PROGRAMMER'S CHALLENGE

by Bob Boonstra, Westford, MA

ROUTER RULES

This month's Challenge is based on a suggestion by Peter Lewis and is motivated by a real-world problem. A certain university has a B-class IP subnet, let's call it 199.232.*.* (with apologies to the real-world owner of that subnet). The subnet is broken down into 256 networks for the various faculties and departments, each one having 256 IP numbers. So, for example, the computer club might have 199.232.101.*. Our hypothetical university is charged for communications based on volume, so some of these networks are allowed to talk to the outside world, and others are not. Outside access is controlled by programming a router with a sequence of rules, each of which allows or denies access to some subset of IP numbers. A rule consists of a (mask, value, allow) triplet. For example, say the networks (in hex) 01, 03, 41, 43 are allowed out, and all the rest are barred. The rules could be simply:

```
FF, 01, allow
FF, 03, allow
FF, 41, allow
FF, 43, allow
00, 00, deny
```

But this could be simplified to:

```
BD, 01, allow
00, 00, deny
```

Your objective for this Challenge is to quickly generate a small sequence of rules that allows outside network access to only a specified set of networks. The prototype for the code you should write is:

```
enum {kDeny=0, kAllow=1};

typedef struct Rule {
  long mask;
  long value;
  long allow; /* 0 == deny, 1== allow */
} Rule;

long RouterRules(
  long allowedValues[],
  long numAllowedValues,
  long numBits,
  Rule rulesArray[],
  long maxRules
);
```

The array `allowedValues` is the set of `numAllowedValues` networks that are to be given outside network access. All other networks should be denied access. Instead of being limited to 8 bits as in the example above, network values have `numBits` bits. Your code should generate a sequence of rules that provides access to these networks, and no others. The rule sequence should be as short as possible and stored in `rulesArray`, which is allocated by the caller and is of size `maxRules`. Your code should return the number of rules generated, or return -1 if it cannot find a solution no longer than `maxRules`.

Rules will be triggered by the router in the order provided by your solution, and the first rule to fire for a given network will apply. At least one rule must fire for any possible network value. For example, if `numBits==3`, and we want to allow access to networks 0, 2, 3, 6, and 7, you could use the following rules:

```
3, 1, deny
6, 4, deny
7, 7, allow
```

To encourage code that generates both fast and short solutions, the ranking will be based on minimizing the following function of execution time on my 8500/150 and the number of rules generated:

score = (number of rules generated) + (execution time in seconds) / 2

This will be a native PowerPC Challenge, using the latest CodeWarrior environment. Solutions may be coded in C, C++, or Pascal.

TWO MONTHS AGO WINNER

Congratulations to **Xan Gregg** (Durham, N.C.), for submitting the fastest entry to the ByteCode Interpreter Programmer's Challenge, narrowly beating out the second-place entry by **Ernst Munter**. The Challenge was to write an interpreter for a subset of the byte code language implemented by the Java Virtual Machine. The Challenge rules pointed to the Java Virtual Machine Specification for a description of the opcodes, with some exclusions about the opcodes and features that were to be implemented, and with the significant simplifying assumption that the Virtual Machine need only deal with a single class file. Of the five solutions submitted, three worked correctly for all test cases, one worked for all but one test case, and the fifth was acknowledged by the author to be incomplete.

The rules for September permitted the use of assembly language, and Xan was the only contestant to submit a solution that took advantage of this. After parsing the header to identify the constants, fields, and methods contained in the class file, the solution dispatches and executes each opcode. As described by the comments in the code, the main execution loop contains a table with 32 bytes of PowerPC instructions implementing each opcode. Opcodes that require more code than will fit into the table entry overflow to code outside the table. Particular features that you might want to examine in the code include the implementation of the jump table and the pseudo-opcode ExitCode used to trigger a return to the calling routine. Congratulations to Xan on an elegant, efficient, and instructive solution. Several readers commented that they learned quite a bit about Java from implementing a Virtual Machine. Congratulations as well to everyone who participated in this more difficult than usual Challenge!

The table below summarizes the results for each entry, including execution time in milliseconds, code size, and data size. An asterisk indicates a test case that was not successfully completed by a solution. Numbers in parenthesis after a person's name indicate that person's cumulative point total for all previous Challenges, not including this one.

Name	Language	Test1	Test2	Test3	Test4	Time	Code	Data
Xan Gregg (92)	C/Assembly	31088	12923	24607	45190	113808	10064	171
Ernst Munter (214)	C++	18661	14664	28260	52496	114081	6260	879
Turlough O'Connor	C++	23073	15946	30503	57444	126967	14536	5818
Conor MacNeill	C++	44446	*	43298	84020	*	82536	10909

The test code was contained in standard Java applets, which allowed me to use the interpreters supplied with the Symantec and Metrowerks environments to confirm expected results. This also

allowed a comparison of the execution time of the mini Virtual Machines submitted as solutions with the execution time of the commercial interpreters. Results for the same four test cases used to score the solutions are presented below for the Applet Viewer provided with Symantec Cafe (version 1.0) and for the Metrowerks Java interpreter provided with CodeWarrior 9. While the comparison is not entirely fair because of the simplifying assumptions used in this Challenge, the table indicates that three of the four solutions were faster than both of these interpreters. Although CodeWarrior 10 has not been finalized as this column is being written, it should be available at publication, and Metrowerks was kind enough to give me a preview of the Java interpreters available in that release. For my limited set of test cases, the CW10 version of Metrowerks Java was approximately 25% faster than the CW9 version. Even more impressive was the Just-In-Time version of the interpreter, which (I assume) first compiles the byte-coded instructions into PowerPC instructions before execution. My tests suggest that Metrowerks Java JIT executes an order of magnitude faster than the CW9 interpreter (not counting the preprocessing compilation time, which my tests did not measure).

Product	Test1	Test2	Test3	Test4	Time
Symantec Cafe 1.0	41000	34000	62000	134000	271000
Metrowerks Java (CW9)	24175	18714	35075	68077	146041
Metrowerks Java (CW10 preview)	18336	14253	27494	52285	112368
Metrowerks Java JIT (CW10 preview)	5487	1549	2560	5977	15573

TOP 20 CONTESTANTS

Here are the Top 20 Contestants for the Programmer's Challenge. The numbers below include points awarded over the 24 most recent contests, including points earned by this month's entrants.

