CS 536 — Fall 2012

**CSX Code Generation Routines**

Part III

**Indexing and Assigning Arrays**

The JVM includes special instructions for loading from and storing into arrays. Integer arrays use iaload and iastore. Boolean arrays use baload and bastore. Character arrays use caload and castore.

To assign arrays we’ll use the CSXLib methods:

int [] cloneIntArray(int[]),

boolean[] cloneBoolArray(boolean[]),

char[] cloneCharArray(char[]),

char[] convertString(String),

int [] checkIntArrayLength(int[], int[]),

boolean[] checkBoolArrayLength(boolean[], boolean[]),

char[] checkCharArrayLength(char[], char[]).

These routines make a copy (clone) of the source array and check the length of the source

and target arrays (in case an array parameter is involved). If the array lengths are not compatible, an ArraySizeException is raised.

We’ll extend computeAdr, storeName and the visit methods for nameNodes and asgNodes to include array indexing and assignment.

void visit(nameNode n) { // Final version

n.adr = stack;

if (n.subscriptVal.isNull()) {

// Simple (unsubscripted) identifier

if (n.varName.idinfo.kind == Var ||

n.varName.idinfo.kind == Value ||

n.varName.idinfo.kind == ScalarParm) {

// id is a scalar variable, parameter or const

if (n.varName.idinfo.adr == Global){

// id is a global

String label = n.varName.idinfo.label;

loadGlobalInt(label);

} else { // (n.varName.idinfo.adr == Local)

n.varIndex = n.varName.idinfo.varIndex;

loadLocalInt(n.varIndex);

} } else { // varName is an array var or array parm

if (n.varName.idinfo.adr == Global){

n.label = n.varName.idinfo.label;

loadGlobalReference(n.label,

arrayTypeCode(n.varName.idinfo.type));

} else { // (n.varName.idinfo.adr == local)

n.varIndex = n.varName.idinfo.varIndex;

loadLocalReference(n.varIndex);

} }

} else { // This is a subscripted variable

// Push array reference first

if (n.varName.idinfo.adr == Global){

n.label = n.varName.idinfo.label;

loadGlobalReference(n.label,

arrayTypeCode(n.varName.idinfo.type));

} else { // (n.varName.idinfo.adr == local)

n.varIndex = n.varName.idinfo.varIndex;

loadLocalReference(n.varIndex);

} // Next compute subscript expression

this.visit(n.subscriptVal);

// Now load the array element onto the stack

switch(n.type){

case Integer:

// Generate: iaload

break;

case Boolean:

// Generate: baload

break;

case Character:

// Generate: caload

break;

} } }

// Compute address associated w/ name node

// DON'T load the value addressed onto the stack

void computeAdr(nameNode name) { // Final version

if (name.subscriptVal.isNull()) {

// Simple (unsubscripted) identifier

if (name.varName.idinfo.kind == Var ||

name.varName.idinfo.kind == ScalarParm) {

// id is a scalar variable

if (name.varName.idinfo.adr == Global) {

name.adr = Global;

name.label = name.varName.idinfo.label;

} else { // varName.idinfo.adr == Local

name.adr = Local;

name.varIndex =

name.varName.idinfo.varIndex;

}} else { // Must be an array

// Push ref to target array to check length

if (name.varName.idinfo.adr == Global){

name.label = name.varName.idinfo.label;

loadGlobalReference(name.label,

arrayTypeCode(name.varName.idinfo.type));

} else { // (name.varName.idinfo.adr == local)

name.varIndex =

name.varName.idinfo.varIndex;

loadLocalReference(name.varIndex);

} }

} else { // This is subscripted variable

// Push array reference first

if (name.varName.idinfo.adr == Global){

name.label = name.varName.idinfo.label;

loadGlobalReference(name.label,

arrayTypeCode(name.varName.idinfo.type));

} else { // (name.varName.idinfo.adr == local)

name.varIndex = name.varName.idinfo.varIndex;

loadLocalReference(name.varIndex);

} // Next compute subscript expression

this.visit(name.subscriptVal);

} }

void storeName(nameNode name) { // Final version

if (name.subscriptVal.isNull()) {

// Simple (unsubscripted) identifier

if (name.varName.idinfo.kind == Var ||

name.varName.idinfo.kind == ScalarParm) {

if (name.adr == Global)

storeGlobalInt(name.label);

else // (name.adr == Local)

storeLocalInt(name.varIndex);

} else {// Must be an array

// Check the lengths of source & target arrays

switch(name.type){

case Integer:

genCall("CSXLib/checkIntArrayLength([I[I)[I");

break;

case Boolean:

genCall(

"CSXLib/checkBoolArrayLength([Z[Z)[Z");

break;

case Character:

genCall(

"CSXLib/checkCharArrayLength([C[C)[C");

break;

} // Now store source array in target variable

if (name.varName.idinfo.adr == Global){

name.label = name.varName.idinfo.label;

storeGlobalReference(name.label,

arrayTypeCode(name.varName.idinfo.type));

} else { // (name.varName.idinfo.adr == local)

name.varIndex =

name.varName.idinfo.varIndex;

storeLocalReference(name.varIndex);

} }

} else // This is a subscripted variable

// A reference to the target array, the

// subscript expression and the source expression

// have already been pushed.

// Now store the source value into the array

switch(name.type){

case Integer:

//Generate: iastore

break;

case Boolean:

//Generate: bastore

break;

case Character:

//Generate: castore

break;

}

}

void visit(asgNode n) { // Final version

// Compute address associated with LHS

computeAdr(n.target);

// Translate RHS (an expression)

this.visit(n.source);

// Check to see if source needs to be cloned or converted

if (n.source.kind == Array ||

n.source.kind == ArrayParm)

switch(n.source.type){

case Integer:

genCall("CSXLib/cloneIntArray([I)[I");

break;

case Boolean:

genCall("CSXLib/cloneBoolArray([Z)[Z");

break;

case Character:

genCall("CSXLib/cloneCharArray([C)[C");

break;

}

else if (n.source.kind == String)

genCall("CSXLib/convertString(Ljava/lang/String;)[C");

// Value to be stored is now on the stack

// Store it into LHS

storeName(n.target);

}

The increment operation is essentially a load, add and store. Incremented array elements require special care since the index expression must be evaluated only once (in case of side-effects). In the case of indexed arrays, an array reference and index expression are pushed onto the stack and then duplicated. One pair is used to load the array element and the other pair is used to store the incremented value.

void visit(incrementNode n) {

if (n.target.subscriptVal.isNull()){

// Simple (unsubscripted) identifier

this.visit(n.target); //Evaluate ident onto stack

loadI(1);

gen("iadd"); //incremented ident now on stack

computeAdr(n.target);

storeName(n.target);

} else { // Subscripted array element

computeAdr(n.target); //Push array ref and index

gen("dup2"); // Duplicate array ref and index

// (one pair for load, 2nd pair for store)

// Now load the array element onto the stack

switch(n.target.type){

case Integer:

gen("iaload");

break;

case Boolean:

gen("baload");

break;

case Character:

gen("caload");

break;

}

loadI(1);

gen("iadd"); // incremented identifier now on stack

storeName(n.target);

}

}