LUCIDATA Pascal

Run-time System

Version 1.2

December 5, 2021, Preliminary

While it is believed that the code will perform as intended, the user assumes all risk in connection with the use of this software.

Table of Contents

Introduction

This is a description of the run-time system for LUCIDATA Pascal. It is the result of detailed analysis of version 1.2 of P-6800 for the Motorola 6800 microprocessor running the FLEX operating system.

This research was done to enable implementing a compatible interpreter running on a 6502 microprocessor.

The Stack Machine

The LUCIDATA Pascal run-time system implements a virtual stack-oriented architecture.

P-code instructions are four bytes in length. The first byte identifies the instruction.

The 6800 instruction dispatcher copies the second, third and fourth bytes of the instruction to memory locations $DA, $DB and $DC respectively before passing control to the instruction handler. The 6502 dispatcher names these locations InstrMode, InstrParm1 and InstrParm2 respectively.

For the instruction to calculate the offset into an array from subscript(s), the second byte is the number of dimensions, the third byte is the size of an array element and the fourth byte identifies the first entry into the subscript range table for this array.

The word at $F2 points to the first entry of the subscript range table.

A subscript range entry for an array [3..7] would look like this:

0 ; High byte of lowest allowed subscript
3 ; Low byte of lowest allowed subscript
0 ; High byte of number of subscripts in the allowed range
5 ; Low byte of number of subscripts in the allowed range

For the 6502 implementation, all multi-byte entities except for those used indirect indexed addressing are stored in big-endian format.

The subscript(s) have been pushed onto the stack in left to right order; that is, the rightmost subscript is at the top of the stack.

The stack pointer points to the last byte of the top item on the stack.

The 6800 implementation uses the two bytes following the storage for the stack pointer pseudo-register (Reg_SP) for scratch space; the 6502 implementation will do the same. In this case, it points to the byte before one of the subscripts pushed onto the stack instead of passing it in registers.