Rank	Name	Points	Rank	Name	Points
1.	Munter, Ernst	193	11.	Cutts, Kevin	21
2.	Gregg, Xan	112	12.	Picao, Miguel Cruz	21
3.	Larsson, Gustav	87	13.	Brown, Jorg	20
4.	Lengyel, Eric	40	14.	Gundrum, Eric	20
5.	[Name deleted]	40	15.	Karsh, Bill	19
6.	Lewis, Peter	32	16.	Stenger, Allen	19
7.	Boring, Randy	27	17.	Cooper, Greg	17
8.	Beith, Gary	24	18.	Mallett, Jeff	17
9.	Kasparian, Raffi	22	19.	Nevard, John	17
10.	Vineyard, Jeremy	22	20.	Nicolle, Ludovic	14

There are three ways to earn points: (1) scoring in the top 5 of any Challenge, (2) being the first person to find a bug in a published winning solution or, (3) being the first person to suggest a Challenge that I use. The points you can win are:

1st place	20 points	5th place	2 points
2nd place	10 points	finding bug	2 points
3rd place	7 points	suggesting Challenge	2 points
4th place	4 points		

Here is Xan's winning solution:

JAVAMINIVM.C


```
/*
The core of the interpreter is written in mostly PowerPC
assember, and the parsing of the tables is done in C
before interpreting begins.  I create tables of relevant
data for constants, fields, and methods.  The constants
data is just the address of the Java constant_pool data for
that index.  For fields and methods I allocate structs
with the useful data and insert a pointer to the struct
into the field's or method's constant_pool entry.

So, for some up-front overhead and memory usage (16 bytes
per field or method), the interpreter has less work to
do when dealing with a field or method.

String objects are pointers to the constant_pool
payload, just as the provided test code expects.

Objects in general are greatly simplified because of the
one class limitation.  No type information is stored with
any object in the heap.

Of the 10M of provided heapSpace, I use 512K for the Java
heap, 512K for the Java stacks (data and return), and
the rest is available for the constant, field, and method
tables, which will consume less that 20 * NumConstants
bytes.

The heap size and stack size can be set by the macros
HEAP_K and STACK_K below.  No garbage collection is
performed on the heap, so if there is lots of object
creation, a larger heap may be needed.

The VM has two stacks, a data stack and a return stack.
The data stack grows up from the start of the stack space,
and the return stack grows down from the end of the
stack space.  The data stack is used for parameters and
normal Java stack operations.  The return stack is used
method nesting.  Each call makes a three-entry stack frame
consisting of the return address, frame pointer (start of
locals), and tsBase (the method base for use by tableswitch
and lookupswitch).

Having separate stacks made the parameter passing easier,
but now I realize that with some extra work, I could have
just put the return frame into added local variables.

The Top-Of-Stack is kept in a register.  This requires
a little handshaking when the interpreter calls another
function, which doesn't have access to the TOS register.

The core of the interpreter is in the assembly routine,
StartVM.  Its main loop fetches and dispatches opcodes,
as expected.  It includes a table with a 32-byte entry
for each opcode (up through 209).  An entry consists
of code to implement the opcode.  Most of them fit in
32-bytes, and any extra space is used to prime the
dispatch loop.
```

Opcodes that take more than 7 instructions to implement
can spill over to code outside of the table. Opcodes
that are particularly complex (like multianewarray)
are implemented with C functions.

Limitations:
I try to safely ignore operations involving long, double,
and float operands by treating these types as taking 0
bytes each. Most instructions become a NOP, and others
are very simple (i2d become a POP). However, I have
realized one flaw to this system: The untyped instructions
(like DUP) may try to operate on the unsupported types.
It's too late to fix now, but since the challenge will try
to avoid unsupported types, it's unlikely the stack
manipulation of them will be needed.

Future:
With the PowerPC code done for each opcode, it would
not be too tough to make a compiler that would string
together the PowerPC code for each opcode in each method.
*/

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Java types
typedef signed char s1;
typedef signed short s2;
typedef signed long s4;
typedef unsigned char u1;
typedef unsigned short u2;
typedef unsigned long u4;

// Set these if necessary
#define HEAP_K 512
#define STACK_K  512

// the CONSTANT_ id's
enum {C_Utf8 = 1, C_Unicode, C_Integer, C_Float, C_Long,
   C_Double, C_Class, C_String, C_Fieldref,
   C_Methodref, C_InterfaceMethodref, C_NameAndType};

const u2 ACC_STATIC = 0x0008;

// The contents of my tables
typedef u1 *ConstantData;

typedef struct
{
  u2   *fieldInfoP;
  long offset;   // in bytes
  long size;       // in bytes
  u1   *type;        // ptr to sig constant
} FieldData;
```

```
typedef struct
{
  u2   *methodInfoP;
  u2   *codeAttrP;
  u1   *codeP;
  u4   paramCount;   // in bytes
} MethodData;

// prototypes
void JavaMiniVM(void *constant_pool, void *fields,
  void *methods, void *classFile,
  long methodToExecute, void *heapSpace,void *returnStack);

static short IndexConstants(void *constants,
            ConstantData *indexArray);
static short IndexFields(void *fields);
static short IndexMethods(void *methods);
static void CreateAndPush(void);
static void AllocateStaticFields(void);
static void ResolveFields(void);
static void ResolveMethods(void);
static void CountParams(MethodData *methodP);
asm static long StartVM(register MethodData *methodP);
static void PushConstant(long n);
static u1 *TableSwitch(u1 *ip, long tsBase, long n);
static u1 *LookupSwitch(u1 *ip, long tsBase, long key);
static long GetNewArray(long type, long size);
static long GetANewArray(long size);
static void PushMultiANewArray(long classIndex,
            long numDimensions);

// globals
static long    NumMethods;
static long    NumFields;
static long    NumConstants;
static MethodData *Methods;
static FieldData *Fields;
static ConstantData *Constants;
static long    *FP; // frame pointer
static long    *SP; // stack pointer
static long    *S0; // stack base
static long    *RP; // return stack pointer
static long    *R0; // return stack base
static long    *HP; // heap pointer
static long    *H0; // heap base
static long    TotalStatic;          // in bytes
static long    TotalNonStatic;       // in bytes
static long    LastField;
static long    LastMethod;

/* java types
typedef struct
{
0 u2   attribute_name;
2 u4   attribute_length;
6 u1   info[1];
} attribute_info;
```

