
   30;    CON+7
   31;    CF        22;
   32;    RETURN
```

OPCODE23 Function Call

```
"procedure" pr(r); "value" r; "real" r; pr := r/2.0;
   14;    UJ        23;
   15;    PE        833
   16;    /2 0           (value real)
   17;    RFUN      832  (pr: result address)
   18;    TFAI      833  (r: argument)
   19;    TRC       2.0
   20;    R:=R/R
   21;    ST
   22;    RETURN
"print" pr(1.0)
   23;    INOUT     20
   24;    UP             (create space for result)
   25;    UJ        26;
   26;    TRC       1.0
   27;    CF        15;
   28;    INOUT     4
   29;    RETURN
```

OPCODE24 Procedure Call – Function Parameter

```
"procedure" pr(r); "value" r; "real" r; pr := r/2.0;
   14;    UJ        23;
   15;    PE        833
   16;    /2 0
   17;    RFUN      832
   18;    TFAI      833
```

```
   19;    TRC       2.0
   20;    R:=R/R
   21;    ST
   22;    RETURN
   23;    INOUT     20
   24;    UP
   25;    UJ        26;
   26;    TRC       1.0
   27;    CF        15;
   28;    INOUT     4
   29;    RETURN
"procedure" p(pp);  "real" "procedure" pp;
      "print" pp((1.0));
   23;    UJ        33;
   24;    PE        849
   25;    6 1;
   26;    INOUT     20
   27;    UP             (slot for result)
   28;    UJ        29;
   29;    TRC       1.0
   30;    CFF       849
   31;    INOUT     4
   32;    RETURN
   33;    UJ        34;
   34;    TA        15;  p
   35;    CON+6
   36;    CF        24;
   37;    RETURN
```

OPCODE25 Procedure Call – Code Function Parameter

```
"procedure" p(pp);  "real" "procedure" pp;
    "print" pp((1.0));
   14;    UJ        24;
   15;    PE        833
   16;    6 1
   17;    INOUT     20
   18;    UP
   19;    UJ        20;
   20;    TRC       1.0
   21;    CFF       833
   22;    INOUT     4
   23;    RETURN
“print” sqrt(2.0)
   24;    INOUT     20
   25;    UP
   26;    UJ        27;
“print” sqrt(2.0);
   27;    TRC       2.0
   28;    CF        SQRT+0    (gets modified to /14)
   29;    INOUT     4
```

```
   30;    UJ         31;
   31;    TA         SQRT+0    (gets modified to 15)
   32;    CON+6
   33;    CF         p
   34;    RETURN
```

OPCODE26 Procedure Call – Switch Parameter

```
“switch” sw := l, m;
"procedure" p(s); "switch" s;
    "goto" s[2];
   14;    UJ         20;
   15;    PE         833
   16;    8 0
   17;    TIC        +2
   18;    GTFS       833
   19;    RETURN
p(sw)
   20;    UJ   21;
   21;    TICA       3
   22;    CON8
   23;    CF         15;
   24;    FINISH
m: “print” ...
   25;    INOUT      20
```

OPCODE27 Procedure Call – Value Array Argument

```
“integer” “array” m[1:10];
   14;    TIC        +1
   15;    TIC        +10
   16;    MAMPS      65
   17;    +1
“procedure” p(n, i); “value” n, i;
    “integer” “array” n; “integer” i;
   18;    UJ         33
   19;    PE         834
   20;    /3 1
   21;    /1 0
    n[i] := i;
   22;    TFAI       833  (n)
   23;    TFAI       834  (i)
   24;    INDA       3
   25;    TFAI       834  (i)
   26;    ST
    “print” n[i];
   27;    INOUT      20
   28;    TFAI       833  (n)
   29;    TFAI       834  (i)
   30;    INDR       3
```

```
   31;    INOUT     3    (+122883)
   32;    RETURN
m[1] := 10
   33;    TIA       m
   34;    TIC       +1
   35;    INDA      3
   36;    TIC       +10
   37;    ST
p(m,1);
   38;    UJ        39;
   39;    TIA       m
   40;    UJ        41;
   41;    TIC       +1
   42;    CF        19;
   43;    INOUT     20
“print” m[1]
   44;    TIA       m
...
```

OPCODE28 Procedure Call – Formal Value Label Argument

