

## Master's Thesis

Martin Holm Cservenka
Department of Computer Science
University of Copenhagen
djp595@alumni.ku.dk

Design and Implementation of Dynamic Memory Management in a Reversible Object-Oriented Programming Language

Supervisors: Robert Glück & Torben Ægidius Mogensen

Submitted: January  $25^{th}$ , 2018

Revision 1.01

## **Revision History**

1.01

Revision Date	$\mathbf{Author}(\mathbf{s})$	Description
---------------	-------------------------------	-------------

 $2018\text{-}04\text{-}11 \quad \text{Martin}$ 

Fixed incorrect subfigures (d, e, f) in Figure 4.11. Updated Figure 3.4 to better emphesize output copying. Updated Figure 3.5 to better reflect the harmless garbage flow through programs.

Revision History 1 of 231

### Abstract

The reversible object-oriented programming language (ROOPL) was presented in late 2016 and proved that object-oriented programming paradigms works in the reversible setting. The language featured simple statically scoped objects which made non-trivial programs tedious, if not impossible to write using the limited tools provided. We introduce an extension to ROOPL in form the new language ROOPL++, featuring dynamic memory management and fixed-sized arrays for increased language expressiveness. The language is a superset of ROOPL and has formally been defined by its language semantics, type system and computational universality. Considerations for reversible memory manager layouts are discussed and ultimately lead to the selection of the Buddy Memory layout. Translations of the extensions added in ROOPL++ to the reversible assembly language PISA are presented to provide garbage-free computations. The dynamic memory management extension successfully increases the expressiveness of ROOPL and as a result, shows that non-trivial reversible data structures, such as binary trees and doubly-linked lists, are feasible and do not contradict the reversible computing paradigm.

Abstract 2 of 231

### Preface

This Master's Thesis is submitted as the last part for the degree of Master of Science in Computer Science at the University of Copenhagen, Department of Computer Science, presenting a 30 ECTS workload.

The thesis consists of 231 pages and a ZIP archive containing source code and test programs developed as part of the thesis work.

I would like to thank my two supervisors, Robert Glück and Torben Mogensen, for their invaluable supervision and guidance throughout this project and introduction to the field of reversible computing. A big thanks to my university colleague and friend, Tue Haulund, for allowing me to continue his initial work on ROOPL and providing information, sparring and source code material and for being a great ally through our years at the University of Copenhagen. In addition, thanks to my dear aunt Doris, for financially supporting my studies by paying for all my books needed. Finally, a thanks to Jess, for all the love and support throughout the entire span of my thesis process.

Preface 3 of 231

## Table of Contents

1	$\mathbf{Intr}$	oduction 8
	1.1	Reversible Computing
	1.2	Object-Oriented Programming
	1.3	Reversible Object-Oriented Programming
	1.4	Motivation
	1.5	Thesis Statement
	1.6	Outline
<b>2</b>	The	Roopl++ Language 11
	2.1	Syntax
	2.2	Object Instantiation
	2.3	Array Model
	2.4	Array Instantiation
	2.5	Referencing
	2.6	Local Blocks
	2.7	ROOPL++ Expressiveness
		2.7.1 Linked List
		2.7.2 Binary Tree
		2.7.3 Doubly Linked List
	2.8	Type System
		2.8.1 Preliminaries
		2.8.2 Expressions
		2.8.3 Statements
		2.8.4 Programs
	2.9	Language Semantics
		2.9.1 Preliminaries
		2.9.2 Expressions
		2.9.3 Statements
		2.9.4 Programs
	2 10	Program Inversion
	2.10	2.10.1 Invertibility of Statements
		2.10.2 Type-Safe Statement Inversion
	2.11	Computational Strength
		2.11.1 Reversible Turing Machines
		2.11.2 Tape Representation
		2.11.3 Reversible Turing Machine Simulation
3	Dyn	amic Memory Management 45
•	3.1	Fragmentation
	0.1	3.1.1 Internal Fragmentation
		on the contract of the contraction of the contracti

Table of Contents 4 of 231

		3.1.2 External Fragmentation	46		
	3.2	Memory Garbage	47		
	3.3	Linearity and Reference Counting	49		
	3.4	Heap Manager Layouts	49		
		3.4.1 Memory Pools	50		
		3.4.2 One Heap Per Record Size	51		
		3.4.3 One Heap Per Power-Of-Two	52		
		3.4.4 Shared Heap, Record Size-Specific Free Lists	54		
		3.4.5 Buddy Memory	55		
4	Con	npilation	58		
	4.1	1	58		
	4.2		59		
	4.3	Inherited ROOPL features	60		
	4.4	8	61		
	4.5	Buddy Memory Translation	62		
	4.6	3	66		
	4.7	Referencing	67		
	4.8	Arrays	68		
		4.8.1 Construction and Destruction	69		
		4.8.2 Array Element Access	70		
	4.9	Error Handling	70		
	4.10	Implementation	72		
	4.11	Evaluation	73		
5	Con	nclusions	74		
	5.1	Future Work	74		
$\mathbf{R}$	References 76				
$\mathbf{A}_{1}$	ppen	dix A Pisa Translated Buddy Memory	<b>7</b> 9		
$\mathbf{A}_{1}$	Appendix B ROOPLPPC Source Code				
A	Appendix C Example Ouput				

Table of Contents 5 of 231

# List of Figures

2.1	Example ROOPL++ program implementing the Fibonacci function	11
2.2	Syntax domains and EBNF grammar for ROOPL++	12
2.3	construct/destruct-blocks can be considered syntactic sugar	18
2.4	Linked List cell class	19
2.5	Linked List class	20
2.6	Binary Tree class	21
2.7	Binary Tree node class (cont)	21
2.8	Binary Tree node class	22
2.9	Multiple identical reference are needed for a doubly linked list implementation .	22
2.10	Doubly Linked List class	23
2.11		23
2.12	Doubly Linked List Cell class (cont)	24
2.13	Definition gen for constructing the finite class map $\Gamma$ of a given program $p$ ,	
	originally from [11]	26
2.14	Definition of fields and methods, originally from [11]	26
2.15	Definition array Type for mapping types of arrays to either class types or the	
	integer type	26
2.16	Typing rules for expressions in ROOPL, originally from [11]	27
2.17	Typing rule extension for the ROOPL typing rules	27
2.18	Typing rules for statements in ROOPL, originally from [11]	28
2.19	Typing rules extensions for statements in ROOPL++	29
2.20	Typing rules for class methods, classes and programs, originally from [11]	30
2.21	Semantic values, originally from [11]	31
2.22	Semantic inference rules for expressions, originally from [11]	31
2.23	Extension to the semantic inference rules for expression in ROOPL++	32
2.24	Definition of binary expression operator evaluation, originally from [11]	32
	,	33
2.25b	Semantic inference rules for statements, modified from [11] (cont)	34
2.25c	Extension to the semantic inference rules for statements in ROOPL++	35
2.26	Semantic inference rules for programs, originally from [11]	36
2.27	ROOPL++ statement inverter, extended from [11]	37
2.28	Modified statement inverter for statements, originally from [11]	38
2.29	0 1	43
2.30	Method for executing a single TM transition	44
2.31	Main RTM simulation method	44
3.1a	O Company of the comp	46
3.1b	<u>e</u>	46
3.2	1	47
3.3	Example of avoiding external fragmentation using allocation and deallocation order	47

List of Figures 6 of 231

3.4	The "garbage" output of an injective function $f$ is the input to its inverse function $f^{-1}$	48
3.5	All free lists are considered equivalent "garbage" in terms of injective functions .	48
3.6	Memory layout using one heap per record size	51
3.7	Memory layout using one heap per power-of-two	53
3.8	Record size-specific free lists on a shared heap (powers of two)	55
3.9	Buddy Memory block allocation example	55
4.1	Memory layout of a ROOPL program, originally from [11]	58
4.2	Illustration of object memory layout	59
4.3	Memory layout of a ROOPL++ program	59
4.4	Definition of pseudoinstructions SUBI, PUSH and POP, modified from [11]	60
4.5	Overall layout of a translated ROOPL++ program	61
4.6	Dynamic dispatch approach for entering the allocation subroutine	62
4.7	PISA translation of the nested conditionals in the Buddy Memory algorithm	63
4.8	PISA translation of the outer <b>if-then</b> statement for the Buddy Memory algorithm	64
4.9	PISA translation of the inner <b>if-then</b> statement for the Buddy Memory algorithm	64
4.10	PISA translation of the inner <b>else</b> statement for the Buddy Memory algorithm .	64
4.11	Non-opposite deallocation results in a different free list after termination $\dots$	65
4.12	PISA translation of the <b>malloc</b> procedure entry point of Buddy Memory algorithm	66
4.13	PISA translation of heap allocation and deallocation for objects	66
4.14	PISA translation of a ROOPL++ object block	67
4.15	Illustration of object memory layout	68
4.16	PISA translation of the reference copying and deletion statements	68
4.17	Illustration of prefixing in the memory layout of two ROOPL++ arrays	69
4.18	PISA translations of array allocation and deallocation statements	69
4.19	Illustration of array memory storage layout	70
4.20	Lines of code comparison between target and compiled ROOPL++ programs	73

List of Figures 7 of 231

### Introduction

In recent years, technologies such as cloud-based services, deep learning, cryptocurrency mining and other services requiring large computational power and availability have been on the rise. Most of these services are hosted on massive server parks, consuming immense amounts of electricity in order to power the machines and the cooling architectures as heat dissipates from the hardware. A recent study showed that the Bitcoin network including its mining processes' currently stands at 0.13% of the total global electricity consumption, rivaling the usage of a small country like Denmark's [6]. With the recent years focus on climate and particularly energy consumption, companies have started to attempt to reduce their power usage in these massive server farms. As an example, Facebook built new server park in the arctic circle in 2013, in an attempt to take advantage of the natural surroundings in the cooling architecture to reduce its power consumption [24].

Reversible computing presents a possible solution the problematic power consumption issues revolving around computations. Traditional, irreversible computers dissipates heat during their computation. Landauer's principle states that deletion of information in a system always results by an increase in energy consumption. In reversible computing, all information is preserved throughout the execution, and as such, the energy consumption theoretically should be smaller [14].

Currently, reversible computing is not commercially appealing, as it is an area which still is being actively researched. However, several steps has been taken in the direction of a fully reversible system, which some day might be applicable in a large setting. Reversible machine architectures have been presented such as the Pendulum architecture and its instruction set Pendulum ISA (PISA) [26, 3] and the Bobish architecture and instruction set [23] and high level languages Janus [16, 31, 29] and R [7] exists.

While cryptocurrency mining and many other computations are not reversible, the area remains interesting in terms of its applications and gains.

### 1.1 Reversible Computing

Reversible computing is a two-directional computational model in which all processes are time-invertible. This means, that at any time during execution, the computation can return to a former state. In order to maintain reversibility, the reversible computational model cannot compute many-to-one functions, as the models requires an exact inverse  $f^{-1}$  of a function f in order to

support backwards determinism. Therefore, reversible programs must only consist of *one-to-one* functions, also known as *injective* functions, which result in a garbage-free computation, as garbage-generating functions simply can be unwinded to clean up.

Each step of a reversible program is locally invertible, meaning each step has exactly one inverse step. A reversible program can be inverted simply by computing the inverse of each of its steps, without any knowledge about the overall functionality or requirements of the program. This property immediately yields interesting consequences in terms of software development, as an encryption or compression algorithm implemented in a reversible language immediately yields the decryption or decompression algorithm by running the algorithm backwards.

The reversibility is however not free and comes and the cost of strictness when writing programs. Almost every popular, irreversible programming language features a conditional component in form of **if-else**-statements. In these languages, we only define the *entry*-condition in the conditional, that is, the condition that determines which branch of the component we continue execution in. In reversible languages, we must also specify an *exit*-condition, such that we can determine which branch we should follow, when executing the program in reverse. In theory, this sounds trivial, but in practice it turns to add a new layer of complexity when writing programs.

