Translated Procedures and Qcodes

Table of Contents

1. Variables

There a six main types of variable: Integer, Float, and String, and arrays of Integers, Floats, or Strings. Sometimes the type of a variable needs to be denoted in QCode, and this is done by a single byte in the range 00 to 05.

Format of variables

Integer Variable

Format: HH LL
16 bit integer, first the high byte then the low byte.
Size: 2
Type Byte: 00

Example: 13579
35 0B

Float Variable

Format: MM MM MM MM MM MM EE SS
6-byte mantissa (least to most significant), Exponent byte, Sign byte
Size: 8

Type Byte: 01

Mantissa is binary coded decimal, most significant digit in high nibble.
Exponent is in range -99 to 99, with 0 meaning 1 digit before decimal point.
Sign byte: sign in bit 7, rest clear, so $80 if negative, $00 if positive.

Example: -123.456789
00 90 78 56 34 12 02 80

Float constants inside a procedure are stored in a shortened form:
LL [MM] EE
Length and sign byte LL, a number of mantissa bytes, an exponent byte.

The missing least significant mantissa bytes are considered to be 0.
The sign bit is stored in the high bit of LL.

Example: -123.4
83 34 12 02

String Variable

Format: MM; LL [characters]
Declared maximum length, Actual length, ASCII contents
Size: MM+2
Type Byte: 02

MM is the maximum length of string, i.e. the number of bytes reserved for contents as stated in the LOCAL or GLOBAL declaration.
LL is actual length of string.
In the MM character bytes that follow, the first LL are actual contents, the rest are indeterminate.

Note that the address of the string variable is that of the length byte.

Example: LOCAL A$(10) : A$ = "ABCD"
0A; 04 41 42 43 44 ?? ?? ?? ?? ?? ??

String constants inside a procedure are stored simply as a length byte followed by the contents.

Example: "ABCD"
04 41 42 43 44

Integer Array Variable

Format: LL LL [integers]
Array size, list of integers
Size: LLLL*2+2
Type Byte: 03

LLLL is a word (16-bit integer) containing the number of integers in the array.
This is followed by the array contents, two bytes for each integer.

In OPL, ADDR(array%()) returns address of first integer in the array. In QCode, the array variable's address is the address of the array size word.

Example: LOCAL A%(2) : A%(1)=123 : A%(2)=-1234
00 02, 00 7B, FB 2E

Float Array Variable

Format: LL LL [floats]
Array size, list of floats
Size: LLLL*8+2
Type Byte: 04

LLLL is the number of integers in the array.
This is followed by the array contents, eight bytes for each float.

In OPL, ADDR() returns address of first float. In QCode, the array address is the address of the array size word.

Example: LOCAL A(2) : A(1)=1234567 : A(2)=-123.456789
00 02, 00 00 70 56 34 12 06 00, 00 90 78 56 34 12 02 80

String Array Variable

Format: MM; LL LL [strings]
Declared maximum length, Array size, list of strings
Size: LLLL*(MM+1)+3
Type Byte: 05

MM is shared maximum length of the strings.
LLLL is the number of strings in the array.
This is followed by the array contents, MM+1 bytes of storage space for each string of which the first is its length byte.

In OPL, ADDR() returns address of first string. In QCode, the array address is the address of the array size word.

Example: LOCAL A$(2,4) : A$(1)="ABC" : A$(2)="1234"
04; 00 02, 03 41 42 43 ??, 04 31 32 33 34

---

2. Language Stack

When a procedure is run, it is copied from whatever pack it is on into memory (even if it is on pack A:). It is placed on the language stack, which is an area of memory that grows downwards from the address given in the bta_sbas system variable ($2065/66). The lowest address currently in use by the stack is given by the language stack pointer rta_sp ($A5/A6).

When a procedure is placed on the language stack, it will have the following format:

Highest address, previous head of the stack

- Length (2 bytes) of data added to the stack
- Storage space for variables

- Global variable name table
- Parameters (two bytes each)
- Externals (two bytes each)
- Globals (storage size as explained above)
- Locals (storage size as explained above)

- QCode (translated program instructions)

Lowest address, New head of the stack ----------

The global variable name table contains all the information about the global variables declared in this procedure. If, later on, some procedure is called and loaded that uses global variables declared elsewhere, then the stack is examined to see if those variables have been declared in previously loaded procedures.

The parameters and externals (those global variables declared in other procedures) only need 2 bytes of storage space, which contains the memory address of the location somewhere highter up the language stack where the value of the variable is actually stored. Global variables (declared in this procedure) and local variables are given the amount of space listed in the previous section.