```
"procedure" a(l); "value" l; "label" l; “goto" l;
   14;    UJ        19;
   15;    PE        833
   16;    /9 0
   17;    GTF       833
   18;    RETURN
"procedure" b(l); "value" l; "label" l; a(l);
   19;    UJ        26;
   20;    PE        849
   21;    /9 0
   22;    UJ        23;
   23;    TFAI      849  (stack arg – copy of
                          address of m)
   24;    CF        15;
   25;    RETURN
b(m);
   26;    UJ        27;
   27;    TICA      m    (address of m in QAVNDA)
   28;    CF        20;
stop;
   29;    FINISH
m: “print” {M};
   30;    INOUT     20
   31;    UJ        33;
   32;    £{M}
   33;    TA        32;
   34;    INOUT     15
```

OPCODE29 Procedure Call – Switch Index as Label Actual Parameter

```
"switch" s := m;
```

```
"procedure" p(l); "label" l; "goto" l;
14;  UJ        19;
   15;    PE        833
   16;    /9 0
   17;    GTF       833
   18;    RETURN
p(s[1]);
   19;    TIC       1
   20;    INDS      3
   21;    CF        15;
```

OPCODE30 Procedure Call – Switch Index to Formal Name Label

```
"switch" s := m;
"procedure" p(l); "label" l; "goto" l;
p(s[1]);
   15;    PE        833
   16;    9 0
   17;    GTF       833
   18;    RETURN
   19;    UJ        20;  (thunk – becomes UJ 27;)
   20;    TIC       +1
   21;    INDS      3
   22;    ST
   23;    RETURN
   24;    PE        16
   25;    IFUN      16
   26;    TA        19;  (thunk – becomes UJ 20;)
   27;    MKTHK     9
Halt 15
   28;    CF        p
```

OPCODE31 Procedure Call – Switch Formal Parameter as Actual Parameter

```
"switch" ss := m;
"procedure" p1(s); "switch" s; "goto" s[1];
14;  UJ        20;
   15;    PE        833
   16;    8 0
   17;    TIC       1
   18;    GTFS      833
   19;    RETURN
"procedure" p2(s); "switch" s; p1(s);
   20;    UJ        28;
   21;    PE        849
   22;    8 0
   23;    UJ        24;
```

```
   24;    TFAI      849
   25;    CON8
   26;    CF        p1
   27;    RETURN
    p2(ss);
   28;    UJ        29;
   29;    TICA      ss
   30;    CON8
   31;    CF        p2
```

OPCODE32 Switch to Switch Formal Parameter

```
“procedure” p1(s); “switch” s; “goto” s[1];
   15;    PE        833
   16;    8 0
   17;    TIC       +1
   18;    GTFS      833
   19;    RETURN
   20;    UJ        28;
“procedure” p2(s); “switch” s; p1(s);
   21;    PE        849
   22;    8 0
   23;    UJ        24;
   24;    TFAI      849
   25;    CON8
   26;    CF        15;
   27;    RETURN
   28;    UJ        29;
p2(s)
   29;    TICA      s
   30;    CON8
   31;    CF        p2
```

OPCODE33 Jump out of Recursive Procedure and Loop

```
"procedure" a(i); “value” i; "integer" i;
   "begin"
        "integer" k;
        "procedure" b(j); "value" j; "integer" j;
        "begin" "print" {J=}, sameline, j, {{L}};
           "if" j = 0 "then"
              "goto" 1
          "else"
              b(j-1);
        "end";
        "print" {i=}, sameline, i, {{L}};
        "if" i < 1 "then"
            a(i+1)
        "else"
```

```
            "for" k := i "step" 1 "until" 10 "do"
             "if" k = 5 "then"
                b(10);
             stop;
             l: "print" {DONE AT I=}, sameline, i;
    "end";
    a(-10)
```

OPCODE34 Procedure Call – Integer Argument By Name

```
"procedure" p(i); "integer" i; i := i + 10;
   14;    UJ         23;
   15;    PE         833
   16;    1 0
   17;    GETAD      833
   18;    TRCN       833
   19;    TIC        3
   20;    I:=I+I
   21;    ST
   22;    RETURN
"integer" j;
j := 20;
   23;    TIA        j
   24;    TIC        20
   25;    ST
   26;    UJ         27;
p(j);
   27;    TIA        j
   28;    CF         15;
```

OPCODE35 Procedure Call – Real Argument by Name