### 1.2 Object-Oriented Programming

Object-oriented programming (OOP) has for many years been the most widely used programming paradigm as reflected in the popular usage of object-oriented programming languages, such as the C-family languages, JAVA, PHP and in recent years JAVASCRIPT and PYTHON. The OOP core concepts such as *inheritance*, *encapsulation* and *polymorphism* allows complex systems to be modeled by breaking the system into smaller parts in form of abstract objects [17].

### 1.3 Reversible Object-Oriented Programming

The high-level reversible language ROOPL (Reversible Object-Oriented Programming Language) was introduced in late 2016 [11, 12]. The language extends the design of previously existing reversible imperative languages with object-oriented programming language features such as user-defined data types, class inheritance and subtype-polymorphism. As a first, ROOPL successfully integrates the object-oriented programming (OOP) paradigms into the reversible computation setting using a static memory manager to maintain garbage-free computation, but at cost of programmer usability as objects only lives within **construct** / **deconstruct** blocks, which needs to be predefined, as the program call stack is required to be reset before program termination.

Conceptualizations and ideas for the Joule language was also published in 2016 [22]. The language, a homonym of Janus Object-Oriented Language, Jool, presented an alternative OOP extension to Janus, differing from Roople. The language featured heap allocated objects with constructors and multiple object references, as such also addressing the problems with Roople. The language is still a work in progress, aiming to provide a useful, reversible object oriented-programming language.

#### 1.4 Motivation

The block defined objects of ROOPL and lack of multiple references are problematic when writing complex, reversible programs using OOP methodologies as they pose severe limitations on the expressiveness. It has therefore been proposed to extend and partially redesign the language with dynamic memory management in mind, such that these shortcomings can be addressed, and ultimately increase the usability of reversible OOP. Work within the field of reversible computing related to heap manipulation [2], reference counting [18] and garbage collection [19] suggests that a ROOPL extension is feasible.

### 1.5 Thesis Statement

An extension of the reversible object-oriented programming language with dynamic memory management is feasible and effective. The resulting expressiveness allows non-trivial reversible programming previously unseen, such as reversible data structures, including linked lists, doubly linked lists and trees.

#### 1.6 Outline

This Master's thesis consists of four chapters, besides the introductory chapter. The following summary describes the following chapters.

- Chapter 2 formally defines the ROOPL extension exemplified by the new language ROOPL++, a superset of ROOPL.
- Chapter 3 serves as a brief description of dynamic memory management along with a discussion of various reversible, dynamic memory management layouts.
- Chapter 4 presents the translation techniques utilized in compiling a ROOPL++ program to PISA instructions.
- Chapter 5 presents the conclusions of the thesis and future work proposals.

Besides the five chapters, a number of appendices is supplied, containing PISA translations of the reversible heap allocation algorithm, the source code of the ROOPL++ to PISA compiler, the ROOPL++ source code for the example programs and their translated PISA versions.

### The ROOPL++ Language

With the design and implementation of the Reversible Object-Oriented Programming Language (Roopl) and the work-in-progress report of Joule, the first steps into the uncharted lands of Object-Oriented Programming (OOP) and reversibility were taken. In this chapter, we present Roopl++, the natural successor to Roopl, improving the object instantiation of the language by letting objects live outside **construct/deconstruct** blocks, allowing complex, reversible programs to be written using OOP methodologies. As with its predecessor, Roopl++ is purely reversible and each component of a program written in Roopl++ is locally invertible. This ensures no computation history is required, nor added program size for backwards program execution.

Inspired by other language successors such as C++ was to C, ROOPL++ is a superset of ROOPL, containing all original functionality of its predecessor, extended with new object instantiation methods for increased programming usability and an array type.

```
1 class Fib
                                                           method get(int out)
                                                              out ^= xs[1]
      int[] xs
2
                                                    19
3
                                                    20
      method init()
                                                    21 class Program
          new int[2] xs
5
                                                    22
                                                           int result
                                                           int n
6
                                                    23
      method fib(int n)
7
                                                    24
           if n = 0 then
                                                           method main()
8
                                                    25
9
               xs[0] ^= 1
                                                    26
                                                               n ^= 4
               xs[1] ^= 1
10
                                                    27
11
           else
                                                    28
                                                               new Fib f
               n = 1
                                                               call f::init()
12
                                                    29
               call fib(n)
                                                               call f::fib(n)
13
                                                    30
               xs[0] += xs[1]
                                                    31
                                                               call f::get(result)
15
               xs[0] \iff xs[1]
                                                    32
                                                               uncall f::fib(n)
           fi xs[0] = xs[1]
                                                               uncall f::init()
16
                                                    33
                                                               delete Fib f
```

Figure 2.1: Example ROOPL++ program implementing the Fibonacci function

### 2.1 Syntax

A ROOPL++ program consists, analogously to a ROOPL program, of one or more class definitions, each with a varying number of fields and class methods. The entry point of the program is a nullary main method, which is defined exactly once and is instantiated during program start-up. Fields of the main object will serve as output of the program, just as in ROOPL.

### ROOPL++ Grammar

```
prog
         ::=
                                                                                             (program)
                class c (inherits c)? (t x)^* m^+
                                                                                      (class definition)
   cl
               c \mid c[e] \mid \mathbf{int}[e]
                                                                                    (class and arrays)
    d
               int \mid c \mid int[] \mid c[]
    t
                                                                                            (data type)
         ::=
                                                                                  (variable identifiers)
               x \mid x[e]
    y
                method q(t x, \ldots, t x) s
                                                                                              (method)
   m
         ::=
               y \odot = e \mid y \iff y
                                                                                          (assignment)
    s
                if e then s else s fi e
                                                                                          (conditional)
                from e do s loop s until e
                                                                                                  (loop)
                construct c \ x - s -  destruct x
                                                                                         (object block)
                \mathbf{local}\ t\ x = e s \mathbf{delocal}\ t\ x = e
                                                                                (local variable block)
                new dy \mid delete dy
                                                                       (object con- and destruction)
                copy d y y | uncopy d y y
                                                                   (reference con- and destruction)
                call q(x, \ldots, x) | uncall q(x, \ldots, x)
                                                                           (local method invocation)
                call y::q(x, \ldots, x) \mid \text{uncall } y::q(x, \ldots, x)
                                                                                 (method invocation)
                \mathbf{skip} \mid s \mid s
                                                                                (statement sequence)
              \overline{n} \mid x \mid x[e] \mid nil \mid e \, \otimes \, e
                                                                                           (expression)
    e
                + | - | ^
                                                                                             (operator)
   \odot
               ⊙ | * | / | % | & | | | && | | | | < | > | = | != | <= | >=
                                                                                             (operator)
                                      Syntax Domains
      prog \in Programs
                                         s \in Statements
                                                                         n \in Constants
         cl \in Classes
                                         e \in \text{Expressions}
                                                                          x \in VarIDs
          t \in \text{Types}
                                        \odot \in ModOps
                                                                          q \in MethodIDs
```

Figure 2.2: Syntax domains and EBNF grammar for ROOPL++

 $\otimes \in \text{Operators}$ 

The ROOPL++ grammar extends the grammar of ROOPL with a new static integer or class array type and a new object lifetime option in form of objects outside of blocks, using the **new** and

 $m \in Methods$ 

 $c \in \text{ClassIDs}$ 

delete approach. Furthermore, the local block extension proposed in [11] has become a standard part of the language. Class definitions remains unchanged, and consists of a class keyword followed by a class name. Subclasses must be specified using the **inherits** keyword and a following parent class name. Classes can have any number of fields of any of the data types, including the new Array type. A class definition is required to include at least one method, defined by the **method** keyword followed by a method name, a comma-separated list of parameters and a body.

Reversible assignments for integer variables and integer array elements uses similar syntax as Janus assignments, by updating a variable through any of the addition (+=), subtraction (-=) or bitwise XOR  $(\hat{\ }=)$  operators. As with Janus, when updating a variable x using any of said operators, the right-hand side of the operator argument must be entirely independent of x to maintain reversibility. Usage of these reversible assignment operators for object or array variables is undefined. Variables and array elements of any type can be swapped using the <=> operator as long as the variable is of same type as the array type. If an array is of a base class type, subclass variable values can be swapped in and out of the array, as long as the resulting value in the variable is still of the original subclass type.

ROOPL++ objects can be instantiated in two ways. Either using object blocks known from ROOPL, or by using the **new** statement. The object-blocks have a statically-scoped lifetime, as the object only exists within the **construct** and **destruct** segments. Using **new** allows the object to live until program termination, if the program terminates with a **delete** call. By design, it is the programmers responsibility to deallocate objects instantiated by the **new** statement.

Arrays are also instantiated by usage of **new** and **delete**. Assignment of array cells depend on the type of the arrays, which is further discussed in section 2.4.

The methodologies for argument aliasing and its restrictions on method on invocations from ROOPL carries over in ROOPL++ and object fields are as such disallowed as arguments to local methods to prevent irreversible updates and non-local method calls to a passed objects are prohibited. The parameter passing scheme remains call-by-reference and the object model of ROOPL remains largely unchanged in ROOPL++.

### 2.2 Object Instantiation

Object instantiation through the **new** statement follows the pattern of the mechanics known from the **construct**/**destruct** blocks from ROOPL, but providing improved scoping and lifetime options objects. The mechanisms of the statement

#### construct $c \ x - s -$ destruct x

are as follows:

- 1. Memory for an object of class c is allocated. All fields are automatically zero-initialized by virtue of residing in already zero-cleared memory.
- 2. The block statement s is executed, with the name x representing a reference to the newly allocated object.

- 3. The reference x may be modified by swapping its value with that of other references of the same type, but it should be restored to its original value within the statement block s, otherwise the meaning of the object block is undefined.
- 4. Any state that is accumulated within the object should be cleared or uncomputed before the end of the statement is reached, otherwise the meaning of the object block is undefined.
- 5. The zero-cleared memory is reclaimed by the system.

The statement pair consisting of

#### $\mathbf{new} \ c \ x \qquad \dots \qquad \mathbf{delete} \ c \ x$

could be considered a *dynamic* block, meaning we can have overlapping blocks. Compared to  $\mathbf{construct/destruct}$  block consisting of a single statement, the  $\mathbf{new/delete}$  block consist of two separate statements. We can as such initialize an object x of class c and an object y of class d and destroy x before we destroy y, a feature that was not possible in ROOPL. The mechanisms of the  $\mathbf{new}$  statement are as follows:

- 1. Memory for an object of class c is allocated. All fields are automatically zero-initialized by virtue of residing in already zero-cleared memory.
- 2. The address of the newly allocated block is stored in the previously defined and zero-cleared reference x.

and the mechanisms of the delete statement are as follow

- 1. The reference x may be modified by swapping its internal field values with that of other references of the same type, but should be zero-cleared before a **delete** statement is called on x, otherwise the meaning of the object deletion is undefined.
- 2. Any state that is accumulated within the object should be cleared or uncomputed before the **delete** statement is executed, otherwise the meaning of the object block is undefined.
- 3. The zero-cleared memory is reclaimed by the system.

The mechanisms of the **new** and **delete** statements are, essentially, a split of the mechanisms of the **construct**/**destruct** blocks into two separate statements. As with ROOPL, fields must be zero-cleared after object deletion, otherwise it is impossible for the system to reclaim the memory reversibly. This is the responsibility of the of the programmer to maintain this, and to ensure that objects are indeed deleted in the first place. A **new** statement without a corresponding **delete** statement targeting the same object further ahead in the program is undefined, as is a delete statement without a preceding **new** statement.

Note that variable scopes are always static, but object scopes can be either static (using construct/destruct) or dynamic (using new/delete).

### 2.3 Array Model

Besides asymmetric object lifetimes, ROOPL++ also introduces reversible, fixed-sized arrays of either integer or object types. While ROOPL only featured integers and custom data types in form of classes, one of its main inspirations, JANUS, implemented static, reversible arrays [31].