```
typedef struct
{
0 u2   access_flags;
2 u2   name_index;
4 u2   signature_index;
6 u2   attribute_count;
8 attribute_info   attributes[1];
} field_info;

typedef struct
{
0 u2   access_flags;
2 u2   name_index;
4 u2   signature_index;
6 u2   attribute_count;
8 attribute_info   attributes[1];
} method_info;

typedef struct
{
0 u1   tag;
1 u2   length;
3 u1   bytes[1];
} CONSTANT_Utf8_info
*/

#define PUSH(a)   *SP++ = (a)

#define MethodIsStatic(n) \
  ((*Methods[n].methodInfoP & ACC_STATIC) != 0)
```

JavaMiniVM

```
void JavaMiniVM(void *constant_pool, void *fields,
  void *methods, void *classFile,
  long methodToExecute, void *heapSpace,void *returnStack)
{
  //allocate java heap
  H0 = (long *) heapSpace;
  HP = H0;
  // allocate java stack (stack grows up in memory)
  S0 = H0 + HEAP_K * 256L;  // 256 longs = 1K
  SP = S0;

  // return stack based at end of stack space (grows down)
  R0 = S0 + STACK_K * 256L; // 256 longs = 1K
  RP = R0;

  Constants = (ConstantData *) R0;
  NumConstants = IndexConstants(constant_pool, Constants);

  Fields = (FieldData *) (Constants + NumConstants);
  NumFields = IndexFields(fields);
  Methods = (MethodData *) (Fields + NumFields);
  NumMethods = IndexMethods(methods);

  ResolveFields();
  ResolveMethods();
```

```
  AllocateStaticFields();
  if (!MethodIsStatic(methodToExecute))
    CreateAndPush(); // so it needs an obj
  *(long *) returnStack =
     StartVM(&Methods[methodToExecute]);

}
```

IndexConstants

```
// create on index into the constant_pool
static short IndexConstants(void *constants,
            ConstantData *indexArray)
{
  short n, i;
  u1   *p;

  // list is preceded by length
  n = * ((u2 *) constants - 1);
  p = (u1 *) constants;
  LastField = 0;
  LastMethod = 0;
  for (i = 1; i < n; i++)
  {
    indexArray[i] = p;
    switch (*p)
    {
      case C_Utf8:  p += 3 + *(u2*)(p+1); break;
      case C_Unicode:  p += 3 + *(u2*)(p+1); break;
      case C_Integer:  p += 5; break;
      case C_Float: p += 5; break;
      case C_Long:  p += 9; i += 1; break;
      case C_Double: p += 9; i += 1; break;
      case C_Class: p += 3; break;
      case C_String: p += 3; break;
      case C_Fieldref: p += 5; LastField = i; break;
      case C_Methodref: p += 5; LastMethod = i; break;
      case C_InterfaceMethodref: p += 5; break;
      case C_NameAndType: p += 5; break;
    }
  }
  indexArray[0] = p;  //useful for getting to 'this_class'
  return n;
}
```

IndexFields

```
// Gather FieldData about each field
static short IndexFields(void *fields)
{
  short n, i;
  u1   *p;
  u1   *sigP;
  u2   sigIndex;
  FieldData  *fieldP;
  u1   c;
  long staticOffset = 0;
  long memberOffset = 0;
```

```
  // list is preceded by length
  n = * ((u2 *) fields - 1);
  p = (u1 *) fields;
  fieldP = Fields;
  for (i = 0; i < n; i++)
  {
    short count;  // attribute count

    fieldP->fieldInfoP = (u2 *) p;
    sigIndex = *(u2 *) (p + 4);
    sigP = Constants[sigIndex];
    c = *(sigP + 3);
    if (c == 'L' || c == 'D' || c == 'F')
      fieldP->size = 0;
    else
      fieldP->size = 4;
    fieldP->type = sigP;
    if ((*fieldP->fieldInfoP & ACC_STATIC) != 0)
    {
      fieldP->offset = staticOffset;
      staticOffset += fieldP->size;
    }
    else
    {
      fieldP->offset = memberOffset;
      memberOffset += fieldP->size;
    }
    // skip attributes
    count = * (u2 *) (p + 6);
    p += 8;
    while (count > 0)
    {
      count -= 1;
      p += 6 + *(u4 *) (p+2);
    }
    fieldP += 1;
  }
  TotalStatic = staticOffset;
  TotalNonStatic = memberOffset;
  return n;
}
```

IndexMethods

```
// Gather MethodData about each method
static short IndexMethods(void *methods)
{
  short n, i;
  u1   *p;
  MethodData *methodP;

  // list is preceded by length
  n = * ((u2 *) methods - 1);
  p = (u1 *) methods;
  methodP = Methods;
  for (i = 0; i < n; i++)
  {
    short count;  // attribute count
```

```
    u2   nameIndex;
    u1   *nameEntryP;

    methodP->methodInfoP = (u2 *) p;
    CountParams(methodP);
    count = * (u2 *) (p + 6);
    p += 8;
    while (count > 0)
    {
      nameIndex = * (u2 *) (p);
      nameEntryP = Constants[nameIndex];
      if (*(u2*)(nameEntryP + 1) == 4
         && *(long *) (nameEntryP + 3) == 'Code')
      { // this is the code attr
        methodP->codeAttrP = (u2 *) p;
        methodP->codeP = (u1 *) (p + 14);
      }
      count -= 1;
      p += 6 + *(u4 *) (p+2);
    }
    methodP += 1;
  }
  return n;
}
```

CreateAndPush

```
// allocate 'this' and push a reference to it
static void CreateAndPush(void)
{
  *SP = (long) HP;
  SP ++;
  // heap already initialized to zeros
  HP = (long *) ((long) HP + TotalNonStatic);
}
```

AllocateStaticFields

```
static void AllocateStaticFields(void)
{
  // heap already initialized to zeros
  HP = (long *) ((long) HP + TotalStatic);
}
```

ResolveFields

```
// Change the payload of CONSTANT_Fieldref items to be a
// pointer into the FieldData array
static void ResolveFields(void)
{
  short i, j;
  u1   *p;
  u1   *q;
  ConstantData  *constantP;
  FieldData  *fieldP;
  u2   nameTypeIndex;
  u2   nameIndex;

  constantP = Constants+1;
  for (i = 1; i <= LastField; i++)
```

```
  {
    p = *constantP++;
    if (*p == C_Fieldref)
    {
      fieldP = Fields;
      nameTypeIndex = *(u2*) (p + 3);
      q = Constants[nameTypeIndex];
      nameIndex = *(u2*) (q + 1);
      for (j = 0; j < NumFields; j++)
      {
        if (*(fieldP->fieldInfoP+1) == nameIndex)
        { // found matching field ref, change it
          *(u4*) (p+1) = (u4) fieldP;
          break;
        }
        fieldP += 1;
      }
    }
  }
}
```

ResolveMethods