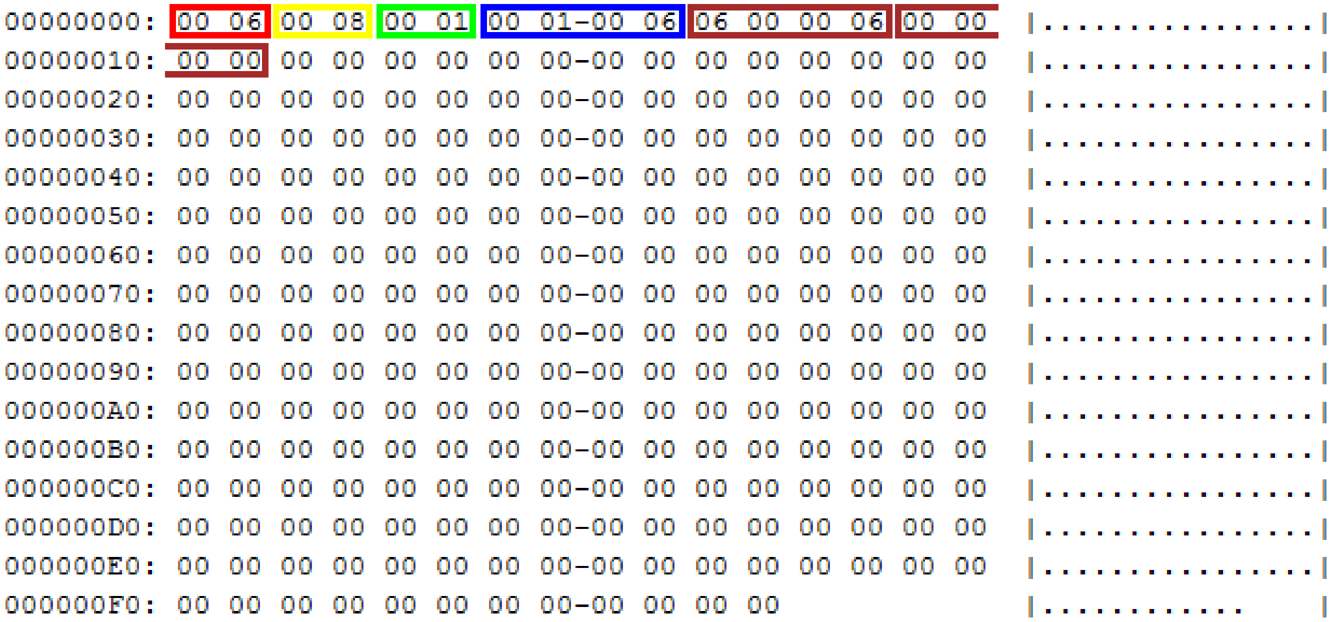
File Format

The following program:

```
PROGRAM TEST;

  BEGIN
  END.
```

compiles to:



```
00000000: 00 06 00 08 00 01 00 01-00 06 06 00 00 06 00 00  |................|
00000010: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
00000020: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
00000030: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
00000040: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
00000050: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
00000060: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
00000070: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
00000080: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
00000090: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
000000A0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
000000B0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
000000C0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
000000D0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
000000E0: 00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00  |................|
000000F0: 00 00 00 00 00 00 00 00-00 00 00 00              |............    |
```

All multi-byte fields are stored in big-endian byte order, that is, most-significant byte first.

The data in red is the highest numbered instruction opcode used by this program.

The data in yellow is the length of the program image in bytes.

The data in green is the number of entries in the subscript range table.

The data in blue is the subscript range table. Each entry consists of four bytes. The first two is the lower bound of the range; the second two is the number of values in the range. The entry shown is the range for the ALFA type predeclared as an array [1..6] of char.

The data in brown is the program image. Each instruction is a multiple of four bytes long.

The Instruction Set

Instructions are a multiple of four bytes.

The first byte is an opcode identifying the instruction.

The second byte often serves as a mode indicating variations of the instruction or allowing several different operations to share one opcode.

The third and fourth bytes provide additional information as needed.

Program execution begins with the instruction at address $0000 and continues until the halt instruction is encountered or a fatal error occurs.

Unless otherwise specified, unused fields are filled with zero values.

$00 - Halt

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $00     |    $00     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Close files and terminate the program.

$00 - Case Variable Error

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $00     |    $01     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Issue a case variable error message and terminate the program.

Note: Version 1.2 of the run-time system treats all non-zero mode forms of this instruction as a case variable error.

$01 - Jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |   Address  |   Address  |
|    $01     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Program execution continues at the specified target address.

$02 - Nop

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $02     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Execution continues with the following instruction.

Note: the stack is not changed.

$02 - If false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $02     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove the byte on the top of the stack. If it is zero, program execution continues at the specified target address, otherwise continue with the following instruction.

$03 - Complete subroutine call

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Target   |   Target   |
|            |   Level    |  Address   |  Address   |
|    $03     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Update the activation record at the frame pointer and transfer control to the subroutine at the target address. An allocate return value or create stack frame instruction must be executed before invoking this instruction.

$04 - Create stack frame

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |            |            |            |
|            |            |            |            |
|    $04     |    $00     |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Create a stack frame. This or an allocate return value instruction must be executed before pushing parameters for a subroutine call.

$04 - Allocate return value

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |            |   Return   |   Return   |
|            |            |    Value   |    Value   |
|    $04     |    $00     |    Size    |    Size    |
|            |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Allocate space on the stack for a return value then create a stack frame. This or a create stack frame instruction must be executed before pushing parameters for a subroutine call.

$04 - Call procedure

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Target   |   Target   |
|            |   Level    |  Address   |  Address   |
|    $04     |            |            |            |
|            | %1000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Call a procedure needing no parameters by creating a stack frame and transferring control to the subroutine at the target address.

$05 - Return

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Unused   |   Unused   |
|            |            |            |            |
|    $05     |            |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Return control from a subroutine.

$06 - Manage Stack Pointer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Offset   |   Offset   |
|            |            |            |            |
|    $06     |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

If the Mode is zero, the Offset is added to the Stack Pointer and the result checked for a stack overflow, otherwise, the Offset is subtracted from the Stack Pointer. If the Mode and Offset are both zero, the stack is checked for overflow.

$07 - Push Constant

Format:

```
+-----------+-----------+-----------+-----------+
|           |           |           |           |
|  Opcode   |   Size    |   Data    |   Data    |
|           |           |           |           |
|   $07     |           |   byte    |   byte    |
|           |           |    #1     |    #2     |
|           |           |           |           |
+-----------+-----------+-----------+-----------+
```

Function:

Push a constant value onto the stack and leave the Stack Pointer pointing its last (highest address) byte.