---

3. The procedure file format

A procedure file is a binary file of type $83, as described in the file formats document. Its binary data block has the following format:

VVVV	Total size that variables need on the stack. This is size of storage space reserved for local/global varables defined in this procedure plus two bytes for each external variable or parameter plus size of global variable name table.
QQQQ	Size of procedure QCode
XX	Number of parameters
[XX]	Type bytes for parameters, in reverse order
XXXX	Size of global variable name table in bytes
[string,	Global variable name (starts with length byte, name will include any final % or $)
BB,	Global variable type byte (0-5)
AAAA]	Global variable address (offset into reserved variable space) Variables are listed in order of use in the procedure
XXXX	Size of external variable name table in bytes
[string,	External variable name (starts with length byte, name will include any final % or $)
BB]	External variable type byte (0-5) Variables are listed in order of use in the procedure
XXXX	Size of string fixup table in bytes

[AAAA, Address of string variable or string array, minus 1 (offset into reserved variable space)
MM] Maximum number of bytes in string contents

XXXX Size of array fixup table in bytes
[AAAA, Address of array (offset into reserved variable space)
LLLL] Number of elements in the array

[CC] QCode, number of bytes is QQQQ above.

When a procedure is loaded, VVVV bytes are reserved on the stack. This is partially filled by a copy of the global variable name table (after checking that these names have not been defined in the name table of any previous procedure on the stack). Below that is the space reserved for variables. The addresses of parameters and externals are filled in (the latter by looking through the global variable name tables of previous procedures). The rest of the variable storage is filled with zeroes, except for the array lengths and maximum string lengths. These are filled in by using the data in the two fixup tables.

The addresses in the fixup tables and the addresses of variables in the global variable name table and in the QCode block are all relative to the top of the part of the stack reserved for the variables of the procedure. The variables are referred to by their offset from the top of this variable storage space. If no globals were defined (so the global variable name table would just consist of the length word 0000) then the first variable would have the offset address $0000-2-x where x is the space needed by the variable. An integer would therefore have offset address $FFFC.

---

4. Example QCode Header

The following example procedure uses all possible types of variable.

```
TEST:(P1,P2%,P3$) LOCAL L1,L2%,L3$(5) LOCAL
L4(4),L5%(5),L6$(6,12),L7% GLOBAL G1,G2%,G3$(13) GLOBAL
G4(4),G5%(5),G6$(6,14),G7% REM rest of code, including use of the
following externals PRINT E1;E2%;E3$ PRINT E4(4);E5%(5);E6$(6)
```

The translated procedure will have the following header data:

0000 : 017D	Size of variable space	
014C	Size of the qcode	**LOCAL L1, L2%, L3$(5)**
03	Number of parameters	
02 00 01	Parameter types	
0008 : 00 2F	Size of global varname table	
02 47 31 01 FFB3	G1, float type, address	
03 47 32 25 00 FFB1	G2%, integer type, address	
03 47 33 24 02 FFA3	G3$, string type, address	
02 47 34 04 FF80	G4, float array type, address	
03 47 35 25 03 FF74	G5%, integer array type, address	
03 47 36 24 05 FF18	G6$, string array type, address	
03 47 37 25 00 FF15	G7%, integer type, address	

0039 : 0020	Size of external varname table
02 45 31 01	E1, float type
03 45 32 25 00	E2%, integer type
03 45 33 24 02	E3$, string type
02 45 34 04	E4, float array type
03 45 35 25 03	E5%, integer array type
03 45 36 24 05	E6$, string array type
02 4C 35 04	E7%, integer type
005B : 000C	Size of string fixups
FF04 05	Fixup for L3$ maximum string length
FE85 0C	Fixup for L6$() maximum string length
FFA2 0D	Fixup for G3$ maximum string length
FF17 0E	Fixup for G6$() maximum string length
0069 : 0018	Size of array fixups
FEE2 0004	Fixup for L4() array length
FED6 0005	Fixup for L5%() array length
FE86 0006	Fixup for L6$() array length
FF80 0004	Fixup for G4() array length
FF74 0005	Fixup for G5%() array length
FF18 0006	Fixup for G6$() array length
0083 :	Start of the QCode instructions

The variable storage area used by the program will look as follows:

> **FFFE-FF Global variable name table length (will contain 002F) FFCF-FD Global variable name table, of length 002F, exact copy of the above. FFCD-CE P1 Parameters and externals use 2 bytes each FFCB-CC P2% FFC9-CA P3$ FFC7-C8 E1 FFC5-C6 E2% FFC3-C4 E3$ FFC1-C2 E4() FFBF-C0 E5%() FFBD-BE E6$() FFBB-BC L5() FFB3-BA G1 Float uses 8 bytes FFB1-B2 G2% Integer uses 2 bytes FFA3-B0 G3$ String uses 2+maxStrLen bytes, but maxStrLen value stored in preceding byte FFA2 Maximum string length of G3$ FF80-A1 G4() float array uses 2+8*arrLen bytes FF74-7F G5%() integer array uses 2+2*arrLen bytes FF18-73 G6$() String array uses 3 + (1+maxStrLen)*arrLen, but maxStrLen value stored in preceding byte FF17 Maximum string length of G6$() elements FF15-16 G7% FF0D-14 L1 FF0B-0C L2% FF05-0A L3$ FF04 Maximum string length of L3$ FEE2-03 L4() FED6-E1 L5% () FE86-D5 L6$() FE85 Maximum string length of L6$() elements**

---

5. QCode

The instructions in the procedure are translated into qcode. Almost every keyword in OPL (functions and statements) is translated into a single code byte. When such a qcode byte is executed it will gather its input by popping whatever arguments it needs from the stack, and possibly read some data bytes that immediately follow the qcode instruction byte. After its work is done, it may push a single resulting value back onto the stack.