While ROOPL by design did not include any data storage language constructs, as they are not especially noteworthy nor interesting from an OOP perspective, they do generally improve the expressiveness of the language. Arrays were decided to be part of the core language for this reason, as one of the main goals of ROOPL++ is increased expressiveness while implementing reversible programs.

In ROOPL++, arrays expand upon the array model from Janus. Arrays are indexed by integers, starting from 0. In Janus, only integer arrays were allowed, while in ROOPL++ arrays of any type can be defined, meaning either integer arrays or custom data types in form of class arrays. They are however, still restricted to one dimension.

Array element accessing is accomplished using the bracket notation known from Janus. Accessing an out-of-bounds index is undefined. Array instantiation and element assignments, aliasing and circularity is described in detail in the following section.

Arrays can contain elements of different classes sharing a base class, that is, say class A and B both inherit from some class C and array x is of type C[]. In this case, the array can hold elements of type A, B, and C. When swapping array elements from a base class array with object references, the programmer must be careful not to swap the values of, say, and A object into a B reference.

### 2.4 Array Instantiation

Array instantiation uses the **new** and **delete** keywords to reversibly construct and destruct array types. The mechanisms of the statement

new int
$$[e]$$
 x

in which we reserve memory for an integer array are as follows

- 1. The expression e is evaluated
- 2. Memory equal to the integer value that e evaluates to and an additional small amount of memory for of overhead is reserved for the array.
- 3. The address of the newly allocated memory is stored in the previously defined and zerocleared reference x.

In ROOPL++, we only allow instantiation of fixed-sized arrays of a length defined in the given expression e. Array elements are assigned dependent on the type of the array. For integer arrays, any of the reversible assignment operators can be used to assign values to cells. For class arrays, we assign cell elements a little differently. We either make use of the **new** and **delete** statements, but instead of specifying which variable should hold the newly created/deleted object or array,

we specify which array cell it should be stored in or we use the **swap** statement to swap values in and out of array cells. Usage of the assignment operators on non-integer arrays is undefined.

```
1
      new int[5] intArray
                                   // Init new integer array
      new Foo[2] fooArray
2
                                    // init new Foo array
      intArray[1] += 10
                                   // Legal array integer assignment
4
      intArray[1] -= 10
                                    // Legal Zero-clearing for integer array cells
5
6
      new Foo fooObject
7
                                   // Legal object array cell assignment
8
      fooArray[0] <=> fooObject
9
      new Foo fooArray[2]
                                   // Legal object array cell assignment
10
                                    // Clear all array cells
11
12
      delete Foo fooArray[0]
                                    // Legal object array cell zero-clearing
13
14
       delete Foo fooArray[1]
                                    // Legal object array cell zero-clearing
```

Listing 2.1: Assignment of array elements

As with ROOPL++ objects instantiated outside of **construct**/**destruct** blocks, arrays must be deleted before program termination to reversibly allow the system to reclaim the memory. Before deletion of an array, all its elements must be zero-cleared such that no garbage data resides in memory after erasure of the array reference.

Consider the statement

delete 
$$int[e] x$$

with the following mechanics

- 1. The reference x may be modified by swapping, assigning cell element values and zero-clearing cell element values, but must be restored to an array of same type with fully zero-cleared cells before the **delete** statement. Otherwise, the meaning of the statement is undefined.
- 2. The value of the expression e is evaluated and used to reclaim the allocated memory space.
- 3. If the reference x is a fully zero-cleared array upon the **delete** statement execution, the zero-cleared memory is reclaimed by the system.

With reversible, fixed-sized arrays of varying types, we must be extremely careful when updating and assigning values, to ensure we maintain reversibility and avoid irreversible statements. Therefore, when assigning or updating integer elements with one of the reversible assignment operators, we prohibit the cell value from being reference on the right hand side, meaning the following statement is prohibited

$$x[5] += x[5] + 1$$

However, we do allow other initialized, non-zero-cleared array elements from the same array or arrays of same type to be referenced in the right hand side of the statement. As with regular assignment, we still prohibit the left side reference to occur in the ride side, meaning the following statements are also prohibited

$$x += y[x]$$
$$y[x] += x$$

### 2.5 Referencing

Besides the addition of dynamically lifetimed objects and arrays, ROOPL++ also increases program flexibility by allowing multiple references to objects and arrays through the usage of the **copy** statement. Once instantiated through either a **new** or **construct/destruct** block, an object or array reference can be copied into another zero-cleared variable. The reference acts as a regular instance and can be modified through methods as per usual. To delete a reference, the logical inverse statement **uncopy** must be used.

The syntax for referencing consists of the statement

$$copy \ c \ x \ x'$$

which copies a reference of variable x, an instance of class or array c, and stores the reference in variable x'.

For deleting copies, the following statement is used

uncopy 
$$c \ x \ x'$$

which simply zero-clears variable x', which is a reference to variable x, an instance of class or array c.

The mechanism of the **copy** statement is simply as follows

1. The memory address stored in variable x is copied into the zero-cleared variable x'. If x' is not zero-cleared or x is not a class instance, then **copy** is undefined.

The mechanism of the **uncopy** statement is simply as follows

1. The memory address stored in variable x' is zero-cleared if it matches the address stored in x. If x' is not a copy of x or x has been zero-cleared before the **uncopy** statement is executed, said statement is undefined.

As references do not require all fields or cells to be zero-cleared (as they are simple pointers to existing objects or arrays), the reversible programmer should carefully ensure that all references are un-copied before deleting said object or array, as copied references to cleared objects or arrays would be pointing to cleared memory, which might be used later by the system. These type of references are also known as *dangling pointers*.

It should be noted, that from a language design perspective, it is the programmer's responsibility to ensure such situations do not occur. From an implementation perspective, such situations are usually checked by the compiler either statically during compilation or during the actual runtime of the program. This is addressed later in sections 3.3 and 4.9.

#### 2.6 Local Blocks

The local block presented in the extended Janus in [29] consisted of a local variable allocation, a statement and a local variable deallocation. These local variable blocks add immense programmer usability as the introduce a form of reversible temporary variable. The ROOPL compiler features support for local integer blocks, but not object blocks. In ROOPL++, local blocks can be instantiated with all of the languages variable types; integers, arrays and user-defined types in the form of objects.

Local integer blocks works exactly the same as in ROOPL and JANUS, where the local variable initialized will be set to the evaluated result of a given expression.

Local array and object blocks feature a number of different options. If a local array or object block is initialized with a **nil** value, the variable must afterwards be initialized using a new statement before any type-specific functionality is accessible. If the block is initiated with an existing object or array reference, the local variable essentially becomes a reference copy, analogous to a variable initialized from a **copy** statement.

Figure 2.3: construct/destruct-blocks can be considered syntactic sugar

For objects, the **construct**/**destruct**-blocks can be considered syntactic sugar for a local block defined with a **nil** value, containing a **new** statement in the beginning of its statement block and a **delete** statement in the very end, as shown in figure 2.3.

As local array and object blocks allow freedom in terms of their interaction with other statements in the language, it is the programmer's responsibility that the local variable is deallocated using a correct expression at the end of the block definition. The value of the variable is a pointer to an object or an array. Said object or array must have all fields/cells zero-cleared before the pointer is zero-cleared at the end of the local block. If the pointer is at any point exchanged with the pointer of another object or array using the **swap** statement, the same conditions apply.

### 2.7 ROOPL++ Expressiveness

By introducing dynamic lifetime objects and by allowing objects to be referenced multiple times, we can express non-trivial reversible programs. To demonstrate the capacities, expressiveness and possibilities of ROOPL++, the following section presents previously unseen reversible data structures, which now are feasible, written in ROOPL++.

#### 2.7.1 Linked List

Haulund presented a linked list implemented in ROOPL in [11]. The implementation featured a ListBuilder and a Sum class, required to determine and retain the sum of a constructed linked list as the statically scoped object blocks of ROOPL would deallocate automatically after building the full list. In ROOPL++, we do not face the same challenges and the implementation becomes much more straightforward. Figure 2.5 implements a LinkedList class, which simply has the head of the list and the list length as its internal fields. For demonstration, the class allows extension of the list by either appending or prepending cell elements to the list. In either case, we first check if the head field is initialized. If not, the cell we are either appending or prepending simply becomes the new head of the list. If we are appending a cell the Cell-class append method is called on the head cell with the new cell as its only argument. When prepending, the existing head is simply appended to the new cell and the new cell is set as head of the linked list.

```
class Cell
           Cell next
2
3
           int data
 4
           method constructor(int value)
5
6
               data ^= value
7
           method append(Cell cell)
8
9
               if next = nil & cell != nil then
                   next <=> cell
10
                                             // Store as next cell if current cell is end of list
11
               else skip
               fi next != nil & cell = nil
13
               if next != nil then
14
                   call next::append(cell) // Recursively search until we reach end of list
15
               else skip
16
               fi next != nil
17
```

Figure 2.4: Linked List cell class

Figure 2.4 shows the *Cell* class of the linked list which has a *next* and a *data* field, a constructor and the *append* method. The append method works by recursively looking through the linked cell nodes until we reach the end of the free list, where the *next* field has not been initialized yet. When we find such a cell, we simply swap the contents of the *next* and *cell* variables, s.t. the cell becomes the new end of the linked list.

An interesting observation is that the *append* method is called an additional time *after* setting the cell as the new end of the linked list. In a non-reversible programming language, we would simply call append in the else-branch of the first conditional. In the reversible setting, this is not an option, as the append call would modify the value of the *next* and *cell* variables and as

```
class LinkedList
          Cell head
2
3
          int listLength
4
          method insertHead(Cell cell)
5
               if head = nil & cell != nil then
6
                   head <=> cell
                                                 // Set cell as head of list if list is empty
7
               else skip
8
               fi head != nil & cell = nil
10
          method appendCell(Cell cell)
11
               call insertHead(cell)
                                                 // Insert as head if empty list
12
13
14
               if head != nil then
15
                   call head::append(cell)
                                                 // Iterate until we hit end of list
               else skip
16
17
               fi head != nil
18
19
               listLength += 1
                                                 // Increment length
20
          method prependCell(Cell cell)
21
22
               call insertHead(cell)
                                                 // Insert as head if empty list
23
               if cell != nil & head != nil then
24
                   call cell::append(head)
                                                 // Set cell.next = head. head = nil after execution
25
               else skip
26
               fi cell != nil & head = nil
27
               if cell != nil & head = nil then
29
                                                 // Set head = cell. Cell is nil after execution
30
                   cell <=> head
               else skip
31
               fi cell = nil & head != nil
32
33
34
               listLength += 1
                                                 // Increment length
35
36
           method length(int result)
               result ^= listLength
37
```

Figure 2.5: Linked List class

such, corrupt the control flow as the exit condition would be true after executing both the thenand else-branch of the conditional. To avoid this, we simply call one additional time with a **nil** value *cell*. This "wasted" additional call with a **nil** value is a recurring technique in the following presented reversible data structure implementations.