If the Size is one, the Data in byte #2 is pushed onto the stack.

If the Size is greater than two, additional instruction words are added until the entire constant is encoded. The last instruction word is padded as necessary with random bytes.

$08 - Determine Array Index

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   | Dimensions |   Element  |    First   |
|            |            |    Size    |  Subscript |
|    $08     |            |            |    Entry   |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Determines the index to an element in an array.

$09 - Not

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |  Target    |  Target    |
|            |            | Address    | Address    |
|    $09     |    $00     |            |            |
|            |            |   $00      |   $00      |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Invert the byte at the top of the stack.

$09 - Not, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |  Target    |  Target    |
|            |            | Address    | Address    |
|    $09     |    $00     |            |            |
|            |            |   high     |   low      |
|            |            |   byte     |   byte     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove and invert the byte at the top of the stack. If the result is false, program execution continues at the target address, otherwise continue with the following instruction.

$09 - And

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |  Target    |  Target    |
|            |            | Address    | Address    |
|    $09     |    $01     |            |            |
|            |            |   $00      |   $00      |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove and perform an and operation with two bytes at the top of the stack. If the result is false, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$09 - And, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |  Target    |  Target    |
|            |            | Address    | Address    |
|    $09     |    $01     |            |            |
|            |            |   high     |   low      |
|            |            |   byte     |   byte     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove and perform an and operation with two bytes at the top of the stack. If the result is false, program execution continues at the target address, otherwise continue with the following instruction.

$09 - Or

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $09     |    $02     |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove and perform an or operation with two bytes at the top of the stack. If the result is false, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

Note: Version 1.2 of the run-time system treats all values of mode other than zero or one as or.

$09 - Or, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $09     |    $02     |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove and perform an or operation with two bytes at the top of the stack. If the result is false, program execution continues at the target address, otherwise continue with the following instruction.

Note: Version 1.2 of the run-time system treats all values of mode other than zero or one as or.

$0A - unknown

$0B - Call user function

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $0B     |    $00     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Call user function whose address is at $01A2.

Note: Version 1.2 of the run-time system treats all values of mode other than one or two as call user function.

$0B - Peek

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $0B     |    $01     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

One byte must be reserved for the return value before executing this instruction. Remove the integer on the top of the stack; it is the address of the memory location to peek. Read that value and push it onto the stack.

$0B - Poke

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $0B     |    $02     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

A stack frame must be created before executing this instruction. Remove the byte at the top of the stack. Then remove the integer at the top of the stack; it is the address to write the byte.

$0C - EOLN

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Target   |   Target   |
|            |   Number   |   Address  |   Address  |
|    $0C     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Check end of line status. If false, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$0C - EOLN, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Target   |   Target   |
|            |   Number   |   Address  |   Address  |
|    $0C     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Check end of line status. If false, program execution continues at the target address, otherwise continue with the following instruction.

$0D - EOF

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Target   |   Target   |
|            |   Number   |   Address  |   Address  |
|    $0D     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Check end of file status. If false, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$0D - EOF, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Target   |   Target   |
|            |   Number   |   Address  |   Address  |
|    $0D     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Check end of file status. If false, program execution continues at the target address, otherwise continue with the following instruction.

$0E - Rewrite file

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Unused   |   Unused   |
|            |   Number   |            |            |
|    $0E     |            |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Open a file for writing. The file number 2 is associated with the standard output text stream. File numbers 3 through 8 are associated with files on disk by naming them on the command line of the RUN command; unassigned files are associated with the system console. An existing disk file is truncated and if it is not written, deleted when the program terminates.

$0F - Reset file

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Unused   |   Unused   |
|            |   Number   |            |            |
|    $0F     |            |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Open a file for reading. The file number 1 is associated with the standard input text stream. File numbers 3 through 8 are associated with files on disk by naming them on the command line of the RUN command; unassigned files are associated with the system console.

$10 - Compare bytes for =

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $10     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two bytes at the top of the stack; they are removed from the stack in the process. If the bytes are not equal, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$10 - Compare bytes for =, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $10     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two bytes at the top of the stack; they are removed from the stack in the process. If the bytes are not equal, program execution continues at the specified target address, otherwise continue with the following instruction.

$11 - Compare bytes for <>

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $11     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two bytes at the top of the stack; they are removed from the stack in the process. If the bytes are equal, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$11 - Compare bytes for <>, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $11     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two bytes at the top of the stack; they are removed from the stack in the process. If the bytes are equal, program execution continues at the specified target address, otherwise continue with the following instruction.