Abbreviation for bytes following the QCode byte:

v	=	Address of variable, 2 bytes, offset into variable storage area
V	=	Address of external variable, 2 bytes, offset into variable storage area
m	=	Single byte, offset into calculator memory
f	=	Single byte, logical file name, 0-3 for A-D
I	=	Integer value
F	=	Float value (in compact form)
S	=	String value
B	=	Byte value
O	=	Single byte, 0=OFF, 1=ON
D	=	Two bytes as a relative distance. Distance to next instruction is 0002.
f+list	=	A logical file name as above, followed by a list of field names where each field is a variable type byte (0-2) followed by a string that is the field name. The list is terminated by a $88 qcode byte.

Abbreviation for stack pushes/pops:

I	=	Integer value
F	=	Float value
S	=	String value
B	=	Byte value
i	=	Integer reference
f	=	Float reference
s	=	String reference
params	=	List of parameters, list of parameter type bytes, byte with number of parameters.
FList	=	Either: fIB where B is zero byte or: [F]BB where first B is number of floats, second is 1.

QCode	+bytes	pops	pushes	Description
00	v	-	I	Push local/global integer variable value
01	v	-	F	Push local/global float variable value
02	v	-	S	Push local/global string variable value
03	v	I	I	Pop integer index and push local/global integer array variable value
04	v	I	F	Pop integer index and push local/global float array variable value
05	v	I	S	Pop integer index and push local/global string array variable value
06	m	-	F	Push calculator memory. Is followed by a byte indicating which of the 10 memories.
07	V	-	I	Push parameter/external integer variable value
08	V	-	F	Push parameter/external float variable value
09	V	-	S	Push parameter/external string variable value
0A	V	I	I	Pop integer index and push parameter/external integer array variable value
0B	V	I	F	Pop integer index and push parameter/external float array variable value
0C	V	I	S	Pop integer index and push parameter/external string array variable value
0D	v	-	i	Push local/global integer variable reference

0E	v	-	f	Push local/global float variable reference
0F	v	-	s	Push local/global string variable reference
10	v	I	i	Pop integer index and push local/global integer array variable reference
11	v	I	f	Pop integer index and push local/global float array variable reference
12	v	I	s	Pop integer index and push local/global string array variable reference
13	m	-	f	Push calculator memory reference. Is followed by a byte indicating which of the 10 memories.
14	V	-	i	Push parameter/external integer variable reference
15	V	-	f	Push parameter/external float variable reference
16	V	-	s	Push parameter/external string variable reference
17	V	I	i	Pop integer index and push parameter/external integer array variable reference
18	V	I	f	Pop integer index and push parameter/external float array variable reference
19	V	I	s	Pop integer index and push parameter/external string array variable reference
1A	f	S	I	Push file field as integer. Is followed by 1 byte logical file name (0-3 for A-D)
1B	f	S	F	Push file field as float. Is followed by 1 byte logical file name (0-3 for A-D)
1C	f	S	S	Push file field as string. Is followed by 1 byte logical file name (0-3 for A-D)
1D	f	S	I	Push reference of file integer field. Is followed by 1 byte logical file name (0-3 for A-D)
1E	f	S	F	Push reference of file float field. Is followed by 1 byte logical file name (0-3 for A-D)
1F	f	S	S	Push reference of file string field. Is followed by 1 byte logical file name (0-3 for A-D)
20	B	-	B	Push byte literal
21	I	-	I	Push word literal (same as integer literal)
22	I	-	I	Push integer literal
23	F	-	F	Push float literal
24	S	-	S	Push string literal
25	-	-	-	Special call to machine code. Not used by the organiser's compiler. Calls the machine code routine at the address given by the rta_1vct system variable. On return, if carry set then error B is raised.
26	-	-	-	Calls UT$LEAV, which quits OPL?? Not used by the organiser's compiler.
27	-	II	I	<
28	-	II	I	<=
29	-	II	I	>
2A	-	II	I	>=
2B	-	II	I	<>
2C	-	II	I	=
2D	-	II	I	+

2E	-	II	I	-
2F	-	II	I	*
30	-	II	I	/
31	-	II	I	**
32	-	I	I	- (unary)
33	-	I	I	NOT
34	-	II	I	AND
35	-	II	I	OR
36	-	FF	I	<
37	-	FF	I	<=
38	-	FF	I	>
39	-	FF	I	>=
3A	-	FF	I	<>
3B	-	FF	I	=
3C	-	FF	F	+
3D	-	FF	F	-
3E	-	FF	F	*
3F	-	FF	F	/
40	-	FF	F	**
41	-	F	F	- (unary)
42	-	F	I	NOT
43	-	FF	I	AND
44	-	FF	I	OR
45	-	SS	I	<
46	-	SS	I	<=
47	-	SS	I	>
48	-	SS	I	>=
49	-	SS	I	<>
4A	-	SS	I	=
4B	-	SS	S	+
4C	-	II	-	AT
4D	-	II	-	BEEP
4E	-	-	-	CLS
4F	O	-	-	CURSOR
50	O	-	-	ESCAPE
51	D	-	-	GOTO
52	-	-	-	OFF
53	D	-	-	ONERR
54	-	I	-	PAUSE
55	-	II	-	POKEB
56	-	II	-	POKEW
57	-	I	-	RAISE
58	-	F	-	RANDOMIZE
59	-	-	-	STOP
5A	-	-	-	TRAP
5B	-	-	-	APPEND
5C	-	-	-	CLOSE