#### 2.7.2 Binary Tree

Figures 2.6, 2.8 and 2.7 shows the implementation of a binary tree in form of a rooted, unbalanced, min-heap. The *Tree* class shown in figure 2.6 has a single root node field and the three methods insertNode, sum and mirror. For insertion, the insertNode method is called from the root, if it is initialized and if not, the passed node parameter is simply set as the new root of the tree. The insertNode method implemented in the Node class shown in figure 2.8 first determines if we need to insert left or right but checking the passed value against the value of the current node. This is done recursively, until an uninitialized node in the correct subtree has been found. Note that as a consequence of reversibility, the value of node we wish to insert must be passed separately in the method call as we otherwise cannot zero-clear it after swapping the node we are inserting

with either the right or left child of the current cell.

```
class Tree
2
3
           method insertNode(Node node, int value)
               if root = nil & node != nil then
5
                   root <=> node
6
               else skip
               fi root != nil & node = nil
8
9
               if root != nil then
10
                   call root::insertNode(node, value)
11
12
               else skip
               fi root != nil
13
14
           method sum(int result)
15
               if root != nil then
16
17
                   call root::getSum(result)
18
               else skip
               fi root != nil
19
20
21
           method mirror()
               if root != nil then
22
                   call root::mirror()
23
               else skip
24
25
               fi root != nil
```

Figure 2.6: Binary Tree class

Summing and mirroring the tree works in a similar fashion by recursively iterating each node of the tree. For summing we simply add the value of the node to the sum and for mirroring we swap the children of the node and then recursively swap the children of the left and right node, if initialized. The sum and mirror methods are implemented in figure 2.7.

```
method getSum(int result)
          result += value
2
                                             // Add the value of this node to the sum
3
           if left != nil then
4
               call left::getSum(result)
                                            // If we have a left child, follow that path
5
6
           else skip
                                            // Else, skip
          fi left != nil
7
           if right != nil then
9
               call right::getSum(result) // If we have a right child, follow that path
10
11
           else skip
                                            // Else, skip
12
           fi right != nil
13
      method mirror()
14
                                            // Swap left and right children
          left <=> right
15
16
          if left = nil then skip
17
          else call left::mirror()
                                            // Recursively swap children if left != nil
18
19
           fi left = nil
20
21
          if right = nil then skip
22
           else call right::mirror()
                                            // Recursively swap children if right != nil
           fi right = nil
23
```

Figure 2.7: Binary Tree node class (cont)

```
class Node
           Node left
2
3
           Node right
4
           int value
5
6
           method setValue(int newValue)
7
               value ^= newValue
8
           method insertNode(Node node, int nodeValue)
9
10
               // Determine if we insert left or right
11
               if nodeValue < value then</pre>
                    if left = nil & node != nil then
12
                        // If open left node, store here
13
14
                        left <=> node
15
                    else skip
                    fi left != nil & node = nil
16
17
                    if left != nil then
18
19
                        // If current node has left, continue iterating
                        call left::insertNode(node, nodeValue)
20
21
                    else skip
22
                    fi left != nil
23
               else
                    if right = nil & node != nil then
24
                        // If open right node spot, store here
25
26
                        right <=> node
27
                    else skip
                    fi right != nil & node = nil
28
29
30
                    if right != nil then
                        // If current node has, continue searching
31
                        call right::insertNode(node, nodeValue)
32
33
                    else skip
34
                    fi right != nil
35
               fi nodeValue < value
```

Figure 2.8: Binary Tree node class

#### 2.7.3 Doubly Linked List

Finally, we present the reversible doubly linked list, shown in figures 2.10-2.12. A *cell* in a doubly linked list contains a reference to itself named *self*, a reference to its left and right neighbours, a data and an index field. As with the linked list and binary tree implementation the *DoubleLinkedList* class has a field referencing the head of the list and its *appendCell* method is identical to the one of the linked list.

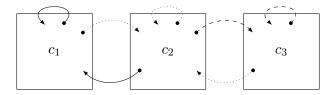


Figure 2.9: Multiple identical reference are needed for a doubly linked list implementation

This data structure is particularly interesting, as it, unlike the former two presented structures, cannot be expressed in ROOPL, as this requires multiple reference to objects, in order for an object to point to itself and to its left and right neighbours. Figure 2.9 shows the multiple

references needed for the doubly linked list implementation denoted by the three different arrow types.

```
1
       class DoublyLinkedList
           Cell head
           int length
3
4
           method appendCell(Cell cell)
5
               if head = nil & cell != nil then
6
7
                   head <=> cell
               else skip
8
               fi head != nil & cell = nil
9
10
               if head != nil then
11
12
                   call head::append(cell)
13
               else skip
               fi head != nil
14
               length += 1
16
```

Figure 2.10: Doubly Linked List class

When we append a cell to the list, we first search recursively through the list until we are at the end. The new cell is then set as *right* of the current cell. A reference to the current self is created using the **copy** statement, and set as *left* of the new end of the list, thus resulting in the new cell being linked to list and now acting as end of the list.

```
class Cell
           int data
 3
           int index
 4
           Cell left
           Cell right
 5
 6
           Cell self
           method setData(int value)
 8
 9
                data ^= value
10
           method setIndex(int i)
11
                index ^= i
13
           method setLeft(Cell cell)
14
               left <=> cell
15
16
17
           method setRight (Cell cell)
               right <=> cell
18
19
20
           method setSelf(Cell cell)
21
                self <=> cell
```

Figure 2.11: Doubly Linked List Cell class

The data structure could relatively easily be extended to work as a dynamic array. Currently each cell contains an index field, specifying their position in the list. If, say, we wanted to insert some new data at index n, without updating the existing value, but essentially squeezing in a new cell, we could add a method to the DoublyLinkedList class taking a data value and an index. When executing this method, we could iterate the list until we reach the cell with index n, construct a new cell instance, update required left and right pointers to insert the new cell at

```
method append (Cell cell)
2
          if right = nil & cell != nil then
                                                // If current cell does not have a right neighbour
               right <=> cell
3
                                                // Set new cell as right neighbour of current cell
4
5
               local Cell selfCopy = nil
               copy Cell self selfCopy
                                                // Copy reference to current cell
6
7
               call right::setLeft(selfCopy)
                                                // Set current as left of right neighbour
               delocal Cell selfCopy = nil
8
9
10
               local int cellIndex = index + 1
               call right::setIndex(cellIndex) // Set index in right neighbour of current
11
               delocal int cellIndex = index + 1
          else skip
13
          fi right != nil & cell = nil
14
15
          if right != nil then
16
17
               call right::append(cell)
                                                // Keep searching for empty right neighbour
          else skip
18
19
          fi right != nil
```

Figure 2.12: Doubly Linked List Cell class (cont)

the correct position, in such a way that the old cell at index n now is the new right neighbour of the cell and finally recursively iterating the list, incrementing the index of cells to the right of the new cell by one. In reverse, this would remove a cell from the list. If we want to update an existing value at a index, a similar technique could be used, where we iterate through the cells until we find the correct index. If we are given an index that is out of bounds in terms of the current length of the list, we could extend the tail on the list until reach a cell with the wanted index. When we are zero-clearing a value that is the furthest index, the inverse would apply, and a such we would zero-clear the cell, and the deallocate cells until we reach a cell which does not have a zero-cleared data field.

This extended doubly linked list would also allow lists of n-dimensional lists, as the type of the data field simply could be changed to, say, a FooDoublyLinkedList, resulting in an array of Foo arrays.

### 2.8 Type System

The type system of ROOPL++ expands on the type system of ROOPL presented by Haulund [11] and is analogously described by syntax-directed inference typing rules in the style of Winskel [28]. As ROOPL++ introduces two new types in form of references and arrays, a few ROOPL typing rules must be modified to accommodate these added types. For completeness all typing rules, including unmodified rules, are included in the following sections.

#### 2.8.1 Preliminaries

The types in ROOPL++ are given by the following grammar:

```
\tau ::= \text{int} \mid c \in \text{ClassIDs} \mid r \in \text{ReferenceIDs} \mid i \in \text{IntegerArrayIDs} \mid o \in \text{ClassArrayIDs}
```

The type environment  $\Pi$  is a finite map pairing variables to types, which can be applied to an identifier x using the  $\Pi(x)$  notation. Notation  $\Pi' = \Pi[x \mapsto \tau]$  defines updates and creation of a new type environment  $\Pi'$  such that  $\Pi'(x) = \tau$  and  $\Pi'(y) = \Pi(y)$  if  $x \neq y$ , for some variable identifier x and y. The empty type environment is denoted as [] and the function  $vars: Expressions \to VarIDs$  is described by the following definition

```
    \text{vars}(\bar{n}) &= \emptyset \\
    \text{vars}(\mathbf{nil}) &= \emptyset \\
    \text{vars}(x) &= \{ x \} \\
    \text{vars}(x[e]) &= \{ x \} \cup \text{vars}(e) \\
    \text{vars}(e_1 \otimes e_2) &= \text{vars}(e_1) \cup \text{vars}(e_2).
```

The binary subtype relation  $c_1 \prec : c_2$  is required for supporting subtype polymorphism and is defined as follows:

```
c_1 \prec: c_2 if c_1 inherits from c_2
c \prec: c (reflexivity)
c_1 \prec: c_3 if c_1 \prec: c_2 and c_2 \prec: c_3 (transitivity)
```

Furthermore, we formally define object models in such a way that inherited fields and methods are included, unless overridden by the derived fields. Therefore, we define  $\Gamma$  to be the class map of a program p, such that  $\Gamma$  is a finite map from class identifiers to tuples of methods and fields for the class p. Application of a class map  $\Gamma$  to some class cl is denoted as  $\Gamma(cl)$ . Construction of a class map is done through function gen, as shown in figure 2.13. Figure 2.14 defines the fields and methods functions to determine these given a class. Set operation  $\Theta$  defines method overloading by dropping base class methods if a similarly named method exists in the derived class. The definitions shown in Figure 2.13 and 2.14 are originally from [11].

$$\operatorname{gen}\left(\overbrace{cl_1, \dots, cl_n}^{p}\right) = \overbrace{\left[\alpha(cl_1) \mapsto \beta(cl_1), \dots, \alpha(cl_n) \mapsto \beta(cl_n)\right]}^{\Gamma}$$

$$\alpha\left(\operatorname{\mathbf{class}} c \dots\right) = c \qquad \beta(cl) = \left(\operatorname{fields}(cl), \operatorname{methods}(cl)\right)$$

Figure 2.13: Definition gen for constructing the finite class map  $\Gamma$  of a given program p, originally from [11]

$$\mathrm{fields}(cl) = \begin{cases} \eta(cl) & \text{if } cl \sim \text{ [class } c \text{ } \cdots \text{]} \\ \eta(cl) \cup \mathrm{fields} \left(\alpha^{-1}(c')\right) & \text{if } cl \sim \text{[class } c \text{ inherits } c' \text{ } \cdots \text{]} \end{cases}$$

$$\mathrm{methods}(cl) = \begin{cases} \delta(cl) & \text{if } cl \sim [\mathbf{class}\ c\ \cdots] \\ \delta(cl) \uplus \, \mathrm{methods} \Big(\alpha^{-1}(c')\Big) & \text{if } cl \sim [\mathbf{class}\ c\ \mathbf{inherits}\ c'\ \cdots] \end{cases}$$

$$A \ \uplus B \ \stackrel{def}{=} \ A \cup \left\{ m \in B \ \middle| \ \nexists \ m' \Big( \zeta(m') = \zeta(m) \wedge m' \in A \Big) \right\}$$
 
$$\zeta \Big( \mathbf{method} \ q \ (\cdots) \ s \Big) = q \qquad \eta \Big( \mathbf{class} \ c \ \cdots \ \overbrace{t_1 f_1 \ \cdots \ t_n f_n}^{fs} \ \cdots \Big) = fs$$
 
$$\delta \Big( \mathbf{class} \ c \ \cdots \ \underbrace{\mathbf{method}}_{ms} \ q_1 \ (\cdots) \ s_1 \ \cdots \ \mathbf{method}}_{ms} \ \cdots \Big) = ms$$

Figure 2.14: Definition of fields and methods, originally from [11]

Finally, we formally define a link between arrays of a given type and other types. The function array Type, defined in figure 2.15, is c if the passed array a is an array of class c instances.

$$\operatorname{arrayType}(a) = \begin{cases} c & \text{if } a \in ClassArrayIDs \text{ and } a \text{ is a } c \text{ array} \\ \mathbf{int} & \text{if } a \in i \end{cases}$$

Figure 2.15: Definition arrayType for mapping types of arrays to either class types or the integer type

#### 2.8.2 Expressions

The type judgment

$$\Pi \vdash_{expr} e : \tau$$

defines the type of expressions. The judgment reads as: under type environment  $\Pi$ , expression e has type  $\tau$ .

$$\frac{\Pi \vdash_{expr} n : \mathbf{int}}{\Pi \vdash_{expr} e_{1} : \mathbf{int}} \text{ T-Con} \qquad \frac{\Pi(x) = \tau}{\Pi \vdash_{expr} x : \tau} \text{ T-Var} \qquad \frac{\tau \neq \mathbf{int}}{\Pi \vdash_{expr} \mathbf{nil} : \tau} \text{ T-Nill}$$

$$\frac{\Pi \vdash_{expr} e_{1} : \mathbf{int}}{\Pi \vdash_{expr} e_{1} \otimes e_{2} : \mathbf{int}} \text{ T-BinOpInt}$$

$$\frac{\Pi \vdash_{expr} e_{1} : c}{\Pi \vdash_{expr} e_{1} \otimes e_{2} : \mathbf{int}} \text{ T-BinOpObJ}$$

Figure 2.16: Typing rules for expressions in ROOPL, originally from [11]

The original expression typing rules from ROOPL are shown in figure 2.16. The type rules T-Con, T-Var and T-Nil defines typing of the simplest expressions. Numeric literals are of type int, typing of variable expressions depends on the type of the variable in the type environment and the nil literal is a non-integer type. All binary operations are defined for integers, while only equality-operators are defined for objects.