```
“procedure” p(r); “real” r; r:=r+10;
   14;    UJ         23;
   15;    PE         833
   16;    2 0
   17;    GETAD           833
   18;    TRCN            833
   19;    TRC             +10
   20;    R:=R+R
   21;    ST
   22;    RETURN
```

OPCODE36 Procedure Call – Real Array Argument by Name

```
“procedure” p(r); “real” “array” r; r[1] := r[1]+10.0;
   14;    UJ         271
   15;    PE         833
```

```
   16;    4 1
   17;    TFAI       833  (r)
   18;    TIC        1
   19;    INDA       3
   20;    TFAI       833  (r)
   21;    TIC        +1
   22;    INDR       3
   23;    TRC        +10
   24;    R:=R+R
   25;    ST
   26;    RETURN
"real" "array" s[1:10];
   27;    TIC        +1
   28;    TIC        +10
   29;    MAMPS      65
   30;    /0 1
s[1] := 20.0;
   31;    TIA        s
   32;    TIC        +1
   33;    INDA       3
   34;    TRC        +20
   35;    ST
   36;    UJ         37;
p(s);
   37;    TIA        s
   38;    CON4
   39;    CF         15;
```

OPCODE37 Procedure Call – Real Constant Actual Argument for Formal Name Parameter

```
"procedure" p(r); "real" r; "print" r;
   14;    UJ         21;
   15;    PE         833
   16;    2 0
   17;    INOUT      20
   18;    TRCN       833  (r)
   19;    INOUT      4
   20;    RETURN
p(0.5);
   21;    UJ         22;
   22;    TRCA       0.5
   23;    CF         15;
```

OPCODE38 Procedure Call – Formal Name Argument to Formal Name Parameter

```
   14;    UJ         21
"procedure" p1(r); "real" r; "print" r;
```

```
   15;    PE        833
   16;    2 0
   17;    INOUT     20
   18;    TRCN      833  (r)
   19;    INOUT     4
   20;    RETURN
   21;    UJ        28
“procedure p2(r); “real” r; p1(r);
   22;    PE        849
   23;    2 0
   24;    UJ        25;
   25;    TFAI      849  (r)
   26;    CF        p1
   27;    RETURN
p2(0.5)
   28;    UJ        29;
   29;    TRCA      0.5
   30;    CF        p2
```

OPCODE39 Procedure Call – Actual Name Argument to Formal Value Parameter

```
“procedure” p1(r); “value” r; “real” r; “print” r;
   15;    PE        833
   16;    /2 0
   17;    INOUT     20
   18;    TFAI      833  (r)
   19;    INOUT     4
   20;    RETURN
   21;    UJ        28;
“procedure” p2(r); “real” r; p1(r);
   22;    PE        849
   23;    2 0
   24;    UJ        25;
   25;    TRCN      849  (r)
   26;    CF        p1;
   27;    RETURN
   28;    UJ        29;
p2(0.5)
   29;    TRCA      0.5
   30;    CF        22;
```

OPCODE40 Procedure Call – Actual Value Argument to Formal Value Parameter

```
“procedure” p1(r); “value” r; “real” r; “print” r;
   15;    PE        833
   16;    /2 0
   17;    INOUT     20
   18;    TFAI      833  (r)
```

```
   19;    INOUT     4
   20;    RETURN
   21;    UJ        28;
“procedure” p2(r); “value” r; “real” r; p1(r);
   22;    PE        849
   23;    /2 0
   24;    UJ        25;
   25;    TFAI      849  (r)
   26;    CF        15;
   27;    RETURN
   28;    UJ   29;
p1(0.5);
   29;    TRC       0.5
   30;    CF        22;
```

OPCODE41 Procedure Call – Value Actual Argument to Formal Name Parameter