5D	-	SS	-	COPY
5E	f+list	S	-	CREATE
5F	-	S	-	DELETE
60	-	-	-	ERASE
61	-	-	-	FIRST
62	-	-	-	LAST
63	-	-	-	NEXT
64	-	-	-	BACK
65	f+list	S	-	OPEN
66	-	I	-	POSITION
67	-	SS	-	RENAME
68	-	-	-	UPDATE
69	f	-	-	USE
6A	-	I	-	KSTAT
6B	-	s	-	EDIT
6C	-	i	-	INPUT integer
6D	-	f	-	INPUT float
6E	-	s	-	INPUT string
6F	-	I	-	PRINT integer
70	-	F	-	PRINT float
71	-	S	-	PRINT string
72	-	-	-	PRINT ,
73	-	-	-	PRINT newline
74	-	I	-	LPRINT integer
75	-	F	-	LPRINT float
76	-	S	-	LPRINT string
77	-	-	-	LPRINT ,
78	-	-	-	LPRINT newline
79	-	I/F/S	-	RETURN
7A	-	-	-	RETURN (integer 0)
7B	-	-	-	RETURN (float 0)
7C	-	-	-	RETURN (string "")
7D	S	params	F/I/S	Procedure call.
7E	D	I	-	Branch if false
7F	-	iI	-	Assign integer
80	-	fF	-	Assign float
81	-	sS	-	Assign string
82	-	B	-	drop byte from stack
83	-	I	-	drop integer from stack
84	-	F	-	drop float from stack
85	-	S	-	drop string from stack
86	-	I	F	autoconversion int to float
87	-	F	I	autoconversion float to int
88	-	-	-	End of field list for CREATE/OPEN
89	code	-	-	Inline assembly. Not used by the organiser's compiler. Return with carry clear and B=length of code (so that can skip to next qcode), or carry set and B=error code.

8A	-	i/f	I	ADDR
8B	-	S	I	ASC
8C	-	-	I	DAY
8D	-	IS	I	DISP
8E	-	-	I	ERR
8F	-	S	I	FIND
90	-	-	I	FREE
91	-	-	I	GET
92	-	-	I	HOUR
93	-	I	I	IABS
94	-	F	I	INT
95	-	-	I	KEY
96	-	S	I	LEN
97	-	SS	I	LOC
98	-	S	I	MENU
99	-	-	I	MINUTE
9A	-	-	I	MONTH
9B	-	I	I	PEEKB
9C	-	I	I	PEEKW
9D	-	-	I	RECSIZE
9E	-	-	I	SECOND
9F	-	II	I	USR
A0	-	II	I	VIEW
A1	-	-	I	YEAR
A2	-	-	I	COUNT
A3	-	-	I	EOF
A4	-	S	I	EXIST
A5	-	-	I	POS
A6	-	F	F	ABS
A7	-	F	F	ATAN
A8	-	F	F	COS
A9	-	F	F	DEG
AA	-	F	F	EXP
AB	-	F	F	FLT
AC	-	F	F	INTF
AD	-	F	F	LN
AE	-	F	F	LOG
AF	-	-	F	PI
B0	-	F	F	RAD
B1	-	-	F	RND
B2	-	F	F	SIN
B3	-	F	F	SQR
B4	-	F	F	TAN
B5	-	S	F	VAL
B6	-	-	F	SPACE
B7	-	S	S	DIR$
B8	-	I	S	CHR$

B9	-	-	S	DATIM$
BA	-	-	S	ERR$
BB	-	FII	S	FIX$
BC	-	FI	S	GEN$
BD	-	-	S	GET$
BE	-	I	S	HEX$
BF	-	-	S	KEY$
C0	-	SI	S	LEFT$
C1	-	S	S	LOWER$
C2	-	SII	S	MID$
C3	-	FI	S	NUM$
C4	-	SI	S	RIGHT$
C5	-	SI	S	REPT$
C6	-	FII	S	SCI$
C7	-	S	S	UPPER$
C8	-	II	S	USR$
C9	-	s	I	ADDR (string)
CA	SI	-	-	Used in .LNO files by the Developer Emulator to store procedure debug info.
CB	II	-	-	Used in .LNO files by the Developer Emulator to store line and columns number of a statement.