$12 - reserved for future use

Possibly for comparing bytes for <

$13 - reserved for future use

Possibly for comparing bytes for >

$14 - reserved for future use

Possibly for comparing bytes for <=

$15 - reserved for future use

Possibly for comparing bytes for >=

$16 - Push byte

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $16     |            |            |            |
|            | %0000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Push a byte onto the stack. The low order four bits of the nesting level specifies which stack frame contains the variable containing the byte. The frame offset is the offset of the variable from the beginning of its stack frame.

$16 - Push byte from array

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $16     |            |            |            |
|            | %1000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Push a byte onto the stack. The integer originally on the top of the stack provides the offset from the base of the array of the element to push; it is removed first. The low order four bits of the nesting level specifies which stack frame contains the array containing the byte. The frame offset is the offset of the array from the beginning of its stack frame.

$17 - Pop byte

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $17     |            |            |            |
|            | %0000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Pop a byte from the stack. The low order four bits of the nesting level specifies which stack frame contains the variable to receive the byte. The frame offset is the offset of the variable from the beginning of its stack frame.

$17 - Pop byte into array

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $17     |            |            |            |
|            | %1000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Pop a byte from the stack. The integer originally on the top of the stack provides the offset from the base of the array to the element to receive the byte; it is removed first. The low order four bits of the nesting level specifies which stack frame contains the array to receive the byte. The frame offset is the offset of the array from the beginning of its stack frame.

$18 - Convert byte to integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Unused   |   Unused   |
|            |            |            |            |
|    $18     |            |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Zero extend the byte on the top of the stack to an integer.

$19 - Convert unsigned integer to byte

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Unused   |   Unused   |
|            |            |            |            |
|    $19     |            |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Convert the unsigned integer on the top of the stack to a byte. An error is reported if the integer is greater than 255.

$1A - Convert character to integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $1A     |    $00     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Convert the character on the top of the stack to an integer. An error is reported if the ordinal value of the character is greater then 127.

$1A - Convert integer to character

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $1A     |    $01     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Convert the integer on the top of the stack to a character. An error is reported if the integer is greater than 255.

$1B - Succ

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Lower    |   Upper    |
|            |            |   Bound    |   Bound    |
|    $1B     |    $00     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Increment the byte on the top of the stack and check it against the upper bound. A scaler range error is reported if it is out of range.

$1B - Pred

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Lower    |   Upper    |
|            |            |   Bound    |   Bound    |
|    $1B     |    $01     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Decrement the byte on the top of the stack and check it against the lower bound. A scaler range error is reported if it is out of range.

Note: Version 1.2 of the run-time system treats all positive values of mode other than zero as pred.

$1B - Bounds check

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Lower    |   Upper    |
|            |            |   Bound    |   Bound    |
|    $1B     |    $80     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare the byte on the top of the stack with the lower and upper bounds. If the value is out of bounds, $00 (False) replaces the byte, otherwise $FF (True) is used.

Note: Version 1.2 of the run-time system treats all negative values of mode as bounds check.

$1C - Writeln

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Unused   |   Unused   |
|            |   Number   |            |            |
|    $1C     |            |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Start a new line in a file of char.

$1D - Readln

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Unused   |   Unused   |
|            |   Number   |            |            |
|    $1D     |            |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Read characters from a file of char until a new line is encountered.

$1E - Write string

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Field    |   String   |
|            |   Number   |   Width    |   Length   |
|    $1E     |            |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Write the string on the top of the stack to a file, then remove the string. If the field width is greater than the string length, leading spaces are emitted as padding. If the field width is less than the string length, only the characters fitting within the field are emitted.

$1F - Read byte

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Unused   |   Format   |
|            |   Number   |            |            |
|    $1F     |            |            |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Read a byte from a file of byte and push it onto the stack.

$1F - Read character

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Unused   |   Format   |
|            |   Number   |            |            |
|    $1F     |            |            |    $01     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Read a character from a file of char and push it onto the stack.

$20 - Compare integers for =

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $20     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the integers are not equal, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$20 - Compare integers for =, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $20     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the integers are not equal, program execution continues at the specified target address, otherwise continue with the following instruction.