With the addition of the ROOPL++ array type, we extend the expression typing rules with rule T-ArrelemVar which defines typing for array element variables, shown in figure 2.17.

$$\frac{\Pi(x) = \tau[\ ] \qquad \Pi_{expr} \vdash e \ : \ \mathbf{int}}{\Pi \vdash_{expr} x[e] : \tau} \text{T-ArrelemVar}$$

Figure 2.17: Typing rule extension for the ROOPL typing rules

#### 2.8.3 Statements

The type judgment

$$\overline{\langle \Pi, c \rangle} \vdash_{stmt}^{\Gamma} s$$

defines well-typed statements. The judgment reads as under type environment  $\Pi$  within class c, statement s is well-typed with class map  $\Gamma$ .

$$\frac{x \notin \text{vars}(e) \quad \Pi \vdash_{expr} e : \mathbf{int} \quad \Pi(x) = \mathbf{int}}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} x \odot = e} \text{T-AssVar}$$

$$\frac{\Pi \vdash_{expr} e_1 : \mathbf{int} \qquad \langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} s_1 \qquad \langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} s_2 \qquad \Pi \vdash_{expr} e_2 : \mathbf{int}}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{if} e_1 \mathbf{then} \ s_1 \mathbf{else} \ s_2 \mathbf{fi} \ e_2} \text{ T-IF}$$

$$\frac{\Pi \vdash_{expr} e_1 : \mathbf{int} \quad \langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} s_1 \quad \langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} s_2 \quad \Pi \vdash_{expr} e_2 : \mathbf{int}}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{from} \ e_1 \ \mathbf{do} \ s_1 \ \mathbf{loop} \ s_2 \ \mathbf{until} \ e_2} \text{ $\mathrm{T}$-Loop}}$$

$$\frac{\langle \Pi[x \mapsto c'], c \rangle \vdash^{\Gamma}_{stmt} s}{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \mathbf{construct} \ c' \ x \ s \ \mathbf{destruct} \ x} \text{ T-ObjBlock} \qquad \overline{\langle \Pi, c \rangle \vdash^{\Gamma}_{stmt} \mathbf{skip}} \text{ T-Skip}$$

$$\frac{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} s_1 \qquad \langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} s_2}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} s_1 s_2} \text{ T-SeQ} \qquad \frac{\Pi(x_1) = \Pi(x_2)}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} x_1 \iff x_2} \text{ T-SwpVar}$$

$$\frac{\Gamma(\Pi(c)) = \left(fields, \ methods\right) \quad \left(\mathbf{method} \ q(t_1 \ y_1, \ ..., \ t_n \ y_n) \ s\right) \in methods}{\{x_1, \ ..., \ x_n\} \cap fields = \emptyset \qquad i \neq j \implies x_i \neq x_j \qquad \Pi(x_1) \prec: t_1 \ \cdots \ \Pi(x_n) \prec: t_n \\ \hline \left\langle \Pi, c \right\rangle \vdash_{stmt}^{\Gamma} \mathbf{call} \ q(x_1, \ ..., \ x_n)} \text{T-Call}$$

$$\Gamma(\Pi(x_0)) = \begin{pmatrix} fields, \ methods \end{pmatrix} \begin{pmatrix} \mathbf{method} \ q(t_1 \ y_1, \ ..., \ t_n \ y_n) \ s \end{pmatrix} \in methods$$

$$\frac{i \neq j \implies x_i \neq x_j \qquad \Pi(x_1) \prec : t_1 \ \cdots \ \Pi(x_n) \prec : t_n}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{call} \ x_0 :: q(x_1, \ ..., \ x_n)} \text{ T-CallO}$$

$$\frac{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{call} \ q(x_1, \ ..., \ x_n)}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{uncall} \ q(x_1, \ ..., \ x_n)} \text{ T-UC} \qquad \frac{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{call} \ x_0 :: q(x_1, \ ..., \ x_n)}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{uncall} \ x_0 :: q(x_1, \ ..., \ x_n)} \text{ T-UCO}$$

Figure 2.18: Typing rules for statements in ROOPL, originally from [11]

Typing rule T-AssVar defines variable assignments for an integer variable and an integer expression result, given that the variable x does not occur in the expression e.

The type rules T-IF and T-LOOP defines reversible conditionals and loops as known from JANUS, where entry and exit conditions are integers and branch and loop statements are well-typed statements.

The object block, introduced in ROOPL, is only well-typed if its body statement is well-typed.

The **skip** statement is always well-typed, while a sequence of statements are well-typed if each of the provided statements are. Variable **swap** statements are well-typed if both operands are of the same type under type environment  $\Pi$ .

As with ROOPL, type correctness of local method invocation is defined in rule T-Call iff:

- The number of arguments matches the method arity
- No class fields are present in the arguments passed to the method (To prevent irreversible updates)
- The argument list contains unique elements
- Each argument is a subtype of the type of the equivalent formal parameter.

For foreign method invocations, typing rule T-CALLO. A foreign method invocation is well-typed using the same rules as for T-CALL besides having no restrictions on class fields parameters in the arguments, but an added rule stating that the callee object  $x_0$  must not be passed as an argument.

The typing rules T-UC and T-UCO defines uncalling of methods in terms of their respective inverse counterparts.

$$\Pi(x) = \operatorname{int}[\ ] \quad \Pi \vdash_{expr} e_1 : \operatorname{int}$$

$$\frac{\left( \ x \ \cup \operatorname{vars}(e_1) \right) \cap \operatorname{vars}(e_2) = \emptyset \qquad \Pi \vdash_{expr} e_2 : \operatorname{int}}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} x [e_1] \odot = e_2} \quad \text{T-Arrelemass}$$

$$\frac{\Pi(x) = c'}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \operatorname{new} c' x} \quad \text{T-ObJNeW} \qquad \frac{\Pi(x) = c'}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \operatorname{delete} c' x} \quad \text{T-ObJDLT}$$

$$\frac{\operatorname{arrayType}(a) \in \left\{ \operatorname{classIDs}, \operatorname{int} \right\} \quad \Pi \vdash_{expr} e = \operatorname{int} \quad \Pi(x) = a[\ ]}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \operatorname{new} a[e] \ x} \quad \text{T-ArrNeW}$$

$$\frac{\operatorname{arrayType}(a) \in \left\{ \operatorname{classIDs}, \operatorname{int} \right\} \quad \Pi \vdash_{expr} e = \operatorname{int} \quad \Pi(x) = a[\ ]}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \operatorname{delete} a[e] \ x} \quad \text{T-ArrDLT}$$

$$\frac{\Pi(x) = c' \quad \Pi(x') = c'}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \operatorname{copy} c' x \ x'} \quad \text{T-CP} \qquad \frac{\Pi(x) = c' \quad \Pi(x') = c'}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \operatorname{uncopy} c' x \ x'} \quad \text{T-UCP}$$

$$\frac{\langle \Pi, c \rangle \vdash_{expr}^{\Gamma} e_1 : c' \quad \langle \Pi[x \mapsto c'], c \rangle \vdash_{stmt}^{\Gamma} s \quad \langle \Pi, c \rangle \vdash_{expr}^{\Gamma} e_2 : c'}{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \operatorname{local} c' \ x = e_1 \quad s \quad \operatorname{delocal} c' \ x = e_2} \quad \text{T-LocalBlock}$$

Figure 2.19: Typing rules extensions for statements in ROOPL++

Figure 2.19 shows the typing rules for the extensions made to ROOPL in ROOPL++, covering the **new/delete** and **copy/uncopy** statements for objects and arrays and local blocks.

The typing rule T-Arrelemans defines assignment to integer array element variables, and is well-typed when the type of array x is **int**, the variable  $x[e_1]$  is not present in the right-hand

side of the statement, no variables in  $e_1$  exist in  $e_2$  and both expressions  $e_1$  and  $e_2$  evaluates to integers.

The T-OBJNEW and T-OBJDLT rules define well-typed **new** and **delete** statements for dynamically lifetimed objects. The **new** statement is well-typed, as long as  $c' \in \text{classIDs}$  and the variable x is of type c' under type environment  $\Pi$  and **delete** is also well-typed if the type of x under type environment  $\Pi$  is equal to c'.

The T-ARRNEW and T-ARRDLT rules define well-type **new** and **delete** statement for ROOPL++ arrays. The **new** statement is well-typed, if the type of the array either is a classID or **int**, the length expression evaluates to an integer and x is of of type a[] under the type environment  $\Pi$ , and **delete** is well-typed if the type of the array is either a classID or **int**, the length expression evaluates to an integer and x is equal to the array type a.

Typing rules T-CP and T-UCP define well-typed reference copy and un-copying statements. A well-typed **copy** or **uncopy** statement requires that the types of x and x' both are c' under type environment  $\Pi$ 

The rule T-LOCALBLOCK defines well-typed local blocks. A local block is well-typed if its two expression  $e_1$  and  $e_2$  are well-typed and its body statement s is well-typed.

### 2.8.4 Programs

As with ROOPL, a ROOPL++ program is well-typed if all of its classes and their respective methods are well-typed and if there exists a nullary main method. Figure 2.20 shows the typing rules for class methods, classes and programs.

$$\frac{\langle \Pi[x_1 \mapsto t_1, ..., x_n \mapsto t_n], c \rangle \vdash_{stmt}^{\Gamma} s}{\langle \Pi, c \rangle \vdash_{meth}^{\Gamma} \mathbf{method} q(t_1x_1, ..., t_nx_n) s} \text{ T-METHOD}$$

$$\Gamma(c) = \left( \overbrace{\left\{ \langle t_1, f_1 \rangle, \dots, \langle t_i, f_i \rangle \right\}}, \overbrace{\left\{ m_1, \dots, m_n \right\}} \right)$$

$$\Pi = \left[ f_1 \mapsto t_1, \dots, f_n \mapsto t_n \right] \quad \underbrace{\langle \Pi, c \rangle \vdash_{meth}^{\Gamma} m_1 \quad \cdots \quad \langle \Pi, c \rangle \vdash_{meth}^{\Gamma} m_n}_{\vdash_{class}^{\Gamma} c} \quad \text{T-CLASS}$$

$$\frac{\left(\text{method main ()}\ s\right) \in \bigcup_{i=1}^{n} \text{methods}(c_i)}{\Gamma = \text{gen}(c_1, \ \dots, \ c_n) \qquad \vdash_{class}^{\Gamma} c_1 \ \dots \ \vdash_{class}^{\Gamma} c_n}{\vdash_{prog} c_1 \ \dots \ c_n} \text{T-Prog}}$$

Figure 2.20: Typing rules for class methods, classes and programs, originally from [11]

### 2.9 Language Semantics

The following sections contain the operational semantics of ROOPL++, as specified by syntax-directed inference rules.

#### 2.9.1 Preliminaries

We define l to be a location. We define a location for integer variables to bind to a single location in program memory and a vector of memory locations for object and array variables, where the vector is the size of the object or array. A memory location is in the set of non-negative integers,  $\mathbb{N}_0$ . An environment  $\gamma$  is a partial function mapping variables to memory locations. A store  $\mu$  is a partial function mapping memory locations to values. An object is a tuple of a class name and an environment mapping fields to memory locations. A value is either an integer, an object, a location or a vector of locations.