```
“procedure” p1(r); “real” r; “print” r;
   15;    PE        833
   16;    2 0
   17;    INOUT     20
   18;    TRCN      833  (r)
   19;    INOUT     4
   20;    RETURN
   21;    UJ        28;
“procedure” p2(r); “value” r; “real” r; p1(r);
   22;    PE        849
   23;    /2 0
   24;    UJ        25;
   25;    RFUN      849  (r)
   26;    CF        p1
   27;    RETURN
   28;    UJ   29;
p1(0.5)
   29;    TRC       0.5
   30;    CF        p2;
```

OPCODE42 Procedure Call – Array Value Actual Argument to Array Value Formal Parameter

```
“procedure” p1(r); “value” r; “array” r;
“begin”
   19;    PE        833
   20;    /2 1
    r[1] := r[1] + 1.0;
   21;    TFAI      833
   22;    TIC       +1
   23;    INDA      3
   24;    TFAI      833
```

```
   25;     TIC         +1
   26;     INDR        3
   27;     TRC         1.0
   28;     R:=R+R
   29;     ST
...
“end”;
   35;     RETURN
   36;     UJ          43;
“procedure” p2(r); “value” r; “array” r;
   37;     PE          849
   38;     /2 8191          (note bounds not known)
   39;     UJ          40;
    p1(r);
   40;     TFAI        849
   41;     CF          p1;
   42;     RETURN
   ...
p2(s)
   49;     TIA         s
   50;     CF          p2;
```

OPCODE43 Procedure Call – Array Name Actual Argument to Array Value Formal Parameter

```
“procedure” p1(r); “value” r; “array” r;
“begin”
   19;     PE          833
   20;     /2 1
    r[1] := r[1] + 1;
   21;     TFAI        833
   22;     TIC         +1
   23;     INDA        3
   24;     TFAI        833
   25;     TIC         +1
   26;     INDR        3
   27;     TRC         +1.0
   28;     R:=R+R
   29;     ST
...
“end”;
   35;     RETURN
   36;     UJ          43;
“procedure” p2(r); “array” r; p1(r);
   37;     PE          849
   38;     2 8191
   39;     UJ          40;
   40;     TFAI        849
   41;     CF          p1;
   42;     RETURN
...
```

```
p2(s);
   49;     TIA         s
   50;     CON4
   51;     CF          p2
```

OPCODE44 Procedure Call – Array Value Actual Argument to Array Name Formal Parameter

```
“procedure” p1(r); “array” r;
“begin”
   19;     PE          833
   20;     2 1
    r[1] := r[1] + 1;
   21;     TFAI        833
   22;     TIC         +1
   23;     INDA        3
   24;     TFAI        833
   25;     TIC         1
   26;     INDR        3
   27;     TRC         +1.0
   28;     R:=R+R
   29;     ST
...
“end”;
   35;     RETURN
   36;     UJ          44;
“procedure” p1(r); “value” r; “real” r; p1(r);
   37;     PE          849
   38;     /2 8191
   39;     UJ          40;
   40;     TFAI        849
   41;     CON4
   42;     CF          p1
   43;     RETURN
...
p1(s);
   50;     TIA         s
   51;     CF          p2;
```

OPCODE45 Procedure Call – Array Name Actual Argument to Array Name Formal Parameter

```
“procedure” p1(r); “array” r;
“begin”
   19;     PE          833
   20;     2 1
    R[1] := R[1] + 1.0;
   21;     TFAI        833
   22;     TIC         +1
   23;     INDA        3
```

```
   24;    TFAI       833
   25;    TIC        +1
   26;    INDR       3
   27;    TRC        +1.0
   28;    R:=R+R
   29;    ST
...
"end";
   35;    RETURN
   36;    UJ         44;
"procedure" p2(r); "array" r; p1(r);
   37;    PE         849
   38;    2 8191
   39;    UJ         40;
   40;    TFAI       849
   41;    CON4
   42;    CF         p1
   43;    RETURN
...
p2(s)
   50;    TIA        s
   51;    CON4
   52;    CF         p2
```

OPCODE46 Procedure Call – String Actual Argument to String Formal Parameter

```
"procedure" p1(s); "string" s; "print" s;
   15;    PE         833
   16;    10 0
   17;    INOUT      20
   18;    TFAI       833
   19;    INOUT      15
   20;    RETURN
   21;    UJ         29
"procedure" p2(s); "string" s; p1(s)
   22;    PE         849
   23;    10 0
   24;    UJ         25;
   25;    TFAI       849
   26;    CON10
   27;    CF         p1
   28;    RETURN
   29;    UJ         30;
   30;    UJ         32;
   31;    £{S}
   32;    TA         31;
   33;    CON10
   34;    CF         p1({S})
```

OPCODE47 Procedure Call – Name Label Actual Argument to Name Label Formal Parameter