LZ only:

CC	-	FF	F	<%
CD	-	FF	F	>%
CE	-	FF	F	+%
CF	-	FF	F	-%
D0	-	FF	F	*%
D1	-	FF	F	/%
D2	-	I	-	OFFX
D3	-	SS	-	COPYW
D4	-	S	-	DELETEW
D5	-	IIIIIIII	-	UDG
D6	-	I	I	CLOCK
D7	-	III	I	DOW
D8	-	S	I	FINDW
D9	-	IS	I	MENUN
DA	-	III	I	WEEK
DB	-	F	F	ACOS
DC	-	F	F	ASIN
DD	-	III	F	DAYS
DE	-	Flist	F	MAX
DF	-	Flist	F	MEAN
E0	-	Flist	F	MIN
E1	-	Flist	F	STD
E2	-	Flist	F	SUM
E3	-	Flist	F	VAR

E4	-	I	S	DAYNAME$
E5	-	S	S	DIRW$
E6	-	I	S	MONTH$

For example, PRINT "O"; is translated into QCode consisting of a code to push a string literal onto the stack, immediately followed by the string literal "O", followed by the code for a print instruction.

```
PRINT "O"; 24 01 4F 71
```

The expression A%+1, where A% is a locally declared variable (local or global), is translated into QCode consisting of a code to push a locally declared integer variable to the stack, followed by the address of that variable (as an offset into the variable storage area), a code to push an integer followed by the literal integer 1, and finally a code to add two integers.

```
A%+1 00 FFFC 22 0001 2D
```

The address of the variable will be different if the procedure declares any global variables, parameters, uses any external variables, or declares any other variables before it declares A%.

If A% were an external variable, i.e. a parameter or a global variable declared in a previous procedure, then the expression A%+1 would be translated almost the same, except that the code to push an external integer variable to the stack is used instead. The rest is the same.

```
A%+1 07 FFFC 22 0001 2D
```

If a function is used as a statement, then its return value is explicitly removed from the stack afterwards. For example, if GET is used as a statement, then it is translated to the code for the GET function, followed by the code to drop an integer from the stack.

If at any point a float is needed but there is instead an integer on the stack (or vice versa), a code that does that conversion is is used. For example the expression ATAN(1) is translated to QCode consisting of a code to push a literal integer onto the stack followed by the integer 0001, then a code to convert an integer to a float, and finally the code for the ATAN function.

```
ATAN(1) 22 0001 86 A7
```

The only QCodes for flow control commands are branch-if-false (code 7E) and goto (code 51). There are no QCodes specific to the keywords, IF/ELSEIF/ELSE/ENDIF, WHILE/ENDWH, DO/UNTIL, or BREAK/CONTINUE, as these can all be expressed using only goto and branch-if-false. The distance to jump is given by the two bytes after the instruction. This distance is measured relative to the start of those two bytes, so a distance of 0002 would jump to the immediately following instruction, effectively doing nothing at all, whereas a distance of 0003 would skip over one byte. The distance can be negative, so a jump of distance FFFF is an infinite loop.

If the procedure does not end in an explicit RETURN statement, then it will be compiled just as if there were a bare RETURN statement there. The qcode will therefore always end with one of the qcode instructions 7A, 7B, or 7C, depending on the name of the procedure.

Procedures that have been translated for use on the 4-line LZ model have QCode that starts with the bytes 59 B2. The first byte is the code for the STOP command, so that if it is executed on a 2-line machine it will stop. The second byte is the code for the SIN function. No valid source program could ever produce these two codes after each other, so if the LZ encounters them at the start of the program, it can safely assume that it is a procedure meant for the LZ model and start execution at the next byte. Note that the two bytes form a neat pun for Stop Sign.

6. QCode Examples

This somewhat bogus example shows how flow control structures are translated into qcode.

```
TEST: IF 1 PRINT ENDIF IF 2. PRINT ELSE PRINT ENDIF IF 3 PRINT
ELSEIF 4 PRINT ENDIF IF 5 PRINT ELSEIF 6 PRINT ELSE PRINT ENDIF
WHILE 7 PRINT BREAK PRINT CONTINUE PRINT ENDWH DO PRINT BREAK
PRINT CONTINUE PRINT UNTIL 8
```

Translated procedure		Source Code	Remarks
		TEST:	
0000 : 0002	Size of variable space		It will contain only an empty global variable name table.
0062	Size of the qcode		
00	Number of parameters		
0000	Size of global varname table		
0000	Size of external varname table		
0000	Size of string fixup table		
0000	Size of array fixup table		
000D : 59B2	Stop sign		The procedure was compiled for LZ/LZ64
22 0001	Push integer 1	**IF 1**	A bare IF is translated with a single conditional jump
7E 0003	If non-zero, skip 1 byte		
73	Print	**PRINT**	
		ENDIF	
0016 : 23 022000	Push float 2.0	**IF 2.**	Note that when a float is used as a condition, the implicit <>0.0 comparison is added in the translation
23 020000	Push float 0.0		
3A	<>		
7E 0006	If non-zero, skip 4 bytes		The conditional branch jumps to the ELSE statement.
73	Print	**PRINT**	
51 0003	Skip 1 byte		Every non-final block will end with a jump to the ENDIF.
		ELSE	
73	Print	**PRINT**	
		ENDIF	
0027 : 22	Push integer 3	**IF 3**	