Applications of environments  $\gamma$  and stores  $\mu$  are analogous to the type environment  $\Gamma$ , defined in section 2.8.1.

$$\begin{split} l \in \operatorname{Locs} &= \mathbb{N}_0 \\ \gamma \in \operatorname{Envs} &= \operatorname{VarIDs} \rightharpoonup \operatorname{Locs} \\ \mu \in \operatorname{Stores} &= \operatorname{Locs} \rightharpoonup \operatorname{Values} \\ \operatorname{Objects} &= \Big\{ \langle c_f, \ \gamma_f \rangle \mid c_f \in \operatorname{ClassIDs} \ \land \ \gamma_f \in \operatorname{Envs} \Big\} \\ v \in \operatorname{Values} &= \mathbb{Z} \cup \operatorname{Objects} \ \cup \operatorname{Locs} \cup [\operatorname{Locs}] \end{split}$$

Figure 2.21: Semantic values, originally from [11]

#### 2.9.2 Expressions

The judgment:

$$\langle \gamma, \mu \rangle \vdash_{expr} e \Rightarrow v$$

defines the meaning of expressions. We say that under environment  $\gamma$  and store  $\mu$ , expression e evaluates to value v.

$$\frac{\langle \gamma, \mu \rangle \vdash_{expr} n \Rightarrow \bar{n}}{\langle \gamma, \mu \rangle \vdash_{expr} e_1 \Rightarrow v_1} \xrightarrow{\langle \gamma, \mu \rangle \vdash_{expr} e_2 \Rightarrow v_2} VAR \xrightarrow{\langle \gamma, \mu \rangle \vdash_{expr} e_1 \Rightarrow 0} NIL$$

$$\frac{\langle \gamma, \mu \rangle \vdash_{expr} e_1 \Rightarrow v_1 \qquad \langle \gamma, \mu \rangle \vdash_{expr} e_2 \Rightarrow v_2 \qquad [\![\otimes]\!] (v_1, v_2) = v}{\langle \gamma, \mu \rangle \vdash_{expr} e_1 \otimes e_2 \Rightarrow v} BINOP$$

Figure 2.22: Semantic inference rules for expressions, originally from [11]

As shown in figure 2.22, expression evaluation has no effects on the store. Logical values are represented by *truthy* and *falsy* values of any non-zero value and zero respectively. The evaluation of binary operators is presented in figure 2.24.

$$\frac{\langle \gamma, \mu \rangle \vdash_{expr} e \Rightarrow v \qquad \gamma(x) = l \qquad \mu(l)[v] = l' \qquad \mu(l') = w}{\langle \gamma, \mu \rangle \vdash_{expr} x[e] \Rightarrow w} \text{ArrElem}$$

Figure 2.23: Extension to the semantic inference rules for expression in ROOPL++

For ROOPL++, we extend the expression ruleset with a single rule for array element variables shown in figure 2.23. As with the expressions inference rules in ROOPL, this extension has no effect on the store.

Figure 2.24: Definition of binary expression operator evaluation, originally from [11]

#### 2.9.3 Statements

The judgment

$$\gamma \vdash^{\Gamma}_{stmt} s : \mu \rightleftharpoons \mu'$$

defines the meaning of statements. We say that under environment  $\gamma$ , statement s with class map  $\Gamma$  reversibly transforms store  $\mu$  to store  $\mu'$ . Figure 2.25a, 2.25b and 2.25c defines the operational semantics of ROOPL++.

The following semantic rules have been simplified from the original ROOPL semantics [11] to better accommodate the extended language.

The inference rule SKIP defines the operational semantics of **skip** statements and has no effects on the store  $\mu$ .

$$\frac{\gamma \vdash_{stmt}^{\Gamma} \mathbf{skip} : \mu = \mu}{\gamma \vdash_{stmt}^{\Gamma} \mathbf{skip} : \mu = \mu} \overset{\text{SKIP}}{} = \frac{\gamma \vdash_{stmt}^{\Gamma} \mathbf{s}_1 : \mu = \mu'}{\gamma \vdash_{stmt}^{\Gamma} \mathbf{s}_1 : \mu = \mu''} \overset{\text{Y} \vdash_{stmt}^{\Gamma} \mathbf{s}_2 : \mu' = \mu''}{\gamma \vdash_{stmt}^{\Gamma} \mathbf{s}_1 : \mu = \mu''} \overset{\text{SEQ}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{stmt}^{\Gamma} \mathbf{s}_1 : \mu = \mu'}{\gamma \vdash_{stmt}^{\Gamma} \mathbf{s}_1 : \mu = \mu'} \overset{\text{Y} \vdash_{stmt}^{\Gamma} \mathbf{s}_2 : \mu' = \mu''}{\gamma \vdash_{stmt}^{\Gamma} \mathbf{s}_1 : \mu = \mu'} \overset{\text{SWVAR}}{} = \frac{\mu(\gamma(x_1)) = v_1}{\gamma \vdash_{stmt}^{\Gamma} \mathbf{s}_1 : \mu = \mu'} \overset{\text{Y} \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu' = \mu''}{\gamma \vdash_{stmt}^{\Gamma} \mathbf{s}_1 : \mu = \mu'} \overset{\text{Y} \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu' = \mu''}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu} \overset{\text{LoopMain}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_2 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu' = \mu''} \overset{\text{LoopBase}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu''} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu''} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu''} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_1 : \mu = \mu' \quad \langle \gamma, \mu' \rangle \vdash_{expr}^{\Gamma} e_2 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu''} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu''} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu''} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu'} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu'} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu'} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu'} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu'} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1, s_1, s_2, e_2) : \mu = \mu'} \overset{\text{LoopRec}}{} = \frac{\langle \gamma, \mu \rangle \vdash_{expr}^{\Gamma} e_1 \Rightarrow 0}{\gamma \vdash_{loop}^{\Gamma} (e_1,$$

Figure 2.25a: Semantic inference rules for statements, modified from [11]

Rule SEQ defines statement sequences where the store potentially is updated between each statement execution.

Rule AssVar defines reversible assignment in which variable identifier x under environment  $\gamma$  is mapped to the value v' resulting in an updated store  $\mu'$ . For variable swapping SWPVAR defines how value mappings between two variables are exchanged in the updated store.

For loops and conditionals, Rules LOOPMAIN, LOOPBASE and LOOPREC define the meaning of loop statements and IfTrue and IfFalse, similarly to the operational semantics of Janus, as

$$\gamma(this) = l \quad \mu(l) = l' \quad \mu(l') = \left\langle c, (l_1, \dots, l_m) \right\rangle \quad \gamma(y_i) = l'_i$$

$$\Gamma(c) = \left\langle \overbrace{(x_1, \dots, x_m)}, \overbrace{(\dots, \text{methods})} q(t_1 z_1, \dots, t_l z_k) \ s, \dots \right\rangle$$

$$\gamma' = [this \mapsto l, x_1 \mapsto l_1, \dots, l_m \mapsto v_m, z_1 \mapsto l'_1, \dots, z_k \mapsto l'_k] \quad \gamma' \vdash_{stmt}^{\Gamma} s : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ uncall } q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma(x_0) = l \quad \mu(l) = l' \quad \mu(l') = \left\langle c, (l_1, \dots, l_m) \right\rangle \quad \gamma(y_i) = l'_i$$

$$\gamma' = [this \mapsto l, x_1 \mapsto l_1, \dots, l_m \mapsto v_m, z_1 \mapsto l'_1, \dots, z_k \mapsto l'_k] \quad \gamma' \vdash_{stmt}^{\Gamma} s : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : \mu \rightleftharpoons \mu'$$

$$\gamma \vdash_{stmt}^{\Gamma} \text{ call } x_0 :: q(y_1, \dots, y_n) : q($$

Figure 2.25b: Semantic inference rules for statements, modified from [11] (cont)

presented in [29]. LOOPMAIN is entered if  $e_1$  is true and each iteration enters LOOPREC until  $e_2$  is false, in which case LOOPBASE is executed. Similarly, if  $e_1$  and  $e_2$  are true, rule IFTRUE is entered, executing the then-branch of the conditional. If  $e_1$  and  $e_2$  are false, the IFFALSE rule is executed and the else-branch is executed.

Rules Call, Uncall, Callobj and Uncallobj respectively define local and non-local method invocations. For local methods, method q in current class c should be of arity n matching the number of arguments. The updated store  $\mu'$  is obtained after statement body execution in the object environment. As local uncalling is the inverse of local calling, the direction of execution is simply reversed, and as such the input store a **call** statement serves as the output store of the

$$\langle \gamma, \mu \rangle \vdash_{stmt}^{\Gamma} e_1 \Rightarrow v_1 \quad \langle \gamma, \mu \rangle \vdash_{stmt}^{\Gamma} e_2 \Rightarrow v_2$$

$$\underline{\gamma(x) = l \quad \mu(l)[v_1] = l' \quad \mu(l') = w \quad [\![\odot]\!](w, v_2) = w'} \quad \text{AssArrElemVar}$$

$$\underline{\gamma \vdash_{stmt}^{\Gamma} x[e_1] \odot = e_2 : \mu \Rightarrow \mu[l' \mapsto w']} \quad \text{AssArrElemVar}$$

$$\underline{\Gamma(c) = \left\langle (x_1, \dots, x_m), methods \right\rangle} \quad \gamma(x) = l \quad l_0 \notin \text{dom}(\mu) \dots l_m \notin \text{dom}(\mu)$$

$$\underline{\gamma \vdash_{stmt}^{\Gamma} \text{new } c \ x : \mu[l \mapsto 0] \Rightarrow \mu \left[ l \mapsto l_0, l_0 \mapsto \left\langle c, (l_1, \dots, l_m) \right\rangle, l_1 \mapsto 0, \dots, l_m \mapsto 0 \right]} \quad \text{OBJN}$$

$$\underline{\gamma \vdash_{stmt}^{\Gamma} \text{new } c \ x : \mu[l \mapsto 0] \Rightarrow \mu \left[ l \mapsto l_0, l_0 \mapsto \left\langle c, (l_1, \dots, l_m) \right\rangle, l_1 \mapsto 0, \dots, l_m \mapsto 0 \right]} \quad \text{OBJDELETE}$$

$$\underline{\langle \gamma, \mu \rangle \vdash_{stmt}^{\Gamma} \text{new } a[e] \ x : \mu \Rightarrow \mu'} \quad \text{OBJDELETE} } \quad \underline{\langle \gamma, \mu \rangle \vdash_{stmt}^{\Gamma} \text{new } a[e] \ x : \mu \Rightarrow \mu[l \mapsto l', l' \mapsto 0^n]} \quad \text{ArrNew}$$

$$\underline{\gamma \vdash_{stmt}^{\Gamma} \text{delete } a[e] \ x : \mu \Rightarrow \mu'} \quad \text{ArrDelete}$$

$$\underline{\gamma \vdash_{stmt}^{\Gamma} \text{delete } a[e] \ x : \mu \Rightarrow \mu'} \quad \text{ArrDelete}$$

$$\underline{\gamma \vdash_{stmt}^{\Gamma} \text{delete } a[e] \ x : \mu \Rightarrow \mu'} \quad \text{Copy}$$

$$\underline{\gamma \vdash_{stmt}^{\Gamma} \text{copy } c \ x \ x' : \mu[l' \mapsto 0] \Rightarrow \mu[l' \mapsto v]} \quad \text{Copy}$$

$$\underline{\langle l, \gamma \rangle \vdash_{stmt}^{\Gamma} \text{copy } c \ x \ x' : \mu \Rightarrow \mu'} \quad \text{Uncopy}$$

$$\underline{\langle l, \gamma \rangle \vdash_{stmt}^{\Gamma} \text{copy } c \ x \ x' : \mu \Rightarrow \mu'} \quad \text{Uncopy}$$

$$\underline{\langle \gamma, \mu \rangle \vdash_{stmt}^{\Gamma} \text{e}_1 \Rightarrow v_1} \quad \langle \gamma, \mu' \rangle \vdash_{stmt}^{\Gamma} e_2 \Rightarrow v_2$$

$$\underline{r \notin \text{dom}(\mu)} \quad \gamma[x \mapsto r] \vdash_{stmt}^{\Gamma} s : \mu[r \mapsto v_1] \Rightarrow \mu'[r \mapsto v_2] \quad \text{LocalBlock}$$

Figure 2.25c: Extension to the semantic inference rules for statements in ROOPL++

uncall statement, similarly to techniques presented in [31, 29].