```
// Change the payload of CONSTANT_Methodref items to be a
// pointer into the MethodData array
static void ResolveMethods(void)
{
  short i, j;
  u1   *p;
  u1   *q;
  ConstantData  *constantP;
  MethodData *methodP;
  u2   nameTypeIndex;
  u2   nameIndex;
  u2   sigIndex;

  constantP = Constants+1;
  for (i = 1; i <= LastMethod; i++)
  {
    p = *constantP++;
    if (*p == C_Methodref)
    {
      methodP = Methods;
      nameTypeIndex = *(u2*) (p + 3);
      q = Constants[nameTypeIndex];
      nameIndex = *(u2*) (q + 1);
      sigIndex = *(u2*) (q + 3);
      for (j = 0; j < NumMethods; j++)
      {
        if (methodP->methodInfoP[1] == nameIndex
          && methodP->methodInfoP[2] == sigIndex)
        { // found matching method ref, change it
          *(u4*) (p+1) = (u4) methodP;
          break;
        }
        methodP += 1;
      }
    }
```

```
  }
}
```

CountParams

```
static void CountParams(MethodData *methodP)
{
  long sigIndex;
  long paramCount;
  u1   *sigPtr;
  u2   sigLength;
  u1   ch;

  sigIndex = *(methodP->methodInfoP + 2);
  sigPtr = Constants[sigIndex];
  sigLength = * (u2 *) (sigPtr + 1);
  sigPtr += 4;  // skip tag, length and '('
  if ((*methodP->methodInfoP & ACC_STATIC) == 0)
    paramCount = 4;  // implicit class object parameter
  else
    paramCount = 0;
  while (1)
  {
    ch = *sigPtr;
    if (ch == ')')
      break;
    paramCount += 4;
    if (ch == 'D' || ch == 'J' || ch == 'F')
      paramCount -= 4; // these are 0-byte types
    else if (ch == 'L')
    { // skip class name
      while (*++sigPtr != ';')
        ;
    }
    else if (ch == '[')
    { // we don't care what it's an array of
      while (*++sigPtr == '[')
        ;
      if (*sigPtr == 'L')
        while (*++sigPtr != ';')
          ;
    }
    sigPtr += 1;
  }
  ch = *++sigPtr;
  methodP->paramCount = paramCount;
}

#define slwi(r, n)       \
  rlwinm  r, r, n, 0, 31-n;

#define times4(r)      \
  rlwinm  r, r, 2, 0, 29;

// the number after 'used' indicates the number of
// instructions used so far in this slot.
#define used0       \
  lbz   opcode, 0(ip);   \
```

```
  slwi(opcode, 5)      \
  add   a, base, opcode; \
  mtctr a;       \
  addi ip, ip, 1;    \
  bctr;        \
  nop; nop;

#define used1       \
  lbz   opcode, 0(ip);   \
  slwi(opcode, 5)      \
  add   a, base, opcode; \
  mtctr a;       \
  addi ip, ip, 1;    \
  bctr;        \
  nop;

#define used2       \
  lbz   opcode, 0(ip);   \
  slwi(opcode, 5)      \
  add   a, base, opcode; \
  mtctr a;       \
  addi ip, ip, 1;    \
  bctr;

#define used3       \
  lbz   opcode, 0(ip);   \
  slwi(opcode, 5)      \
  add   a, base, opcode; \
  mtctr a;       \
  b  next4;

#define used4       \
  lbz   opcode, 0(ip);   \
  slwi(opcode, 5)  \
  add   a, base, opcode; \
  b  next3;

#define used5       \
  lbz   opcode, 0(ip);   \
  slwi(opcode, 5)      \
  b  next2;

#define used6       \
  lbz   opcode, 0(ip);   \
  b  next1;

#define used7       \
  b  next0;

#define pushtos   \
  stw   tos, 0(sp); \
  addi sp, sp, 4;

#define poptos   \
  lwzu tos, -4(sp)

#define pushi(n)  \
```

```
  stw   tos, 0(sp); \
  li   tos, n;  \
  addi sp, sp, 4;

#define pushr(r)  \
  stw   tos, 0(sp); \
  mr   tos, r;  \
  addi sp, sp, 4;
```

// The initial return address points here.

```
u1 ExitCode[1] = {203};
```

StartVM

```
asm static long StartVM(register MethodData *methodP)
{
  register long *sp;
  register long *rp;
  register long *fp;
  register long *h0;
  register long *cp; // Constants
  register u1   *ip;   // instruction ptr
  register u1   opcode;
  register long tsBase; // for tableswitch padding
  register long a;
  register long b;
  register long c;
  register long base;  // base of our opcode table
  register long tos;

  fralloc
  lwz   sp, SP
  lwz   rp, RP
  lwz   h0, H0
  lwz   cp, Constants
  lwz   a, ExitCode
  stwu a, -4(rp)      // push initial return ip
  lwzu tos, -4(sp)    // put TOS into a reg.

  // get address of opcode table
  bl   nowhere
nowhere:
  mflr base
  // offset hard-coded since MW won't subtract labels
  addi base, base, 22*4 //table - nowhere

start_method:
  stwu fp, -4(rp)

  pushtos // put tos (last param) in memory
  mr   fp, sp
  // FP points to first parameter
  lwz   a, methodP->paramCount
  subf fp, a, fp
  // allocate space for locals
  lwz   b, methodP->codeAttrP
  lhz   c, 8(b)   // max locals
  times4(c)
```

```
  subf c, a, c
  add   sp, sp, c
  // init ip
  lwz   ip, methodP->codeP
  // init tsBase
  andi. tsBase, ip, 3
  stwu tsBase, -4(rp)

next0:
  lbz   opcode, 0(ip)
next1:
  slwi(opcode, 5)
next2:
  add   a, base, opcode
next3:
  mtctr a
next4:
  addi ip, ip, 1
  bctr

table:
  // 0 nop
  used0
  // 1 aconst_null
  pushi(0)
  used3
  // 2 iconst_m1
  pushi(-1)
  used3
  // 3 iconst_0
  pushi(0)
  used3
  // 4 iconst_1
  pushi(1)
  used3
  // 5 iconst_2
  pushi(2)
  used3
  // 6 iconst_3
  pushi(3)
  used3
  // 7 iconst_4
  pushi(4)
  used3
  // 8 iconst_5
  pushi(5)
  used3
  // 9 lconst_0
  used0
  // 10 lconst_1
  used0
  // 11 fconst_0
  used0
  // 12 fconst_1
  used0
  // 13 fconst_2
  used0
  // 14 dconst_0
```