```
“procedure” p1(l); “label” l; “goto” l;
   15;    PE        833
   16;    9 0
   17;    GTF       833
   18;    RETURN
   19;    UJ        27;
“procedure” p2(l) “label” l; p1(l);
   20;    PE        849
   21;    9 0
   22;    UJ        23;
   23;    TFAI      849
   24;    CON9
   25;    CF        p1
   26;    RETURN
   27;    UJ   28;
p1(l)
   28;    TICA      l
   29;    CON9
   30;    CF        p1
```

OPCODE48 Procedure Call – Value Label Actual Argument to Name Label Formal Parameter

```
“procedure” p1(l); “label” l; “goto” l;
   15;    PE        833
   16;    9 0
   17;    GTF       833
   18;    RETURN
   19;    UJ        27;
“procedure” p2(l) “value” l; “label” l; p1(l);
   20;    PE        849
   21;    /9 0
   22;    UJ        23;
   23;    TFAI      849
   24;    CON9
   25;    CF        p1
   26;    RETURN
   27;    UJ        28;
p2(l);
   28;    TICA      l
   29;    CF        p2;
```

OPCODE49 Procedure Call – Name Label Actual Argument to Value Label Formal Parameter

```
“procedure” p1(l); “value” l; “label” l; “goto” l;
```

```
   15;     PE          833
   16;     /9 0
   17;     GTF         833
   18;     RETURN
   19;     UJ          26;
“procedure” p2(l); “label” l; p1(l);
   20;     PE          849
   21;     9 0
   22;     UJ          23;
   23;     TFAI        849
   24;     CF          p1
   25;     RETURN
   26;     UJ          27;
p2(l);
   27;     TICA l
   28;     CON9
   29;     CF          p2
```

OPCODE50 Procedure Call – Value Label Actual Argument to Value Label Formal Parameter

```
“procedure” p1(l); “label” l; “goto” l;
   15;     PE          833
   16;     /9 0
   17;     GTF         833
   18;     RETURN
   19;     UJ          26
 “procedure” p2(l); “label” l; p1(l);
   20;     PE          849
   21;     /9 0
   22;     UJ          23;
   23;     TFAI        849
   24;     CF          p1
   25;     RETURN
   26;     UJ          27;
p2(l)
   27;     TICA        l
   28;     CF          p2
```

OPCODE51: Procedure Call – Switch Actual Argument to Switch Formal Parameter

```
“procedure” p1(s); “switch” s; goto s[1];
   15;     PE          833
   16;     8 0
   17;     TIC         +1
   18;     GTFS        833
   19;     RETURN
   20;     UJ          28;
“procedure” p2(s); “switch” s; p1(s);
```

```
   21;     PE          849
   22;     8 0
   23;     UJ          24;
   24;     TFAI        849
   25;     CON8
   26;     CF          p1
   27;     RETURN
   28;     UJ          29;
p2(s);
   29;     TICA        s
   30;     CON8
   31;     CF          p2
```

OPCODE52: Procedure Call – Procedure Actual Argument to Procedure Formal Parameter

```
"procedure" p; "print" {P};
   15;     PE          832
   16;     INOUT       20
   17;     UJ          19;
   18;     £{P}
   19;     TA          18;
   20;     INOUT       15
   21;     RETURN
   22;     UJ          27;
"procedure" p1(p); "procedure" p; p;
   23;     PE          849
   24;     7 0
   25;     CFF         849  (p)
   26;     RETURN
   27;     UJ          35;
"procedure" p2(p); "procedure" p; p1(p);
    p2(p)
   28;     PE          865
   29;     7 8191           (parameters not known)
   30;     UJ          31;
   31;     TFAI        865
   32;     CON7
   33;     CF          p1;
   34;     RETURN
   35;     UJ          36;
   36;     TA          p
   37;     CON7
   38;     CF          p1
```

OPCODE53: Procedure Call – Function Actual Argument to Function Formal Parameter

```
"integer" "procedure" p1; p1 := 1;"procedure" p2(p);
   15;     PE          832
```