The statically scoped object blocks are defined in rule OBJBLOCK. The operation semantics of these blocks are similar to **local**-blocks from JANUS. We add the reference x to a new environment and afterwards map the location of x to the object tuple at location  $l_0$ , containing the locations of all object fields, all of which, along with  $l_0$ , must be unused in  $\mu$ . The result store  $\mu''$  is obtained after executing the body statement s in store  $\mu'$  mapping x to object reference at  $l_0$ , as long as all object fields are zero-cleared in  $\mu'''$  afterwards. If any of these conditions fail, the object block statement is undefined.

Figure 2.25c shows the extensions to the semantics of ROOPL with rules for new/delete and

copy/uncopy statements, array element assignment and local blocks.

Rule AssArrElemVar defines reversible assignment to array elements. After evaluating expressions  $e_1$  to  $v_1$  and  $e_2$  to  $v_2$ , the value at the location of variable  $x[v_1]$  under environment  $\gamma$  is mapped to the value  $v_3$  resulting in an updated store  $\mu'$ .

Dynamic object construction and destruction is defined by rules ObjNew and ObjDelete. For construction, x must be bound to a location l. We then make location l point to a new pair consisting of the class name and a vector of m new locations mapping object fields to locations. For destruction, x is still bound to l return l to a null pointer. As with object blocks, it is the program itself responsible for zero-clearing object fields before destruction. If the object fields are not zero-cleared, the ObjDelete statement is undefined.

Array construction and destruction is very similar to object construction and destruction. The major difference is we bind the location to a vector of size equal to the evaluated expression result. For deletion, we return the location of x to a null pointer and remove the binding to the vector from the store.

Object and array referencing is defined by rules COPY and UNCOPY. A reference is created and a new store  $\mu'$  obtained by mapping x' to the reference r which x current maps to, if c matches the tuple mapped to the location l. A reference is removed and a new store  $\mu'$  obtained if x and x' maps to the same reference r and x' then is removed from the store.

Local blocks are as previously mentioned, semantically similar to object blocks, where the memory location of variable x is mapped to an unused reference r in the store  $\mu$ . Before body statement execution, we let r bind to the evaluated value of  $e_1$ ,  $v_1$ . The result store after body statement execution,  $\mu'$  must have r mapped to the expression value of  $e_2$ ,  $v_2$ . r is then zero-cleared using the value of expression evaluation and becomes unused again.

### 2.9.4 Programs

The judgment

$$\vdash_{prog} p \Rightarrow \sigma$$

defines the meaning of programs. The class p containing the main method is instantiated and the main function is executed with the partial function  $\sigma$  as the result, mapping variable identifiers to values, correlating to the class fields of the main class.

$$\Gamma = \operatorname{gen}(c_1, \dots, c_n) \xrightarrow{fields} \Gamma(c_1) = \left( \overbrace{\{\langle t_1, f_1 \rangle, \dots, \langle t_n, f_n \rangle\}}, methods \right)$$

$$\left( \begin{array}{c} \left( \mathbf{method \ main} \ () \ s \right) \in methods \quad \gamma = [f_1 \mapsto 1, \ \dots, \ f_i \mapsto i] \\ \hline \mu = [1 \mapsto 0, \ \dots, \ i \mapsto 0, \ this \mapsto i+1, i+1 \mapsto \langle c_1, \gamma \rangle] \quad \gamma \vdash_{stmt}^{\Gamma} s \ : \ \mu \rightleftharpoons \mu' \\ \hline \vdash_{prog} c_1 \ \cdots \ c_n \Rightarrow (\gamma, \mu') \end{array} \right)$$
MAIN

Figure 2.26: Semantic inference rules for programs, originally from [11]

As with ROOPL programs, the fields of the main method in the main class c are bound in a new environment, starting at memory address 1, as 0 is reserved for **nil**. The fields are zero-initialized

in the new store  $\mu$  and address i+1 which maps to the new instance of c. After body execution, store  $\mu'$  is obtained.

# 2.10 Program Inversion

In order to truly show that ROOPL++ in fact is a reversible language, we must demonstrate and prove local inversion of statements is possible, such that any program written in ROOPL++, regardless of context, can be executed in reverse. Haulund presented a statement inverter for ROOPL in [11], which maps statements to their inverse counterparts. Figure 2.27 shows the statement inverter, extended with the new ROOPL++ statements for construction/destruction and referencing copying/copy removal.

```
\mathcal{I}\llbracket s_1 \ s_2 \rrbracket = \mathcal{I}\llbracket s_2 \rrbracket \ \mathcal{I}\llbracket s_1 \rrbracket
\mathcal{I}[skip] = skip
                                                                                        \mathcal{I}[x -= e] = x += e
\mathcal{I}[x += e] = x -= e
\mathcal{I}[x \triangleq e] = x \triangleq e
                                                                                        \mathcal{I}[x \iff e] = x \iff e
                                                                                        \mathcal{I}[x[e_1] -= e_2] = x[e_1] += e_2
\mathcal{I}[x[e_1] += e_2] = x[e_1] -= e_2
                                                                                        \mathcal{I}[x[e_1] \iff e_2] = x[e_1] \iff e_2
\mathcal{I}[x[e_1] \triangleq e_2] = x[e_1] \triangleq e_2
                                                                                        \mathcal{I}[\operatorname{copy} c \ x \ x'] = \operatorname{uncopy} c \ x \ x'
\mathcal{I}[\![\mathbf{new}\ c\ x]\!] = \mathbf{delete}\ c\ x
                                                                                        \mathcal{I}\llbracket \mathbf{uncopy} \ c \ x \ x' \rrbracket \ = \mathbf{copy} \ c \ x \ x'
\mathcal{I}[\![\mathbf{delete}\ c\ x]\!] = \mathbf{new}\ c\ x
\mathcal{I}[[\mathbf{call}\ q(\ldots)]] = \mathbf{uncall}\ q(\ldots)
                                                                                        \mathcal{I}[[\mathbf{call}\ x :: q(\ldots)]] = \mathbf{uncall}\ x :: q(\ldots)
\mathcal{I}[\mathbf{uncall}\ q(\ldots)] = \mathbf{call}\ q(\ldots)
                                                                                        \mathcal{I}[\mathbf{uncall}\ x :: q(\ldots)] = \mathbf{call}\ x :: q(\ldots)
\mathcal{I}[\mathbf{if}\ e_1\ \mathbf{then}\ s_1\ \mathbf{else}\ s_2\ \mathbf{fi}\ e_2]
                                                                                          = if e_1 then \mathcal{I}[s_1] else \mathcal{I}[s_2] fi e_2
\mathcal{I}[\mathbf{from}\ e_1\ \mathbf{do}\ s_1\ \mathbf{loop}\ s_2\ \mathbf{until}\ e_2]
                                                                                          = from e_1 do \mathcal{I}[s_1] loop \mathcal{I}[s_2] until e_2
\mathcal{I}[construct c \ x \ s \ destruct \ x]
                                                                                          = construct c \ x \ \mathcal{I}[s] destruct x
\mathcal{I}[\mathbf{local}\ t\ x\ = e \quad s \quad \mathbf{delocal}\ t\ x\ = e]
                                                                                          = local t x = e \mathcal{I}[s] delocal t x = e
```

Figure 2.27: ROOPL++ statement inverter, extended from [11]

Program inversion is conducted by recursive descent over components and statements. A proposed extension to the statement inverter for whole-program inversion is retained in the ROOPL++ statement inverter. The extension covers a case that reveals itself during method calling. As a method call is equivalent to an uncall with the inverse method we simply change calls to uncalls during inversion, the inversion of the method body cancels out. The proposed extension, presented in [31, 11], simply avoids inversion of calls and uncalls, as shown in figure 2.28.

### 2.10.1 Invertibility of Statements

While the invertibility of statements remains untouched by the extensions made in ROOPL++, the following proof, originally presented in [11], has been included for completeness.

$$\begin{split} \mathcal{I}' \llbracket \mathbf{call} \ q(\ldots) \rrbracket \ &= \mathbf{call} \ q(\ldots) \\ \mathcal{I}' \llbracket \mathbf{uncall} \ q(\ldots) \rrbracket \ &= \mathbf{call} \ q(\ldots) \\ \mathcal{I}' \llbracket \mathbf{uncall} \ q(\ldots) \rrbracket \ &= \mathbf{uncall} \ q(\ldots) \\ \mathcal{I}' \llbracket \mathbf{uncall} \ x :: q(\ldots) \rrbracket \ &= \mathbf{uncall} \ x :: q(\ldots) \\ \mathcal{I}' \llbracket \mathbf{s} \rrbracket = \mathcal{I} \llbracket \mathbf{s} \rrbracket \end{split}$$

Figure 2.28: Modified statement inverter for statements, originally from [11]

If execution of a statement s in store  $\mu$  yields  $\mu'$ , then execution of the inverse statement,  $\mathcal{I}[\![s]\!]$  in store  $\mu'$  should yield  $\mu$ . Theorem 2.1 shows that  $\mathcal{I}$  is a statement inverter.

**Theorem 2.1.** (Invertibility of statements, originally from [11])

$$\overbrace{\langle l, \gamma \rangle \vdash_{stmt}^{\Gamma} s : \mu \rightleftharpoons \mu'}^{\mathcal{S}} \iff \overbrace{\langle l, \gamma \rangle \vdash_{stmt}^{\Gamma} \mathcal{I}[\![s]\!] : \mu' \rightleftharpoons \mu}^{\mathcal{S}'}$$

*Proof.* By structural induction on the semantic derivation of  $\mathcal{S}$  (omitted). It suffices to show that  $\mathcal{S} \implies \mathcal{S}'$ , as this can serve as proof of  $\mathcal{S}' \implies \mathcal{S}$ , as  $\mathcal{I}$  is an involution.

### 2.10.2 Type-Safe Statement Inversion

Given a well-typed statement, the statement inverter  $\mathcal{I}$  should always produce a well-typed, inverse statement in order to correctly support backwards determinism of injective functions. Theorem 2.2 describes this.

**Theorem 2.2.** (Inversion of well-typed statements, originally from [11])

$$\overbrace{\langle \Pi, \ c \rangle \ \vdash_{stmt}^{\Gamma} \ s} \ \implies \ \overline{\langle \Pi, \ c \rangle \ \vdash_{stmt}^{\Gamma} \ \mathcal{I}\llbracket s \rrbracket}$$

*Proof.* By structural induction on  $\mathcal{T}$ . Unmodified ROOPL statements retained in ROOPL++ has been omitted.