0003			
7E 0006	If non-zero, skip 4 bytes		The conditional branch jumps to the next ELSEIF statement.
73	Print	**PRINT**	
51 0009	Skip 7 bytes		Every non-final block will end with a jump to the ENDIF.
22 0004	Push integer 4	**ELSEIF 4**	
7E 0003	If non-zero, skip 1 byte		The conditional branch jumps to the ENDIF statement.
73	Print	**PRINT**	
		ENDIF	
0038 : 22 0005	Push integer 5	**IF 5**	
7E 0006	If non-zero, skip 4 bytes		The conditional branch jumps to the next ELSEIF statement.
73	Print	**PRINT**	
51 000D	Skip 11 bytes		Every non-final block will end with a jump to the ENDIF.
22 0006	Push integer 6	**ELSEIF 6**	
7E 0006	If non-zero, skip 4 bytes		The conditional branch jumps to the ELSE statement.
73	Print	**PRINT**	
51 0003	Skip 1 byte		Every non-final block will end with a jump to the ENDIF.
		ELSE	
73	Print	**PRINT**	
		ENDIF	
004D : 22 0007	Push integer 7	**WHILE 7**	
7E 000E	If non-zero, skip 12 bytes		The conditional branch jumps to after the ENDWH statement.
73	Print	**PRINT**	
51 000A	Skip 8 bytes	**BREAK**	BREAK jumps forward to after the ENDWH statement.
73	Print	**PRINT**	
51 FFF4	Skip back 11 bytes	**CONTINUE**	CONTINUE jumps back to the WHILE condition.
73	Print	**PRINT**	
51 FFF0	Skip back 15 bytes	**ENDWH**	Jump back to the WHILE condition.
		DO	
005F : 73	Print	**PRINT**	
51 000D	Skip 11 bytes	**BREAK**	BREAK jumps forward to after the UNTIL statement.
73	Print	**PRINT**	
51 0003	Skip 1 byte	**CONTINUE**	CONTINUE jumps forward to the UNTIL condition.
73	Print	**PRINT**	
22 0008	Push integer 8	**UNTIL 8**	

7E FFF3	If non-zero, skip back 12 bytes	The conditional branch jumps to the DO statement.
006E : 7B	Return 0.0	Implicit return made explicit

This example shows how all the different types of variable are handled.

```
TEST2:(P1,P2%,P3$) LOCAL L1,L2%,L3$(5) LOCAL
L4(4),L5%(5),L6$(6,12),L7% GLOBAL G1,G2%,G3$(13) GLOBAL
G4(4),G5%(5),G6$(6,14),G7% PRINT "PPP";P1;P2%;P3$ PRINT
"LLL";L1;L2%;L3$ PRINT "LL";L4(4);L5%(5);L6$(6) PRINT
"GGG";G1;G2%;G3$ PRINT "GG";G4(4);G5%(5);G6$(6) PRINT
"EEE";E1;E2%;E3$ PRINT "EE";E4(4);E5%(5);E6$(6) L1=234 L2%=345
L3$="BCD" L4(2)=345 L5(3)=456 L6$(4)="CDE" G1=12 G2%=23 G3$="DEF"
G4(3)=34 G5%(4)=4 G6$(5)="EFG" E1=45 E2%=56 E3$="FGH" E4(4)=67
E5%(5)=78 E6$(6)="GHI"
```

0000 : 017D	Size of variable space	**TEST2:(P1,P2%,P3$)**
014C	Size of the qcode	**LOCAL L1,L2%,L3$(5)**
03	Number of parameters	**LOCAL L4(4),L5%(5),L6$(6,12) L7%**
02 00 01	Parameter types	**GLOBAL G1,G2%,G3$(13)**
		GLOBAL G4(4),G5%(5),G6$(6,14),G7%
0008 : 00 2F	Size of global varname table	
02 47 31 01 FFB3	G1, float type, address	
03 47 32 25 00 FFB1	G2%, integer type, address	
03 47 33 24 02 FFA3	G3$, string type, address	
02 47 34 04 FF80	G4, float array type, address	
03 47 35 25 03 FF74	G5%, integer array type, address	
03 47 36 24 05 FF18	G6$, string array type, address	
03 47 37 25 00 FF15	G7%, integer type, address	
0039 : 0020	Size of external varname table	
02 45 31 01	E1, float type	
03 45 32 25 00	E2%, integer type	
03 45 33 24 02	E3$, string type	
02 45 34 04	E4, float array type	
03 45 35 25 03	E5%, integer array type	
03 45 36 24 05	E6$, string array type	
02 4C 35 04	E7%, integer type	
005B : 000C	Size of string fixups	
FF04 05	Fixup for L3$ maximum string length	
FE85 0C	Fixup for L6$() maximum string length	
FFA2 0D	Fixup for G3$ maximum string length	
FF17 0E	Fixup for G6$() maximum string length	