```
   16;    IFUN      832
   17;    TIC       +1
   18;    ST
   19;    RETURN
   20;    UJ        28;
"procedure" p; p1(p);
21;  PE        849
   22;    5 0
   23;    IFUN      848
   24;    UP
   25;    CFF       849
   26;    ST
   27;    RETURN
“print” p2(p);
   28;    INOUT     20
   29;    UP
   30;    UJ        31;
   31;    TA        15;
   32;    CON5
   33;    CF   21;
```

OPCODE54 Procedure Call – Scalar Call by Name with Expressions as Actual Arguments

```
"array" a[1:1];
"real" r;
      14; TIC       +1
   15;    TIC       +1
   16;    MAMPS     65
   17;    &100001
   18;    UJ        25
"procedure" p1(r); "real" r; "print" r;
   19;    PE        833
   20;    2 0
   21;    INOUT     20
   22;    TRCN      833
   23;    INOUT     4
   24;    RETURN
   25;    UJ        31
"real" "procedure" p2; p2 := 1.0;
   26;    PE        848
   27;    RFUN      848
   28;    TRC       +1.0
   29;    ST
   30;    RETURN
a[1] := r := 2.0;
   31;    TIA       a
   32;    TIC       +1
   33;    INDA      3
   34;    TRA       r
   35;    TRC       2.0
```

```
   36;    STA
   37;    ST
p1(p2);
   38;    UJ        39;
   39;    TA        p2
   40;    CON6
   41;    CF        p1;
p1(a[1]);
   42;    UJ        43;  (this will become UJ 51)
   43;    TIA       a
   44;    TIC       +1
   45;    INDA      3    (this will become INDR)
   46;    ST
   47;    RETURN
   48;    PE        16
   49;    RFUN      16   (store result in arg0)
   50;    TA        42;  (this will become UJ 43;)
   51;    MKTHK     2
Halt 15
   52;    CF        p1
p1(r+a[1]);
   53;    UJ        54;  (this will become UJ 64;)
   54;    TRR       r
   55;    TIA       a
   56;    TIC       +1
   57;    INDR      3
   58;    R:=R+R
   59;    ST
   60;    RETURN
   61;    PE        16
   62;    RFUN      16
   63;    TA        53; (this will become UJ 54;)
   64;    MKTHK     2
Halt 15
   65;    CF        p1
p1(1.0);
   66;    UJ        67;
   67;    TRCA      1.0
   68;    CF        19;
p1(1.0+2.0)
   69;    UJ        70;  (becomes UJ 78;)
   70;    TRC       1.0
   71;    TRC       2.0
   72;    R:=R+R
   73;    ST
   74;    RETURN
   75;    PE        16
   76;    RFUN      16
   77;    TA        69;
   78;    MKTHK     2
Halt 15
   79;    CF   19;
```

OPCODE55: Procedure Call – Scalar Call by Name with Expressions as Actual Arguments

As OPCODE54 but with integers. Note `MKTHK 1` rather than `MKTHK 2`.

```
“integer” “array” a[1:1];
“integer” i;
   14;    TIC       +1
   15;    TIC       +1
   16;    MAMPS     65
   17;    +1
   18;    UJ        25;
“procedure” p1(i); “integer” i; “print” i;
   19;    PE        833
   20;    1 0
   21;    INOUT     20
   22;    TRCN      833
   23;    INOUT     3
   24;    RETURN
   25;    UJ        31;
“integer” procedure” p2(i); “integer” i; i := 1;
   26;    PE        848
   27;    IFUN      848  (i)
   28;    TIC       +1
   29;    ST
   30;    RETURN
a[1,1] := i := 2;
   31;    TIA       +1
   32;    TIC       +1
   33;    INDA      3
   34;    TIA       i
   35;    TIC       +1
   36;    STA
   37;    ST
   38;    UJ        39;
p1(p2);
   39;    TA        p2
   40;    CON5
   41;    CF        p1
p1(a[1]);
   42;    UJ        43;  (becomes UJ 51;)
   43;    TIA       a
   44;    TIC       +1
   45;    INDA      3
   46;    ST
   47;    RETURN
   48;    PE        16
   49;    IFUN      16
   50;    TA        42;  (becomes UJ 43;)
```