```
used0
// 15 dconst_1
used0
// 16 bipush
lbz    a, 0(ip)
extsb a, a
addi ip, ip, 1
pushr(a)
used6
// 17 sipush
lha    a, 0(ip)
addi ip, ip, 2
pushr(a)
used5
// 18 ldc1
lbz    r3, 0(ip)
addi ip, ip, 1
pushtos
b ldc_more
nop;nop;nop
// 19 ldc2
lhz    r3, 0(ip)
addi ip, ip, 2
pushtos
b ldc_more
nop;nop;nop
// 20 ldc2w
lhz    r3, 0(ip)
addi ip, ip, 2
pushtos
b ldc_more
nop;nop;nop
// 21 iload
lbz    a, 0(ip)
addi ip, ip, 1
times4(a)     // convert index to offset
pushtos
lwzx tos, a, fp
used6
// 22 lload
used0
// 23 fload
used0
// 24 dload
used0
// 25 aload
lbz    a, 0(ip)
addi ip, ip, 1
times4(a)     // convert index to offset
pushtos
lwzx tos, a, fp
used6
// 26 iload_0
pushtos
lwz    tos, 0(fp)
used3
// 27 iload_1
pushtos
```

```
lwz   tos, 4(fp)
used3
// 28 iload_2
pushtos
lwz   tos, 8(fp)
used3
// 29 iload_3
pushtos
lwz   tos, 12(fp)
used3
// 30 lload_0
used0
// 31 lload_1
used0
// 32 lload_2
used0
// 33 lload_3
used0
// 34 fload_0
used0
// 35 fload_1
used0
// 36 fload_2
used0
// 37 fload_3
used0
// 38 dlload_0
used0
// 39 dload_1
used0
// 40 dload_2
used0
// 41 dload_3
used0
// 42 aload_0
pushtos
lwz   tos, 0(fp)
used3
// 43 aload_1
pushtos
lwz   tos, 4(fp)
used3
// 44 aload_2
pushtos
lwz   tos, 8(fp)
used3
// 45 aload_3
pushtos
lwz   tos, 12(fp)
used3
// 46 iaload
lwzu b, -4(sp)  // get array address
mr   a, tos
addi b, b, 4
times4(a)          // convert index to offset
lwzx tos, a, b
used5
// 47 laload
```

```
lwzu tos, -8(sp)  // drop2
used1
// 48 faload
lwzu tos, -8(sp)  // drop2
used1
// 49 daload
lwzu tos, -8(sp)  // drop2
used1
// 50 aaload
lwzu b, -4(sp)   // get array address
mr   a, tos
addi b, b, 4
times4(a)          // convert index to offset
lwzx tos, a, b
used5
// 51 baload
lwzu b, -4(sp)   // get array address
addi b, b, 4
lbzx tos, tos, b
extsb tos, tos
used4
// 52 caload
lwzu b, -4(sp)   // get array address
mr   a, tos
addi b, b, 4
slwi(a, 1)       // convert index to offset
lhzx tos, a, b
used5
// 53 saload
lwzu b, -4(sp)   // get array address
mr   a, tos
addi b, b, 4
slwi(a, 1)       // convert index to offset
lhax tos, a, b
used5
// 54 istore
lbz    a, 0(ip)
addi ip, ip, 1
times4(a)          // convert index to offset
stwx tos, a, fp
lwzu tos, -4(sp)
used5
// 55 lstore
used0
// 56 fstore
used0
// 57 dstore
used0
// 58 astore
lbz    a, 0(ip)
addi ip, ip, 1
times4(a)          // convert index to offset
stwx tos, a, fp
lwzu tos, -4(sp)
used5
// 59 istore_0
stw    tos, 0(fp)
```

```
lwzu tos, -4(sp)
used2
// 60 istore_1
stw   tos, 4(fp)
lwzu tos, -4(sp)
used2
// 61 istore_2
stw   tos, 8(fp)
lwzu tos, -4(sp)
used2
// 62 istore_3
stw   tos, 12(fp)
lwzu tos, -4(sp)
used2
// 63 lstore_0
used0
// 64 lstore_1
used0
// 65 lstore_2
used0
// 66 lstore_3
used0
// 67 fstore_0
used0
// 68 fstore_1
used0
// 69 fstore_2
used0
// 70 fstore_3
used0
// 71 dstore_0
used0
// 72 dstore_1
used0
// 73 dstore_2
used0
// 74 dstore_3
used0
// 75 astore_0
stw   tos, 0(fp)
lwzu tos, -4(sp)
used2
// 76 astore_1
stw   tos, 4(fp)
lwzu tos, -4(sp)
used2
// 77 astore_2
stw   tos, 8(fp)
lwzu tos, -4(sp)
used2
// 78 astore_3
stw   tos,12(fp)
lwzu tos, -4(sp)
used2
// 79 iastore
lwzu b, -4(sp)  // index
lwzu c, -4(sp)  // array address
times4(b)          // index to offset
```

```
addi c, c, 4   // skip arraylength field
stwx tos, b, c
lwzu tos, -4(sp)
used6
// 80 lastore
lwzu tos, -8(sp)  // drop2
used1
// 81 fastore
lwzu tos, -8(sp)  // drop2
used1
// 82 dastore
lwzu tos, -8(sp)  // drop2
used1
// 83 aastore
lwzu b, -4(sp)   // index
lwzu c, -4(sp)   // array address
times4(b)          // index to offset
addi c, c, 4   // skip arraylength field
stwx tos, b, c
lwzu tos, -4(sp)
used6
// 84 bastore
lwzu b, -4(sp)   // index
lwzu c, -4(sp)   // array address
addi c, c, 4   // skip arraylength field
stbx tos, b, c
lwzu tos, -4(sp)
used5
// 85 castore
lwzu b, -4(sp)   // index
lwzu c, -4(sp)   // array address
slwi(b, 1)       // index to offset
addi c, c, 4   // skip arraylength field
sthx tos, b, c
lwzu tos, -4(sp)
used6
// 86 sastore
lwzu b, -4(sp)   // index
lwzu c, -4(sp)   // array address
slwi(b, 1)       // index to offset
addi c, c, 4   // skip arraylength field
sthx tos, b, c
lwzu tos, -4(sp)
used6
// 87 pop
lwzu tos, -4(sp)
used1
// 88 pop2
lwzu tos, -8(sp)
used1
// 89 dup
pushtos
used2
// 90 dup_x1
lwz    a, -4(sp)
stw    tos, -4(sp)
stw    a, 0(sp)
```