0069 : 0018	Size of array fixups	
FEE2 0004	Fixup for L4() array length	
FED6 0005	Fixup for L5%() array length	
FE86 0006	Fixup for L6$() array length	
FF80 0004	Fixup for G4() array length	
FF74 0005	Fixup for G5%() array length	
FF18 0006	Fixup for G6$() array length	
0083 : 59B2	Stop sign	
24 03505050	Push "PPP"	**PRINT "PPP";P1;P2%;P3$**
71	Print string	
08 FFCD	Push external flt var value P1	
70	Print float	
07 FFCB	Push external int var value P2%	
6F	Print integer	
09 FFC9	Push external str var value P3$	
71	Print string	
73	Print newline	
0098 : 24 034C4C4C	Push "LLL"	**PRINT "LLL";L1;L2%;L3$**
71	Print string	
01 FF0D	Push flt var value L1	
70	Print float	
00 FF0B	Push int var value L2%	
6F	Print integer	
02 FF05	Push str var value L3$	
71	Print string	
73	Print newline	
00AB : 24 024C4C	Push "LL"	**PRINT "LL";L4(4);L5%(5);L6$(6)**
71	Print string	
22 0004	Push integer 4	
04 FEE2	Push flt arr value L4(.)	
70	Print float	
22 0005	Push integer 5	
03 FED6	Push int arr value L5%(.)	
6F	Print integer	
22 0006	Push integer 6	
05 FE86	Push str arr value L6$(.)	
71	Print string	
73	Print newline	
00C6 : 24 03474747	Push "GGG"	**PRINT "GGG";G1;G2%;G3$**
71	Print string	
01 FFB3	Push flt var value G1	
70	Print float	
00 FFB1	Push int var value G2%	
6F	Print integer	
02 FFA3	Push str var value G3$	
71	Print string	

73	Print newline	
00D9 : 24 024747	Push "GG"	**PRINT "GG";G4(4);G5%(5);G6$(6)**
71	Print string	
22 0004	Push integer 4	
04 FF80	Push flt arr value G4(.)	
70	Print float	
22 0005	Push integer 5	
03 FF74	Push int arr value G5%(.)	
6F	Print integer	
22 0006	Push integer 6	
05 FF18	Push str arr value G6$(.)	
71	Print string	
73	Print newline	
00F4 : 24 03454545	Push "EEE"	**PRINT "EEE";E1;E2%;E3$**
71	Print string	
08 FFC7	Push external flt var value E1	
70	Print float	
07 FFC5	Push external int var value E2%	
6F	Print integer	
09 FFC3	Push external str var value E3$	
71	Print string	
73	Print newline	
0107 : 24 024545	Push "EE"	**PRINT "EE";E4(4);E5%(5);E6$(6)**
71	Print string	
22 0004	Push integer 4	
0B FFC1	Push external flt arr value E4(.)	
70	Print float	
22 0005	Push integer 5	
0A FFBF	Push external int arr value E5%(.)	
6F	Print integer	
22 0006	Push integer 6	
0C FFBD	Push external str arr value E6$(.)	
71	Print string	
73	Print newline	
0122 : 0E FF0D	Push flt var ref L1	**L1=234**
22 00EA	Push integer 234	
86	Convert int to flt	
80	Assign float	
012A : 0D FF0B	Push int var ref L2%	**L2%=345**
22 0159	Push integer 345	
7F	Assign integer	
0131 : 0F FF05	Push str var ref L3$	**L3$="BCD"**
24 03424344	Push "BCD"	
81	Assign string	
013A : 22 0002	Push integer 2	**L4(2)=345**

11 FEE2	Push flt arr ref L4(.)	
22 0159	Push integer 345	
86	Convert int to flt	
80	Assign float	
0145 : 22 0003	Push integer 3	**L5(3)=456**
18 FFBB	Push int arr ref L5%(.)	
22 01C8	Push integer 456	
86	Convert int to flt	
80	Assign float	
0150 : 22 0004	Push integer 4	**L6$(4)="CDE"**
12 FE86	Push str arr ref L6$(.)	
24 03434445	Push "CDE"	
81	Assign string	
015C : 0E FFB3	Push flt var ref G1	**G1=12**
22 000C	Push integer 12	
86	Convert int to flt	
80	Assign float	
0164 : 0D FFB1	Push int var ref G2%	**G2%=23**
22 0017	Push integer 23	
7F	Assign integer	
016B : 0F FFA3	Push str var ref G3$	**G3$="DEF"**
24 03444546	Push "DEF"	
81	Assign string	
0174 : 22 0003	Push integer 3	**G4(3)=34**
11 FF80	Push flt arr ref G4(.)	
22 0022	Push integer 34	
86	Convert int to flt	
80	Assign float	
017F : 22 0004	Push integer 4	**G5%(4)=45**
10 FF74	Push int arr ref G5%(.)	
22 002D	Push integer 45	
7F	Assign integer	
0189 : 22 0005	Push integer 5	**G6$(5)="EFG"**
12 FF18	Push str arr ref G6$(.)	
24 03454647	Push "EFG"	
81	Assign string	
0195 : 15 FFC7	Push ext flt var ref E1	**E1=45**
22 002D	Push integer 45	
86	Convert int to flt	
80	Assign float	
019D : 14 FFC5	Push ext int var ref E2%	**E2%=56**
22 0038	Push integer 56	
7F	Assign integer	