$21 - Compare integers for <>

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $21     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the integers are equal, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$21 - Compare integers for <>, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $21     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the integers are equal, program execution continues at the specified target address, otherwise continue with the following instruction.

$22 - Compare integers for <

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $22     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the second topmost integer is not less than the topmost one, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$22 - Compare integers for <, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $22     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the second topmost integer is not less than the topmost one, program execution continues at the specified target address, otherwise continue with the following instruction.

$23 - Compare integers for >

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $23     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the second topmost integer is not greater than the topmost one, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$23 - Compare integers for >, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $23     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the second topmost integer is not greater than the topmost one, program execution continues at the specified target address, otherwise continue with the following instruction.

$24 - Compare integers for <=

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $24     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the second topmost integer is not less than or equal to the topmost one, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$24 - Compare integers for <=, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $24     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the second topmost integer is not less than or equal to the topmost one, program execution continues at the specified target address, otherwise continue with the following instruction.

$25 - Compare integers for >=

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $25     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the second topmost integer is not greater than or equal to the topmost one, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$25 - Compare integers for >=, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $25     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two integers at the top of the stack; they are removed from the stack in the process. If the second topmost integer is not greater than or equal to the topmost one, program execution continues at the specified target address, otherwise continue with the following instruction.

$26 - Push integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $26     |            |            |            |
|            | %0000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Push an integer onto the stack. The low order four bits of the nesting level specifies which stack frame contains the variable containing the integer. The frame offset is the offset of the variable from the beginning of its stack frame.

$26 - Push integer from array

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $26     |            |            |            |
|            | %1000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Push an integer onto the stack. The integer originally on the top of the stack provides the offset from the base of the array of the element to push; it is removed first. The low order four bits of the nesting level specifies which stack frame contains the array containing the integer. The frame offset is the offset of the array from the beginning of its stack frame.

$27 - Pop integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $27     |            |            |            |
|            | %0000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Pop an integer from the stack. The low order four bits of the nesting level specifies which stack frame contains the variable to receive the integer. The frame offset is the offset of the variable from the beginning of its stack frame.

$27 - Pop integer into array

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $27     |            |            |            |
|            | %1000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Pop an integer from the stack. The integer originally on the top of the stack provides the offset from the base of the array to the element to receive the integer; it is removed first. The low order four bits of the nesting level specifies which stack frame contains the array to receive the integer. The frame offset is the offset of the array from the beginning of its stack frame.

$28 - Add integers

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $28     |    $00     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Add two integers at the top of the stack and replace them with the sum.

$28 - Add constant to integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |  Constant  |  Constant  |
|            |            |  Operand   |  Operand   |
|    $28     |    $01     |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Add the integer on the top of the stack with a constant and replace it with the sum.

Note: Version 1.2 of the run-time system treats all values of Mode other than zero as one.

$29 - Subtract integers

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $29     |    $00     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Subtract the integer at the top of the stack from the second one and replace them with the difference.

$29 - Subtract constant from integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |  Constant  |  Constant  |
|            |            |  Operand   |  Operand   |
|    $29     |    $01     |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Subtract a constant from the integer on the top of the stack and replace it with the difference.

Note: Version 1.2 of the run-time system treats all values of Mode other than zero as one.

$2A - Multiply integers

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $2A     |    $00     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Multiply two integers at the top of the stack and replace them with the product.

$2A - Multiply integer by constant

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |  Constant  |  Constant  |
|            |            |  Operand   |  Operand   |
|    $2A     |    $01     |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Multiply the integer on the top of the stack by a constant and replace it with the product.

Note: Version 1.2 of the run-time system treats all values of Mode other than zero as one.

$2B - Divide integers

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $2B     |    $00     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Divide the integer at the top of the stack into the second one and replace them with the quotient.

$2B - Divide constant into integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |  Constant  |  Constant  |
|            |            |  Operand   |  Operand   |
|    $2B     |    $01     |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Divide a constant into the integer on the top of the stack and replace it with the quotient.

Note: Version 1.2 of the run-time system treats all values of Mode other than zero as one.

$2C - Negate integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Unused   |   Unused   |
|            |            |            |            |
|    $2C     |            |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Negate the integer on the top of the stack.

$2D - Odd

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $2D     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

The integer (two bytes) at the top of the stack is excluded and tested. If it is odd, $FF (True) is pushed onto the stack, otherwise $00 (False) is pushed. Execution continues with the following instruction.