```
  addi sp, sp, 4
  used4
  // 91 dup_x2
  lwz   a, -8(sp)
  lwz   b, -4(sp)
  stw   tos, -8(sp)
  stw   a, -4(sp)
  stw   b, 0(sp)
  addi sp, sp, 4
  used6
  // 92 dup2
  lwz   a, -4(sp)
  stw   tos, 0(sp)
  stw   a, 4(sp)
  addi sp, sp, 8
  used4
  // 93 dup2_x1
  lwz   a, -8(sp)
  lwz   b, -4(sp)
  stw   b, -8(sp)
  stw   tos, -4(sp)
  stw   a, 0(sp)
  stw   b, 4(sp)
  addi sp, sp, 8
  used7
  // 94 dup2_x2
  lwz   a, -12(sp)
  lwz   b, -8(sp)
  lwz   c, -4(sp)
  stw   c, -12(sp)
  stw   tos, -8(sp)
  stw   a, -4(sp)
  stw   b, 0(sp)
  b  dup2_x2_more
  // 95 swap
  lwz   a, -4(sp)
  stw   tos, -4(sp)
  mr   tos, a
  used3
  // 96 iadd
  lwzu a, -4(sp)
  add   tos, a, tos
  used2
  // 97 ladd
  used6      // pretend we had 6 instructions already
  // stash the rest of dup2_x2 here
dup2_x2_more:
  stw   c, 4(sp)
  addi sp, sp, 8
  used4
  // 98 fadd
  used0
  // 99 dadd
  used0
  // 100 isub
  lwzu a, -4(sp)
  subf tos, tos, a
  used2
```

```
// 101 lsub
used0
// 102 fsub
used0
// 103 dsub
used0
// 104 imul
lwzu a, -4(sp)
mullw tos, tos, a
used2
// 105 lmul
used0
// 106 fmul
used0
// 107 dmul
used0
// 108 idiv
lwzu a, -4(sp)
divw tos, a, tos
used2
// 109 ldiv
used0
// 110 fdiv
used0
// 111 ddiv
used0
// 112 irem
lwzu a, -4(sp)
divw b, a, tos
mullw c, b, tos
subf tos, c, a
used4
// 113 lrem
used0
// 114 frem
used0
// 115 drem
used0
// 116 ineg
neg   tos, tos
used1
// 117 lneg
used0
// 118 fneg
used0
// 119 dneg
used0
// 120 ishl
lwzu a, -4(sp)
andi. tos, tos, 31
slw   tos, a, tos
used3
// 121 lshl
lwzu tos, -4(sp)
used1
// 122 ishr
lwzu a, -4(sp)
andi. tos, tos, 31
```

```
sraw tos, a, tos
used3
// 123 lshr
lwzu tos, -4(sp)
used1
// 124 iushr
lwzu a, -4(sp)
andi. tos, tos, 31
srw   tos, a, tos
used3
// 125 lushr
lwzu tos, -4(sp)
used1
// 126 iand
lwzu a, -4(sp)
and   tos, tos, a
used2
// 127 land
used0
// 128 ior
lwzu a, -4(sp)
or   tos, tos, a
used2
// 129 lor
used0
// 130 ixor
lwzu a, -4(sp)
xor   tos, tos, a
used2
// 131 lxor
used0
// 132 iinc
lhz   b, 0(ip)
rlwinm  a, b, 26, 16, 29
extsb b, b
addi ip, ip, 2
lwzx c, fp, a
add   c, c, b
stwx c, fp, a
used7
// 133 i2l
lwzu tos, -4(sp)
used1
// 134 i2f
lwzu tos, -4(sp)
used1
// 135 i2d
lwzu tos, -4(sp)
used1
// 136 l2i
pushi(0)
used3
// 137 l2f
used0
// 138 l2d
used0
// 139 f2i
pushi(0)
```

```
used3
// 140 f2f
used0
// 141 f2d
used0
// 142 d2i
pushi(0)
used3
// 143 d2f
used0
// 144 d2d
used0
// 145 int2byte
extsb tos, tos
used1
// 146 int2char
andi. tos, tos, 0xffff
used1
// 147 int2short
extsh tos, tos
used1
// 148 lcmp
pushi(0)
used3
// 149 fcmpl
pushi(0)
used3
// 150 fcmpg
pushi(0)
used3
// 151 dcmpl
pushi(0)
used3
// 152 dcmpg
pushi(0)
used3
// 153 ifeq
cmpi cr0, 0, tos, 0
lha   a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
subi a, a, 3
bne   next0
add   ip, ip, a
used7
// 154 ifne
cmpi cr0, 0, tos, 0
lha   a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
subi a, a, 3
beq   next0
add   ip, ip, a
used7
// 155 iflt
cmpi cr0, 0, tos, 0
lha   a, 0(ip)
lwzu tos, -4(sp)
```

```
  addi ip, ip, 2
  subi a, a, 3
  bge   next0
  add   ip, ip, a
  used7
  // 156 ifge
  cmpi cr0, 0, tos, 0
  lha   a, 0(ip)
  lwzu tos, -4(sp)
  addi ip, ip, 2
  subi a, a, 3
  blt   next0
  add   ip, ip, a
  used7
  // 157 ifgt
  cmpi cr0, 0, tos, 0
  lha   a, 0(ip)
  lwzu tos, -4(sp)
  addi ip, ip, 2
  subi a, a, 3
  ble   next0
  add   ip, ip, a
  used7
  // 158 ifle
  cmpi cr0, 0, tos, 0
  lha   a, 0(ip)
  lwzu tos, -4(sp)
  addi ip, ip, 2
  bgt   next0
do_jump: // shared by cmp instructions
  subi a, a, 3
  add   ip, ip, a
  used7
  // 159 if_icmpeq
  lwz   b, -4(sp)
  cmp   cr0, 0, b, tos
  lha   a, 0(ip)
  lwzu tos, -8(sp)
  addi ip, ip, 2
  beq   do_jump
  used6
  // 160 if_icmpne
  lwz   b, -4(sp)
  cmp   cr0, 0, b, tos
  lha   a, 0(ip)
  lwzu tos, -8(sp)
  addi ip, ip, 2
  bne   do_jump
  used6
  // 161 if_icmplt
  lwz   b, -4(sp)
  cmp   cr0, 0, b, tos
  lha   a, 0(ip)
  lwzu tos, -8(sp)
  addi ip, ip, 2
  blt   do_jump
  used6
  // 162 if_icmpge
```