01A4 : 16 FFC3	Push ext str var ref E3$	**E3$="FGH"**
24 03464748	Push "FGH"	
81	Assign string	
01AD : 22 0004	Push integer 4	**E4(4)=67**
18 FFC1	Push ext flt arr ref E4(.)	
22 0043	Push integer 67	
86	Convert int to flt	
80	Assign float	
01B8 : 22 0005	Push integer 5	**E5%(5)=78**
17 FFBF	Push ext int arr ref E5%(.)	
22 004E	Push integer 78	
7F	Assign integer	
01C2 : 22 0006	Push integer 6	**E6$(6)="GHI"**
19 FFBD	Push ext str arr ref E6$(.)	
24 03474849	Push "GHI"	
81	Assign string	
01CE : 7B	Return 0.0	

---

7. LNO files

The Organiser Developer kit contains an emulator ORG2.EXE. This emulator allows you to translate, run, and debug OPL procedures. When it translates a procedure, it produces a file with the file extension .LNO, which contains the translated result, similar to an object-only .OB3 file.

If the procedure is translated in the emulator while its DEBUG mode is OFF, then the LNO file is virtually the same as an OB3 file of the same program would be. It differs only in the following aspects:

- The file type is FE instead of 83, i.e. the last byte of the 6-byte file header is FE rather than 83.
- It has no source code block.
- The length word of the file (the fourth and fifth byte of the 6-byte file header) is two less than the length of the OB3 file would be, because it has no source code block at all instead of merely an empty one.
- Every two-byte integer within the qcode and the qcode header is stored little-endian, i.e. low byte followed by high byte.
- It may have some padding at the end, such as FFFF in the place where the empty source code block would be.

Consider for example, the following small test procedure:

```
TEST: LOCAL A% A%=1234 AT 4,1 :PRINT A% GET
```

Here are the OB3 and non-debug LNO files produced by translating the procedure.

OB3	LNO	
4F 52 47	**4F 52 47**	"ORG" magic number

00 29	**00 27**	File Length word
83	**FE**	Block File type
00 25	**00 25**	Block length
00 04	**04 00**	Size of variable space
00 18	**18 00**	QCode length
00	**00**	Number of parameters
00 00	**00 00**	Global varname table
00 00	**00 00**	External varname table
00 00	**00 00**	String fixups
00 00	**00 00**	Array fixups
59 B2	**59 B2**	Stop sign
0D FF FC	**0D FC FF**	A% reference
22 04 D2	**22 D2 04**	1234
7F	**7F**	Assign integer
22 00 04	**22 04 00**	4
22 00 01	**22 01 00**	1
4C	**4C**	AT
00 FF FC	**00 FC FF**	A% value
6F	**6F**	PRINT integer
73	**73**	PRINT newline
91	**91**	GET
83	**83**	Discard integer
7B	**7B**	RETURN
00 00	**FF FF**	Source code block / padding

An LNO file that is created while DEBUG mode is ON will have extra debugging information incorporated into the file, such as the names of all the local variables and parameters, as well as the line number of each statement.

The QCodes CA and CB are used to encode some of this information. The first QCode after the QCode header (or after the stop sign if present) is CA, which is followed by a string literal containing the procedure name, followed by a length word which is 4 plus the length of the remaining QCode. Every section of QCode that represents a single source code statement is preceded by 5 bytes - the CB QCode, a word containing the source line number of the statement (where the top line containing the procedure name is line 1) and the column number of the statement (where column 0 means the start of the line).

Information about the variables is stored after the QCode block, where in an OB3 file the source code block would be. It starts with a word containing the number of variables that follow, and then data for each variable in turn. That data consists of:

1. A string containing the variable's name.
2. A word containing the address of the variable (the usual offset into the variable space)
3. A byte which is 00 for external variables, 01 for parameter variables, 02 for global variables, and 03 for local variables.
4. A variable type byte - which is the usual 00 for an integer, 01 for a floating point, 02 for a string, 03 for an integer array, 04 for a floating point array, and 05 for a string array.

The previous TEST program translated in DEBUG mode is as follows:

LNO, no debug	**LNO, debug**	
4F 52 47	**4F 52 47**	"ORG" magic number

0027	**0051**	File Length word
FE	**FE**	Block File type
0025	**004f**	Block length
0400	**0400**	Size of variable space
1800	**3900**	QCode length
00	**00**	Number of parameters
0000	**0000**	Global varname table
0000	**0000**	External varname table
0000	**0000**	String fixups
0000	**0000**	Array fixups
59 B2	**59 B2**	Stop sign
	CA	Procedure debug info
	04 54 45 53 54	"TEST"
	3300	Length
	CB 0200 0000	Line 2 column 0
0D FCFF 22 D204 7F	**0D FCFF 22 D204 7F**	A%=1234
	CB 0300 0000	Line 3 column 0
22 0400 22 0100 4C	**22 0400 22 0100 4C**	AT 1,4
	CB 0300 0800	Line 3 column 8
00 FCFF 6F 73	**00 FCFF 6F 73**	PRINT A%
	CB 0400 0000	Line 4 column 0
91 83	**91 83**	GET
	CB 0500 0000	Line 5 column 0
7B	**7B**	RETURN
FF FF	**0100**	Padding / Number of variables
	02 41 25	"A%"
	FCFF	Address of A%
	03	Local variable
	00	Integer variable