Note: version 1.2 has a bug. The meaning of odd is reversed for negative numbers.

$2D - Odd, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $2D     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

The integer (two bytes) at the top of the stack is excluded and tested. If it is not odd, program execution continues at the specified target address, otherwise continue with the following instruction.

Note: version 1.2 has a bug. The meaning of odd is reversed for negative numbers.

$2E - Write binary integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Unused   |   Format   |
|            |   Number   |            |            |
|    $2E     |            |            |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove an integer from the stack and write it in big-endian binary format to a file of integer.

$2E - Write ASCII integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Field    |   Format   |
|            |   Number   |   Width    |            |
|    $2E     |            |            |    $01     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove an integer from the stack and write it in ASCII format to a file of char. If the formatted number, including a minus sign as needed, exceeds the field width, the field is filled with asterisks.

Note: Version 1.2 of the run-time system treats all values of Format other than zero as ASCII.

$2F - Read binary integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Unused   |   Format   |
|            |   Number   |            |            |
|    $2F     |            |            |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Read in integer in big-endian binary format from a file of integer and push it onto the stack.

$2F - Read ASCII integer

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    File    |   Unused   |   Format   |
|            |   Number   |            |            |
|    $2F     |            |            |    $01     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Read an integer in ASCII format from a file of char and push it onto the stack.

Note: Version 1.2 of the run-time system treats all values of Format other than zero as ASCII.

$30 - Push set

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $30     |            |            |            |
|            | %0000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Push a set onto the stack. The low order four bits of the nesting level specifies which stack frame contains the variable containing the set. The frame offset is the offset of the variable from the beginning of its stack frame.

$30 - Push set from array

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $30     |            |            |            |
|            | %1000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Push a set onto the stack. The integer originally on the top of the stack provides the offset from the base of the array of the element to push; it is removed first. The low order four bits of the nesting level specifies which stack frame contains the array containing the set. The frame offset is the offset of the array from the beginning of its stack frame.

$31 - Pop set

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $31     |            |            |            |
|            | %0000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Pop a set from the stack. The low order four bits of the nesting level specifies which stack frame contains the variable to receive the set. The frame offset is the offset of the variable from the beginning of its stack frame.

$31 - Pop set into array

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $31     |            |            |            |
|            | %1000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Pop a set from the stack. The integer originally on the top of the stack provides the offset from the base of the array to the element to receive the set; it is removed first. The low order four bits of the nesting level specifies which stack frame contains the array to receive the set. The frame offset is the offset of the array from the beginning of its stack frame.

$32 - Create empty set

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |            |
|            |            |            |            |
|    $32     |    $00     |            |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Create an empty set on the top of the stack.

$32 - Create set containing member

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Member   |
|            |            |            |   Number   |
|    $32     |    $00     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Create a set on the top of the stack containing one member. The member number must be in the range 1..64.

$32 - Add member to set

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Member   |
|            |            |            |   Number   |
|    $32     |    $01     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Add a member to the set on the top of the stack. The member number must be in the range 1..64.

$33 - Set union

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $33     |    $00     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove two sets on the top of the stack and replace them with the union of the sets.

$33 - Set difference

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $33     |    $01     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove two sets on the top of the stack and replace them with the intersection of the topmost set and the inverse of the second topmost one.

Note: Version 1.2 of the run-time system treats all positive values of mode other than zero as set difference.

$33 - Set Intersection

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Unused   |   Unused   |
|            |            |            |            |
|    $33     |    $80     |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove two sets on the top of the stack and replace them with the intersection of the sets.

Note: Version 1.2 of the run-time system treats all negative values of mode as set intersection.

$34 - Determine set membership

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $34     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Check the set at the top of the stack for the member specified by the byte below it; both are removed. If it is not a member, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$34 - Check set membership, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $34     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Check the set at the top of the stack for the member specified by the byte below it; both are removed. If it is not a member, program execution continues at the target address, otherwise continue with the following instruction.

$35 - Compare sets for =

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $35     |    $00     |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove and compare two sets at the top of the stack. If the sets are not equal, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$35 - Compare sets for =, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $35     |    $00     |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove and compare two sets at the top of the stack. If the sets are not equal, program execution continues at the target address, otherwise continue with the following instruction.