```
lwz   b, -4(sp)
cmp   cr0, 0, b, tos
lha   a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
bge   do_jump
used6
// 163 if_icmpgt
lwz   b, -4(sp)
cmp   cr0, 0, b, tos
lha   a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
bgt   do_jump
used6
// 164 if_icmple
lwz   b, -4(sp)
cmp   cr0, 0, b, tos
lha   a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
ble   do_jump
used6
// 165 if_acmpeq
lwz   b, -4(sp)
cmp   cr0, 0, b, tos
lha   a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
beq   do_jump
used6
// 166 if_acmpne
lwz   b, -4(sp)
cmp   cr0, 0, b, tos
lha   a, 0(ip)
lwzu tos, -8(sp)
addi ip, ip, 2
bne   do_jump
used6
// 167 goto
lha   a, 0(ip)
subi  ip, ip, 1
add   ip, ip, a
used3
// 168 jsr
pushtos
lha   a, 0(ip)
addi tos, ip, 2 // addr of next instr
subi ip, ip, 1
add   ip, ip, a
used6
// 169 ret
lbz   a, 0(ip)
times4(a)           // convert to offset
lwzx ip, fp, a
used3
// 170 tableswitch
mr   r3, ip
```

```
mr   r4, tsBase
mr   r5, tos
bl   TableSwitch
mr   ip, r3
poptos
used6
```
// 171 lookupswitch
```
mr   r3, ip
mr   r4, tsBase
mr   r5, tos
bl   LookupSwitch
mr   ip, r3
poptos
used6
```
// 172 ireturn
```
lwz   ip, 8(rp)
mr   sp, fp
lwz   fp, 4(rp)
lwz   tsBase, 0(rp)
addi rp, rp, 12
used5
```
// 173 lreturn
```
lwz   ip, 8(rp)
mr   sp, fp
lwz   fp, 4(rp)
lwz   tsBase, 0(rp)
lwzu tos, -4(sp)
addi rp, rp, 12
used6
```
// 174 freturn
```
lwz   ip, 8(rp)
mr   sp, fp
lwz   fp, 4(rp)
lwz   tsBase, 0(rp)
lwzu tos, -4(sp)
addi rp, rp, 12
used6
```
// 175 dreturn
```
lwz   ip, 8(rp)
mr   sp, fp
lwz   fp, 4(rp)
lwz   tsBase, 0(rp)
lwzu tos, -4(sp)
addi rp, rp, 12
used6
```
// 176 areturn
```
lwz   ip, 8(rp)
mr   sp, fp
lwz   fp, 4(rp)
lwz   tsBase, 0(rp)
addi rp, rp, 12
used5
```
// 177 return
```
lwz   ip, 8(rp)
mr   sp, fp
lwz   fp, 4(rp)
lwz   tsBase, 0(rp)
lwzu tos, -4(sp)
```

```
addi rp, rp, 12
used6
// 178 getstatic
lhz   a, 0(ip)
addi ip, ip, 2
times4(a)
lwzx b, cp, a
b  getstatic_more
nop;nop;nop
// 179 putstatic
lhz   a, 0(ip)
addi ip, ip, 2
times4(a)
lwzx b, cp, a
b  putstatic_more
nop;nop;nop
// 180 getfield
lhz   a, 0(ip)
addi ip, ip, 2
times4(a)
lwzx b, cp, a
b  getfield_more
nop;nop;nop
// 181 putfield
lhz   a, 0(ip)
addi ip, ip, 2
times4(a)
lwzx b, cp, a
b  putfield_more
nop;nop;nop
// 182 invokevirtual
lhz   a, 0(ip)
addi ip, ip, 2
times4(a)        // convert to index
lwzx a, cp, a
lwz   methodP, 1(a)
stwu ip, -4(rp)
b  start_method
nop
// 183 invokenonvirtual
lhz   a, 0(ip)
addi ip, ip, 2
times4(a)        // convert to index
lwzx a, cp, a
lwz   methodP, 1(a)
stwu ip, -4(rp)
b  start_method
nop
// 184 invokestatic
lhz   a, 0(ip)
addi ip, ip, 2
times4(a)        // convert to index
lwzx a, cp, a
lwz   methodP, 1(a)
stwu ip, -4(rp)
b  start_method
nop
// 185 invokeinterface
```

```
addi ip, ip, 2
used1
// 186 undefined
used0
// 187 new
addi ip, ip, 2
used1
// 188 newarray
lbz   r3, 0(ip)
addi ip, ip, 1
mr   r4, tos
bl   GetNewArray
mr   tos, r3
used5
// 189 anewarray
addi ip, ip, 2  // skip type
mr   r3, tos
bl   GetANewArray
mr   tos, r3
used4
// 190 arraylength
lwz   tos, 0(tos)
used1
// 191 athrow
used0
// 192 checkcast
addi ip, ip, 2
used1
// 193 instanceof
addi ip, ip, 2
li   tos, 1       // assume true
used2
// 194 monitorenter
poptos
used1
// 195 monitorexit
poptos
used1
// 196 wide
addi ip, ip, 1
used1
// 197 multianewarray
lhz   r3, 0(ip)
lbz   r4, 2(ip)
addi ip, ip, 3
pushtos
b  multianewarray_more
nop;nop;
// 198 ifnull
cmpi cr0, 0, tos, 0
lha   a, 0(ip)
lwzu tos, -4(sp)
addi ip, ip, 2
subi a, a, 3
bne   next0
add   ip, ip, a
used7
// 199 ifnonnull
```

```
  cmpi cr0, 0, tos, 0
  lha   a, 0(ip)
  lwzu tos, -4(sp)
  addi ip, ip, 2
  subi a, a, 3
  beq   next0
  add   ip, ip, a
  used7
  // 200 goto_w
  lwz   a, 0(ip)
  subi a, a, 1
  add   ip, ip, a
  used3
  // 201 jsr_w
  pushtos
  lwz   a, 0(ip)
  addi tos, ip, 5
  subi a, a, 1
  add   ip, ip, a
  used6
  // 202 breakpoint
  used0
  // 203 unused - but we use it to signal final exit
  b exitVM
  used1
  used0  // 204
  used0  // 205
  used0  // 206
  used0  // 207
  used0  // 208
  // 209 ret_w
  lhz   a, 0(ip)
  times4(a)
  lwzx ip, fp, a
  used3