```
   51;    MKTHK     1
Halt 15
   52;    CF        p1
p1(i+a[1]);
   53;    UJ        54;
   54;    TIR       i
   55;    TIA       a
   56;    TIC       +1
   57;    INDR      3
   58;    I:=I+I
   59;    ST
   60;    RETURN
   61;    PE        16
   62;    IFUN      16
   63;    TA        53;  (becomes UJ 54;)
   64;    MKTHK     1
Halt 15
   65;    CF        p1
p1(1);
   66;    UJ        67;
   67;    TICA      +1
   68;    CF        19;
p1(1+2);
   69;    UJ        70;  (becomes UJ 78;)
   70;    TIC       +1
   71;    TIC       +2
   72;    I:=I+I
   73;    ST
   74;    RETURN
   75;    PE        16
   76;    IFUN      16
   77;    TA        69;  (becomes UJ 70;)
   78;    MKTHK     1
Halt 15
   79;    CF        19;
```

OPCODE57: Procedure Call – Scalar Call by Name with Expressions Actual Arguments, used as LValues

```
“procedure” p1(i); “integer” i; i := 99
   19;    PE        833
   20;    1 0
   21;    GETAD     833
   22;    TIC       +99
   23;    ST
   24;    RETURN
“integer” “procedure” p2; p2 := 1;
   25;    UJ        31;
   26;    PE        848
   27;    IFUN      848
```

```
   28;     TIC        +1
   29;     ST
   30;     RETURN
a[1] := i := 2;
   31;     TIA        a
   32;     TIC        +1
   33;     INDA       3
   34;     1 3
   35;     TIC        +2
   36;     STA
   37;     ST
p1(a[1]);
   38;     UJ         39;  (becomes UJ 47)
   39;     TIA        a
   40;     TIC        +1
   41;     INDA       3
   42;     ST
   43;     RETURN
   44;     PE         16
   45;     IFUN       16
   46;     TA         38;  (becomes UJ 39;)
   47;     MKTHK      1
Halt 15
   48;     CF         19;
```

OPCODE58: Reference to Outer Formal Parameter in Nested Procedures

```
“procedure” p1 (i); “integer” i;
“begin”
   15;     PE         833
   16;     1 0
   17;                23;
    “procedure” p2; “print” i;
   18;     PE         864
   19;     INOUT      20
   20;     TRCN       833  (arg 1 of block 832)
   21;     INOUT      3
   22;     RETURN
   23;     CF         p2
   24;     RETURN
   25;     UJ         26;
   26;     TICA       +99
   27;     CF         15;
```

OPCODE59: Recursive Call by Name with Expressions as Actual Parameters

```
“procedure” p1 (i); “integer” i;
“begin”
```

```
   15;    PE         833
   16;    1 0
    “print” i;
   17;    INOUT      20
   18;    TRCN       833  (i)
   19;    INOUT      3
    “if” i > 1 “then”
   20;    TRCN       833  (i)
   21;    TIC        +1
   22;    B:=I>I
   23;    IFJ        35;
        p1(i-1);
   24;    UJ         25;  (thunk – becomes UJ 33;)
   25;    TRCN       833  (i)
   26;    TIC        +1
   27;    I:=I-I
   28;    ST
   29;    RETURN
   30;    PE         16
   31;    IFUN       16
   32;    TA         24;  (thunk – becomes UJ 25;)
   33;    MKTHK      1
Halt 15
   34;    CF         p1
    “end”;
   35;    RETURN          (-8175)
p2(5);
   36;    UJ   37;
   37;    TICA       +5
   38;    CF         p2
```

OPCODE60: Recursive, Nested Procedures

This is OPCODE59 with the automatically generated thunk written out by hand.

