Bali++ to SaM Code Conversion Templates

CS212

Fall 2005

1 Introduction

1.1 What is this?

This document provides templates for how to express Bali++ code in terms of SaM instructions. It is pretty self explanatory. You can use this document as a guide in writing your compiler. However, you do not have to follow these templates. Note that goal of the project in this course is to create a working compiler and not to create an implementation of these templates. If you implement these templates and your compiler does not work you cannot blame the templates. However, if you do find any problems with these templates please post them to the newsgroup or send an email to the course staff.

2 Functions

2.1 Callee Code

- Code for a function, like **`t0 g ( t1 p1, t2 p2, ... , tn pn) {{ variables } statements }`**

g_label:	specific label for the function
ADDSP *num_variables*	allocate local variables
code for statements	code for primary body in callee
PUSHIMM 0	push default return value
g_label_End:	label for portion of Samcode to finish callee
STOREOFF -(n+1)	store return value
ADDSP -*num_variables*	deallocate local variables
RST	return to caller

2.2 Caller Code

- Code for function call, like **g (e1 , _ , en)**

ADDSP 1	allocate space for function return value
code for e1	resolve *e1* to push param value
...	...
code for en	resolve *en* to push param value
LINK	save FBR of caller and set FBR to callee (*g*)
JSR *g_label*	jump to subroutine *g* (the callee)
UNLINK	pop saved FBR and store as new FBR
ADDSP -n	after returning, pop parameters

3 Statements

- Code for a statement block, like **{ s1 , _ , sn}**

code for s1
...
code for s1

- Code for a conditional statement, like **if (e1) s1**

code for e1
JUMPC *thenifLabel*
JUMP *endifLabel*
thenifLabel:
code for s1
endifLabel:

- Code for a conditional statement, like **if (e1) s1 else s2**

code for e1
JUMPC *thenifLabel*
code for s2
JUMP *endifLabel*
thenifLabel:
code for s1
endifLabel:

- Code for do ... until loop, like **do s1 until (e1)**

doLabel:
code for s1
code for e1
NOT
JUMPC *doLabel*:

- Code for a for loop, like **`for (e1; e2; e3) s1`**

code for e1
ADDSP -1
beginForLabel:
code for e2
ISNIL
JUMPC *endForLabel*
code for s1
code for e3
ADDSP -1
JUMP *beginForLabel*
endForLabel:

Note: if *e2* is not specified it is assigned the value "true"

- Code for an expression statement, like **`e1;`**

code for e1
ADDSP -1

- Code for output, like, **`print e1;`** where *e1* is an integer

code for e1
WRITE

- Code for output, like **`print e1;`** where *e1* is a character

code for e1
WRITECH

- Code for output, like **`print e1;`** where *e1* is a character pointer

code for e1
WRITESTR

- Code for a return statement, like **`return e1;`**

code for e1
JUMP *g_name_End* jump to the label at the end of the current function (see section 2.2)

4 Expressions and Expression Parts

4.1 Assignment Operation

- Code for **`e1 = e2`** where *e1* is the name of a global variable

code for e2
DUP
STOREABS *address_of_e1*

- Code for **`e1 = e2`** where *e1* is the name of a local variable or a parameter

code for e2
DUP
STOREOFF *address_of_e1*

note: if *e1* is a local variable its address is positive if *e1* is a parameter its address is negative

- Code for **`e1 = e2`** where *e1* is a dereferenced pointer, like **`* ep`** (i.e. the code for **`* ep = e2`**)

code for e2
DUP
code for ep
SWAP
STOREIND

4.2 Array Element Expression

- Code for **`e1 [ e2 ]`** is the code for **`* ( e1 + e2)`**

4.3 Binary Operations

- Code for **`e1 + e2`**

code for e1
code for e2
ADD

- Code for **`e1 - e2`**

code for e1
code for e2
SUB

- Code for **`e1 * e2`**

code for e1
code for e2
TIMES

- Code for **`e1 / e2`**

code for e1
code for e2
DIV

- Code for **`e1 % e2`**

code for e1
code for e2
MOD

- Code for ***e1 == e2***

code for e1
code for e2
EQUAL

- Code for ***e1 != e2***

code for e1
code for e2
EQUAL
NOT

- Code for ***e1 > e2***

code for e1
code for e2
CMP
ISNEG

- Code for ***e1 < e2***

code for e1
code for e2
CMP
ISPOS

- Code for ***e1 >= e2***

code for e1
code for e2
CMP
ISPOS
NOT

- Code for ***e1 <= e2***

code for e1
code for e2
CMP
ISNEG
NOT

- Code for ***e1 && e2*** with short circuiting

code for e1
DUP
NOT
JUMPC *endLabel*
code for e2
AND
endLabel:

- Code for **`e1 || e2`** with short circuiting

code for e1
DUP
JUMPC *endLabel*
code for e2
OR
endLabel:

- Code for **`e1 ^ e2`**

code for e1
code for e2
XOR

4.4 Unary Operations

- Code for **`- e1`**

PUSHIMM 0
code for e1
SUB

- Code for **`! e1`**

code for e1
NOT

- Code for **`& e1`** where *e1* is a global variable

PUSHIMMMA *address_of_e1*

- Code for **`& e1`** where *e1* is a local variable or a parameter

PUSHFBR
PUSHIMM *address_of_e1*
ADD

note: if *e1* is a local variable its address is positive if *e1* is a parameter its address is negative

- Code for **`* e1`**

code for e1
PUSHIND

4.5 Other Expression Parts

- Code for **`e1`** where *e1* is the name of a global variable

PUSHABS *address_of_e1*

- Code for **`e1`** where *e1* is the name of a local variable or a parameter

PUSHOFF *address_of_e1*

note: if *e1* is a local variable its address is positive if *e1* is a parameter its address is negative

- Code for **`e1`** where *e1* is the name of an integer constant

PUSHIMM *value_of_e1*

- Code for **`e1`** where *e1* is the name of a boolean constant (i.e. true or false)

PUSHIMM *value_of_e1*

- Code for **`e1`** where *e1* is the name of a character constant

PUSHIMMCH *value_of_e1*

- Code for **`null`**

PUSHIMMMA 0

- Code for integer input, like **`readInt()`**

READ

- Code for character input, like **`readChar()`**

READCH

- Code for memory allocation, like **`malloc( e1 )`**

code for e1
MALLOC

- Code for memory deallocation, like **`free( e1 )`**

code for e1
FREE
PUSHIMM 0

- Code for type cast, like **`< t1 > e1`**

code for e1