  // remaining opcodes unused

exitVM:
  mr   r3, tos
  frfree
  blr

ldc_more:
  stw   sp, SP
  bl   PushConstant
  lwz   sp, SP
  poptos
  b  next0

multianewarray_more:
  stw   sp, SP
  bl   PushMultiANewArray
  lwz   sp, SP
  poptos
  b  next0
```

```
getstatic_more:
  lwz    a, 1(b)   // get fieldP
  lwz    b, 8(a)   // get size
  cmpi cr0, 0, b, 0
  lwz    c, 4(a)   // get offset
  beq    next0
  pushtos
  lwzx tos, h0, c
  b  next0

putstatic_more:
  lwz    a, 1(b)   // get fieldP
  lwz    b, 8(a)   // get size
  cmpi cr0, 0, b, 0
  lwz    c, 4(a)   // get offset
  beq    drop1ngo
  stwx tos, h0, c
  poptos
  b  next0

getfield_more:
  lwz    a, 1(b)   // get fieldP
  lwz    b, 8(a)   // get size
  cmpi cr0, 0, b, 0
  lwz    c, 4(a)   // get offset
  beq    drop1ngo
  lwzx tos, tos, c
  b  next0

putfield_more:
  lwz    a, 1(b)   // get fieldP
  lwz    b, 8(a)   // get size
  cmpi cr0, 0, b, 0
  lwz    c, 4(a)   // get offset
  beq    drop2ngo
  lwzu a, -4(sp)
  stwx tos, a, c
  poptos
  b  next0

drop2ngo:
  poptos
drop1ngo:
  poptos
  b  next0

}
```

PushConstant

```
static void PushConstant(long n)
{
  u1   *p;

  p = Constants[n];
  switch (*p++)
  {
    case C_Utf8:
```

```
    case C_Unicode:
      // since no operations act on these objects,
      // I just make the const data be the object
      // (w/o the tag) for simplicity of the test code.
      PUSH((long)p);
      break;
    case C_Integer:  PUSH(*(s4*)p); break;
    case C_Float: break;
    case C_Long:  break;
    case C_Double: break;
    case C_Class: break;
    case C_String: PushConstant(*(u2*)p); break;
    case C_Fieldref:
    case C_Methodref:
    case C_InterfaceMethodref:
    case C_NameAndType: DebugStr("\pit can happen"); break;
  }
}
```

TableSwitch

```
static u1 *TableSwitch(u1 *ip, long tsBase, long n)
{
  long *base;
  long defaultOffset;
  long low;
  long high;

  base = (long *) (((((long)ip-tsBase) + 3) & -4)+tsBase);
  defaultOffset = *base++;
  low = *base++;
  high = *base++;
  ip -= 1;

  if (n < low || n > high)
    ip += defaultOffset;
  else
    ip += base[n - low];
  return ip;
}
```

LookupSwitch

```
static u1 *LookupSwitch(u1 *ip, long tsBase, long key)
{
  long *base;
  long defaultOffset;
  long numPairs;
  long match;

  base = (long *) (((((long)ip-tsBase) + 3) & -4)+tsBase);
  defaultOffset = *base++;
  numPairs = *base++;
  ip -= 1;

  while (numPairs > 0)
  {
    match = *base++;
    numPairs -= 1;
```

```
    if (key == match)
      return ip + *base;
    base += 1;
  }
  return ip + defaultOffset;
}
```

GetNewArray

```
// Array object: # elements, data
// data is padded to multiple of 4
static long GetNewArray(long type, long size)
{
  long  result;
  long  elementSize;

  result = (long) HP;
  *HP++ = size;
  if (type == 10)       // int
    elementSize = 4;
  else if ((type & 3) == 0) // boolean or byte
    elementSize = 1;
  else if ((type & 3) == 1) // char or short
    elementSize = 2;
  else
    elementSize = 0;   // unsupported types
  HP += ((size * elementSize + 3) >> 2);
  return result;
}
```

GetANewArray

```
static long GetANewArray(long size)
{
  long  result;
  long  elementSize;
  u4    *objectP;

  result = (long) HP;
  *HP++ = size;
  elementSize = 4;
  objectP = (u4*) HP;
  HP = (long *) ((u1 *) HP + size * 4);
  while (size > 0)
  {
    size -= 1;
    *objectP = (u4)HP;
    HP += elementSize;
  }
  return result;
}
```

PushMultiANewArray

```
// Rather complicated function to allocate a
// multi-dimensional array as arrays of array objects.
static void PushMultiANewArray(long classIndex,
            long numDimensions)
{
  u1   *cp;
```

```
long nameIndex;
u1   *name;
int   type;
long elementSize;
long arrayRef;
long size;
long copies;
long i, j;
long dim;
long dataLongs;
long *subArrayRef;
long subArrayLongs;

arrayRef = (long) HP;
cp = Constants[classIndex];
nameIndex = *(u2*) (cp+1);
cp = Constants[nameIndex];
name = cp + 3;
type = name[numDimensions];  // skip the known '[' chars
if (type == 'I' || type == '[')           // int or ref
  elementSize = 4;
else if (type == 'Z' || type == 'B')      // bool or byte
  elementSize = 1;
else if (type == 'C' || type == 'S')      // char or short
  elementSize = 2;
else
  elementSize = 0;   // unsupported types
copies = 1;

size = *(SP-1);       // the nth dimension
dataLongs = (size * elementSize + 3) >> 2;
for (dim = 0; dim < numDimensions - 1; dim++)
{
  size = *(SP-numDimensions+dim);
  if (dim == numDimensions - 2)
    subArrayLongs = dataLongs + 1;
  else
    subArrayLongs = *(SP-numDimensions+dim+1) + 1;
  subArrayRef = HP + (size + 1) * copies;
  for (i = 0; i < copies; i++)
  {
    *HP++ = size;
    for (j = 0; j < size; j++)
    {
      *HP++ = (long) subArrayRef;
      subArrayRef += subArrayLongs;
    }
  }
  copies *= size;
}
// last dim is special since it has no subarrays
size = *(SP-1);       // the nth dimension
for (i = 0; i < copies; i++)
{
  *HP++ = size;
  HP += dataLongs;
}
```

```
  SP -= numDimensions; // remove dimensions
  PUSH(arrayRef);
}
```