$35 - Compare sets for <>

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $35     |    $01     |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove and compare two sets at the top of the stack. If the sets are equal, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

Note: Version 1.2 of the run-time system treats all values of mode other than zero as compare for <>.

$35 - Compare sets for <>, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $35     |    $01     |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Remove and compare two sets at the top of the stack. If the sets are equal, program execution continues at the target address, otherwise continue with the following instruction.

Note: Version 1.2 of the run-time system treats all values of mode other than zero as compare for <>.

$36 - Compare ALFAs for =

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $36     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the ALFAs are not equal, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$36 - Compare ALFAs for =, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $36     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the ALFAs are not equal, program execution continues at the specified target address, otherwise continue with the following instruction.

$37 - Compare ALFAs for <>

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $37     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the ALFAs are equal, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$37 - Compare ALFAs for <>, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $37     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the ALFAs are equal, program execution continues at the specified target address, otherwise continue with the following instruction.

$38 - Compare ALFAs for <

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $38     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the second topmost ALFA is not less than the topmost one, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$38 - Compare ALFAs for <, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $38     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the, program execution continues at the specified target address, otherwise continue with the following instruction.

$39 - Compare ALFAs for >

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $39     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the second topmost ALFA is not greater than the topmost one, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$39 - Compare ALFAs for >, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $39     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the second topmost ALFA is not greater than the topmost one, program execution continues at the specified target address, otherwise continue with the following instruction.

$3A - Compare ALFAs for <=

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $3A     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the second topmost ALFA is not less than or equal to the topmost one, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$3A - Compare ALFAs for <=, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $3A     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the second topmost ALFA is not less than or equal to the topmost one, program execution continues at the specified target address, otherwise continue with the following instruction.

$3B - Compare ALFAs for >=

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $3B     |            |            |            |
|            |            |    $00     |    $00     |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the second topmost ALFA is not greater than or equal to the topmost one, $00 (False) is pushed onto the stack, otherwise $FF (True) is pushed. Execution continues with the following instruction.

$3B - Compare ALFAs for >=, if false then jump

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |   Unused   |   Target   |   Target   |
|            |            |  Address   |  Address   |
|    $3B     |            |            |            |
|            |            |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Compare two ALFAs at the top of the stack; they are removed from the stack in the process. If the second topmost ALFA is not greater than or equal to the topmost one, program execution continues at the specified target address, otherwise continue with the following instruction.

$3C - Push ALFA

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $3C     |            |            |            |
|            | %0000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Push an ALFA onto the stack. The low order four bits of the nesting level specifies which stack frame contains the variable containing the ALFA. The frame offset is the offset of the variable from the beginning of its stack frame.

$3C - Push ALFA from array

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $3C     |            |            |            |
|            | %1000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Push an ALFA onto the stack. The integer originally on the top of the stack provides the offset from the base of the array of the element to push; it is removed first. The low order four bits of the nesting level specifies which stack frame contains the array containing the ALFA. The frame offset is the offset of the array from the beginning of its stack frame.

$3D - Pop ALFA

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $3D     |            |            |            |
|            | %0000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Pop an ALFA from the stack. The low order four bits of the nesting level specifies which stack frame contains the variable to receive the ALFA. The frame offset is the offset of the variable from the beginning of its stack frame.

$3D - Pop ALFA into array

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |  Nesting   |   Frame    |   Frame    |
|            |   Level    |   Offset   |   Offset   |
|    $3D     |            |            |            |
|            | %1000bbbb  |    high    |    low     |
|            |            |    byte    |    byte    |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Pop an ALFA from the stack. The integer originally on the top of the stack provides the offset from the base of the array to the element to receive the ALFA; it is removed first. The low order four bits of the nesting level specifies which stack frame contains the array to receive the ALFA. The frame offset is the offset of the array from the beginning of its stack frame.

$3E - used by the compiler

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |  Unknown   |  Unknown   |
|            |            |            |            |
|    $3E     | %0bbbbbbb  |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Used by the compiler.

$3E - used by the compiler

Format:

```
+------------+------------+------------+------------+
|            |            |            |            |
|   Opcode   |    Mode    |  Unknown   |  Unknown   |
|            |            |            |            |
|    $3E     | %1bbbbbbb  |            |            |
|            |            |            |            |
+------------+------------+------------+------------+
```

Function:

Used by the compiler.