```
“procedure” p1(i); “integer” i;
“begin”
   15;    PE         833
   16;    1 0
   17;    UJ         25;
    “integer” “procedure” p2;
   18;    PE         864
        p2 := i – 1;
   19;    IFUN       864
   20;    TRCN       833  (i)
   21;    TIC        +1
   22;    I:=I-I
   23;    ST
   24;    RETURN
    “print” i;
   25;    INOUT      20
```

```
   26;     TRCN       833   (i)
   27;     INOUT      3
    “if” i > 1 “then”
   28;     TRCN       833   (i)
   29;     TIC        +1
   30;     B:=I>I
   31;     IFJ        36;
   32;     UJ         33;
         p1(p2);
   33;     TA         p2
   34;     CON5
   35;     CF         p2
   36;     RETURN
“end”;
   37;     UJ    38;
p2(5);
   38;     TICA       +5
   39;     CF         p2
```

OPCODE61: Jump to Inner label

Correctly produces runtime error 24.

OPCODE62: Recursive Procedure with FOR Loops

(To stress `FINDFP`).

```
“procedure” p(i); “integer” i;
“begin”
   15;     PE         833
   16;     1 0
    “print” i;
   17;     INOUT      20
   18;     TRCN       833   (i)
   19;     INOUT      3
    “if” i > 1 “then”
   20;     TRCN       833   (i)
   21;     TIC        +1
   22;     B:=I>I
   23;     IFJ        53;
   24;     FOR
   25;     0 32;
   26;     +864
   27;     0 53;
   28;     TIA        j
   29;     TIC        +1
   30;     DO
   31;     FSE
   32;     FOR
   33;     0 40;
```

```
   34;     +880
   35;     0 52;
   36;     TIA         j
   37;     TIC         +1
   38;     DO
   39;     FSE
   40;     UJ          41;   (thunk)
   41;     TRCN        833   (i)
   42;     TIC         +1
   43;     I:=I-I
   44;     ST
   45;     RETURN
   46;     PE          16
   47;     IFUN        16
   48;     TA          40;   (end of thunk)
   49;     MKTHK       1
Halt 15
   50;     CF          p
   51;     FR
   52;     FR
   53;     RETURN
   54;     UJ          55;
   55;     TICA        +5
   56;     CF          p
```

OPCODE63 Nested Recursive Procedure Calls with Procedure Parameter, Call by Value

A real workout for the `FINDFP` (find formal parameter) subroutine in the interpreter.

OPCODE64 Labels inside FOR Loops

Validates jump to a label in a loop body from inside the loop.

```
  "for" i := 1, 2 "do" "begin"
    "if" i = 1 "then" "goto" one "else" "goto" two;
    one: "print" {One{L}}
  "end";
  stop;
two: "print" {Two{L}}
```

```
“for” i := 1, 2 “do” “begin”
   14;      FOR
   15;      0 24;
   16;      +832
   17;      0 39;
   18;      TIA         1
   19;      TIC         +1
   20;      DO
   21;      TIC         +2
   22;      DO
```

```
   23;      FSE
    “if” i=1
   24;      TIR          i
   25;      TIC          +1
   26;      B:=I=I
   27;      IFJ          30;
        “else” “goto” two
   28;      GT           4
   29;      UJ           31;
    “then” “goto” one
   30;      GT           6
one:
   31;      INOUT        20
   32;      UJ           36;
   33;      £{On
   34;      £e{{
   35;      £L}}
   36;      TA           33;
   37;      INOUT        15
   38;      FR
   39;      FINISH
two:
   40;      INOUT        20
   41;      UJ           45;
   42;      £{Tw
   43;      £o{{
   44;      £L}}
   45;      TA           42;
   46;      INOUT        15

QACODL=49;
   53;      0 31;        (one)
   54;      +832
   55;      0 40;        (two)
   56;      +816
```

OPCODE65 IO Functions

Appendix: Character Code Table

This is a generic description of the character code tables in the SIR assembler and Algol translator and interpreter.

<table>
<tr><td></td><td>Group indicator</td><td></td><td>Parity Bit</td><td>Action</td><td>Code</td></tr>
<tr><td></td><td>18-10</td><td>9</td><td>8</td><td>7</td><td>6-1</td></tr>
<tr><td>0-63</td><td colspan="2">&40=separators,
&20=digits,
&10=letters,
&04=others
(and in 920 code: &05 for double characters such as $</td><td rowspan="2"></td><td rowspan="2">Action = 0: converts 7 bit external to internal code.

Action = 1: external needs special action</td><td rowspan="2">SIR internal code, or

0: illegal character

1: newline

2: !

3: ignore

5: halt code</td></tr>
<tr><td>64-127</td><td>Convert internal code to telecode with parity</td><td>Unused</td></tr>
</table>