Appendix A - RUNEQU.TXT

```
 NAM RUNEQU
 TTL RUNEQU.TXT  V 1.2
*
* SYMBOL DEFINITIONS FOR P-6800 RUN-TIME SYSTEM
* NIGEL W.BENNEE MAY 79
*
 OPT PAG
NPAGE EQU 128 MAX CODE PAGES
PSIZE EQU 256 PAGE SIZE IN BYTES
NRPAG EQU 64 NUMBER OF RESIDENT PAGES
NFILE EQU 8 MAX NUMBER OF FILES
NDEV EQU 8 MAX NUMBER OF DEVICES
*
* THE FOLLOWING ARE ACTUAL ADDRESSES
*
ZREL EQU $FC FIRST FREE PAGE ZERO LOCATION
DEVTAB EQU $180 START OF DEVICE TABLE 0-7 BY 4
*
* THE FOLLOWING ARE ALL ADDRESSES OF POINTERS
*
DEVINT EQU $1A0 ADDRESS OF USER DEVICE INITL.
USRENT EQU $1A2 ADDRESS OF USER FUNCTION ENTRY
MARKUS EQU $1A4 USER BASE OF STACK POINTER
LIMIT EQU $1A8 HIGHEST AVAILABLE MEMORY ADDRESS
*
* THE FOLLOWING ARE ADDRESSES OF BYTES
*
TXLF EQU $1A6 CHAR TO SEND AFTER <CR> SENT
RXLF EQU $1A7 CHAR TO SEND AFTER <CR> RECIEVED
*
 END
```

Appendix B - OWNCODE.TXT

```
 NAM OWNCODE
 TTL OWNCODE.TXT  V 1.2
*
* EXAMPLE OF INTERFACING USER FUNCTION AND A
* DEVICE DRIVER TO THE P-6800 RUN-TIME SYSTEM
* DRIVER FOR MP/S ON PORT #2
* NIGEL W.BENNEE  MAY 79
*
*
 OPT PAG
*
MARKUS EQU $1A4
DEVTAB EQU $180
DEVINT EQU $1A0
USRENT EQU $1A2
*
* PUT ADDRESSES OF DRIVERS FOR PORT #2 ACIA IN DEVTAB
*
 ORG DEVTAB+8
 FDB INP2
 FDB OUT2
*
* PUT ADDRESS OF INITILISATION CODE
*
 ORG DEVINT
 FDB INIT
*
* PUT ADDRESS OF USER FUNCTION
*
 ORG USRENT
 FDB START
*
MPS EQU $8008 P0RT #2 ADDRESS
NREL EQU $A100 USE COMMAND SPACE FOR THIS CODE
*
* INITIALIZATION OF INTERFACES
*
 ORG NREL
INIT EQU *
 LDX #MPS INITIALIZE MPS
 LDA A #3 RESET CODE FOR 6850
 STA A 0,X
 LDA A #9 7 DATA BITS EVEN PARITY 1 STOP BIT
 STA A 0,X
 LDA A 0,X CLEAR
 LDA A 1,X
 RTS
*
* USER FUNCTION TO SHIFT FIRST INTEGER PARAMETER
* SECOND INTEGER PARAMETER TIMES TO THE LEFT
* NO CHECKING PERFORMED AND SHIFT COUNT ASSUMED
* LESS THAN 255.
*
START EQU *
 LDX MARKUS FIND OUT WHERE WE ARE
 LDA A 8,X MSB OF OPERAND
 LDA B 9,X LSB OF OPERAND
 TST 11,X MAKE SURE ITS NOT ZERO
 BEQ LOOP1
LOOP EQU *
 ASL B
 ROL A
 DEC 11,X DECREMENT COUNTER
 BNE LOOP
LOOP1 EQU *
 STA A 0,X INTEGER RESULT
 STA B 1,X
 RTS
*
* MPS DRIVER
*
INP2 EQU *
 LDX #MPS
 LDA B #1 RX DONE FLAG
INENT1 EQU *
 BIT B 0,X TEST STATUS
 BEQ INENT1 NOT GOT ANYTHING YET WAIT
 LDA A 1,X GET CHAR
 RTS
*
OUT2 EQU *
 LDX #MPS
 LDA B #2 TX EMPTY FLAG
OUTEN1 EQU *
 BIT B 0,X TEST STATUS
 BEQ OUTEN1 NOT FINISHED YET
 STA A 1,X SEND CHAR
 RTS
 END
```