• Case  $\mathcal{T} =$ 

$$\underbrace{\frac{\mathcal{E}_{1}}{\prod(x) = \text{int}[]}}_{C_{2}}
\underbrace{\frac{\mathcal{E}_{1}}{\prod(x) = \text{int}[]}}_{C_{2}}
\underbrace{\frac{\mathcal{E}_{2}}{\prod(x \cup \text{vars}(e_{1})) \cap \text{vars}(e_{2}) = \emptyset}}_{C_{2}}
\underbrace{\frac{\mathcal{E}_{2}}{\prod(x \cup \text{vars}(e_{1})) \cap \text{vars}(e_{2}) = \emptyset}}_{T-\text{ArrElemAss}}$$

In this case, we have  $\mathcal{I}[x \odot = e] = x \odot' = e$ , for some  $\odot'$ . Therefore,  $\mathcal{T}'$  will also be a derivation of rule T-Arreleman, and as such, we can simply reuse the conditions  $\mathcal{C}_1, \mathcal{C}_2$ 

and the expressions  $\mathcal{E}_1.\mathcal{E}_2$  in construction of  $\mathcal{T}'$ 

$$\mathcal{T}' = \frac{\overbrace{\Pi(x) = \mathbf{int}[\ ]}^{\mathcal{C}_1}}{\prod \vdash_{expr} e_1 : \mathbf{int}} \underbrace{\left(\begin{array}{c} \mathcal{C}_2 \\ x \cup \mathrm{vars}(e_1) \end{array}\right) \cap \mathrm{vars}(e_2) = \emptyset} \underbrace{\begin{array}{c} \mathcal{E}_2 \\ \Pi \vdash_{expr} e_2 : \mathbf{int} \end{array}}_{\left(\begin{array}{c} \mathcal{C}_2 \\ x \cup \mathrm{vars}(e_1) \end{array}\right) \cap \left(\begin{array}{c} \mathcal{C}_2 \\ x \cup \mathrm{vars}(e_2) \end{array}\right)}_{\left(\begin{array}{c} \mathcal{C}_2 \\ \Pi \vdash_{expr} e_2 : \mathbf{int} \end{array}\right)}$$

• Case 
$$\mathcal{T} = \underbrace{\frac{\mathcal{C}_1}{\Pi(x) = c'}}_{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{new} \ c' \ x} \text{T-ObjNew}$$

In this case we have  $\mathcal{I}[\![\mathbf{new}\ c\ x]\!] = \mathbf{delete}\ c\ x$ , meaning  $\mathcal{T}'$  must be of the form:

$$\mathcal{T}' = \underbrace{\frac{C_2}{\Pi(x) = c'}}_{\substack{\zeta\Pi,c\rangle \vdash_{stmt}^{\Gamma} \text{ delete } c' \ x}}$$

• Case 
$$\mathcal{T} = \underbrace{\frac{\mathcal{C}_1}{\Pi(x) = c'}}_{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{delete} \ c' \ x} \text{T-ObjDlt}$$

Inverse of the previous case, we now have  $\mathcal{I}[\![$  delete  $c\ x]\!] = \mathbf{new}\ c\ x$ , meaning  $\mathcal{T}'$  must be of the form:

$$\mathcal{T}' = \underbrace{\frac{C_2}{\Pi(x) = c'}}_{\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{new} \ c' \ x}$$

• Case 
$$\mathcal{T} = \underbrace{\frac{\mathcal{C}_1}{\text{arrayType}(a) \in \left\{\text{classIDs}, \mathbf{int}\right\}}}_{\left\langle\Pi, c\right\rangle \vdash_{stmt}^{\Gamma} \mathbf{new} \ a[e] \ x} \underbrace{\frac{\mathcal{C}_2}{\Pi(x) = a[\ ]}}_{\mathcal{T}-\text{Arrnew}}$$

In this case we still have  $\mathcal{I}[\![\mathbf{new}\ c\ x]\!] = \mathbf{delete}\ c\ x$ . Using  $\mathcal{C}_1$  and  $\mathcal{E}$ ,  $\mathcal{T}'$  must be of the form:

$$\mathcal{T}' = \underbrace{\text{arrayType}(a) \in \left\{ \text{classIDs}, \mathbf{int} \right\}}_{\mathcal{C}_1} \underbrace{\frac{\mathcal{E}}{\prod \vdash_{expr} e = \mathbf{int}}}_{\mathcal{C}_3} \underbrace{\frac{\mathcal{C}_3}{\prod(x) = a[\ ]}}_{\mathcal{C}_3}$$

• Case 
$$\mathcal{T} = \underbrace{\frac{\mathcal{C}_1}{\text{arrayType}(a) \in \left\{\text{classIDs}, \mathbf{int}\right\}}}_{\left\langle\Pi, c\right\rangle \vdash_{stmt}^{\Gamma} \mathbf{delete} \ a[e] \ x} \underbrace{\frac{\mathcal{C}_2}{\Pi(x) = a[\ ]}}_{\mathcal{T}\text{-ArrDlt}}$$

Similar to the object deletion case, we still have  $\mathcal{I}[\![$  delete  $c \ x]\!] = \mathbf{new} \ c \ x$ . Using  $\mathcal{C}_1$  and  $\mathcal{E}$ ,  $\mathcal{T}'$  must be of the form:

$$\mathcal{T}' = \underbrace{\frac{\mathcal{C}_1}{\text{arrayType}(a) \in \left\{\text{classIDs}, \mathbf{int}\right\}}_{\left\langle \Pi, c \right\rangle \vdash_{stmt}^{\Gamma} \mathbf{new} \ a[e] \ x}^{\mathcal{E}} \underbrace{\frac{\mathcal{C}_3}{\Pi(x) = a[\ ]}}_{\left\langle \Pi, c \right\rangle \vdash_{stmt}^{\Gamma} \mathbf{new} \ a[e] \ x}$$

• Case 
$$\mathcal{T} = \underbrace{\frac{C_1}{\Pi(x) = c'}}_{\text{C1}} \underbrace{\frac{C_2}{\Pi(x') = c'}}_{\text{C2}} \text{T-CP}$$

We have  $\mathcal{I}[\![\mathbf{copy}\ c\ x\ x']\!] = \mathbf{uncopy}\ c\ x\ x'$ . Using  $\mathcal{C}_1$ ,  $\mathcal{T}'$  must as such be of the form

$$\mathcal{T}' = \underbrace{\frac{\mathcal{C}_1}{\Pi(x) = c'}}_{\begin{array}{c} \overline{\Pi(x') = c'} \\ \end{array}} \underbrace{\frac{\mathcal{C}_3}{\Pi(x') = c'}}_{\begin{array}{c} \overline{\Pi(x') = c'} \\ \end{array}}$$

• Case 
$$\mathcal{T} = \underbrace{\frac{C_1}{\Pi(x) = c'}}_{\substack{C_1 \\ \overline{\langle \Pi, c \rangle} \vdash_{stmt}^{\Gamma} \mathbf{uncopy} \ c' \ x \ x'}}^{C_2}_{\substack{T-UCP}}$$
 T-UCP

We have  $\mathcal{I}[\mathbf{uncopy}\ c\ x\ x'] = \mathbf{copy}\ c\ x\ x'$ . Using  $\mathcal{C}_1$ ,  $\mathcal{T}'$  must as such be of the form

$$\mathcal{T}' = \underbrace{\frac{C_1}{\prod(x) = c'}}_{\substack{\Gamma \\ \langle \Pi, c \rangle \vdash_{stmt}^{\Gamma}}} \underbrace{\frac{C_3}{\prod(x') = c'}}_{\substack{\Gamma \\ copy \ c' \ x \ x'}}$$

• Case 
$$\mathcal{T} = \underbrace{\frac{\mathcal{E}_1}{\langle \Pi, c \rangle \vdash_{expr}^{\Gamma} e_1}}_{\mathcal{E}_{1}} \underbrace{\frac{\mathcal{E}_2}{\langle \Pi[x \mapsto c'], c \rangle \vdash_{stmt}^{\Gamma} s}}_{\mathcal{T}_{1}} \underbrace{\frac{\mathcal{E}_2}{\langle \Pi, c \rangle \vdash_{expr}^{\Gamma} e_2}}_{\mathcal{T}_{2}} T-LOCALBLOCK}$$

We have  $\mathcal{I}[[local\ t\ x=e]\ s$  delocal  $t\ x=e]$  = local  $t\ x=e$   $\mathcal{I}[[s]]$  delocal  $t\ x=e$ . By the induction hypothesis on  $\mathcal{S}$ , we obtain  $\mathcal{S}'$  of  $\langle \Pi[x\mapsto c'],c\rangle \vdash_{stmt}^{\Gamma} \mathcal{I}[[s]]$ . Using  $\mathcal{E}_1,\ \mathcal{S}'$  and  $\mathcal{E}_2$  we construct  $\mathcal{T}'$ 

$$\mathcal{T}' = \overbrace{\langle \Pi, c \rangle \vdash_{expr}^{\Gamma} e_1}^{\mathcal{E}_1} \overbrace{\langle \Pi[x \mapsto c'], c \rangle \vdash_{stmt}^{\Gamma} \mathcal{I}\llbracket s \rrbracket}^{\mathcal{E}'} \overbrace{\langle \Pi, c \rangle \vdash_{expr}^{\Gamma} e_2}^{\mathcal{E}_2}$$

$$\langle \Pi, c \rangle \vdash_{stmt}^{\Gamma} \mathbf{local} \ c' \ x = e_1 \quad \mathcal{I}\llbracket s \rrbracket \quad \mathbf{delocal} \ c' \ x = e_2$$

Using these added cases to the original proof provided in [11], Theorem 2.2 shows that well-typedness is preserved over inversion of ROOPL++ methods. As methods are well-typed if their body statement is well-typed, inversion of classes and programs also preserve well-typedness, as classes consists of methods and programs of classes, by using the class inverter presented in figure 2.28.

## 2.11 Computational Strength

Traditional, non-reversible programming languages have their computational strength measured in terms of their abilities to simulate the Turing machine (TM). If any arbitrary Turing machine can be implemented in some programming language, the language is said to be computationally universal or Turing-complete. In essence, Turing-completeness marks when a language can compute all computable functions. Reversible programming languages, like Janus, Roopl and Roopl++, are not Turing-complete as they only are capable of computing injective, computable functions.

For determining computing strength of reversible programming languages, Yokoyama et al. suggests that the reversible Turing machine (RTM) could serve as the baseline criterion [29]. As such, a reversible programming language is reversibly universal or r-Turing complete if it is able to simulate a reversible Turing machine cleanly, i.e. without generating garbage data. If garbage was on the tape, the function simulated by the machine would not be an injective function and as such, no garbage should be left after termination of the simulation.

### 2.11.1 Reversible Turing Machines

Before we show that ROOPL++ in fact is r-Turing complete, we present the formalized reversible Turing machine definition, as defined in [29].

**Definition 2.1.** (Quadruple Turing Machine)

A TM T is a tuple  $(Q, \Gamma, b, \delta, q_s, q_f)$  where

Q is the finite non-empty set of states

 $\Gamma$  is the finite non-empty set of tape alphabet symbols

 $b \in \Gamma$  is the blank symbol

 $\delta: (Q \times \Gamma \times \Gamma \times Q) \cup (Q \times \{/\} \times \{L, R\} \times Q)$  is the partial function representing the transitions

 $q_s \in Q$  is the starting state

 $q_f \in Q$  is the final state

The symbols L and R represent the tape head shift-directions left and right. A quadruple is either a symbol rule of the form  $(q_1, s_1, s_2, q_2)$  or a shift rule of the form  $(q_1, /, d, q_2)$  where  $q_1 \in Q$ ,  $q_2 \in Q$ ,  $s_1 \in \Gamma$ ,  $s_2 \in \Gamma$  and d being either L or R.

A symbol rule  $(q_1, s_1, s_2, q_2)$  means that in state  $q_1$ , when reading  $s_1$  from the tape, write  $s_2$  to the tape and change to state  $q_2$ . A shift rule  $(q_1, /, d, q_2)$  means that in state  $q_1$ , move the tape head in direction d and change to state  $q_2$ .

**Definition 2.2.** (Reversible Turing Machine)

A TM T is a reversible TM iff, for any distinct pair of quadruples  $(q_1, s_1, s_2, q_2) \in \delta_T$  and  $(q_1', s_1', s_2', q_2') \in \delta_T$ , we have

$$q_1 = q_1' \implies (t_1 \neq / \land t_1' \neq / \land t_1 \neq t_1')$$
 (forward determinism)  
 $q_2 = q_2' \implies (t_1 \neq / \land t_1' \neq / \land t_2 \neq t_2')$  (backward determinism)

A RTM simulation implemented in ROOPL by representing the set of states  $\{q_1, \ldots, q_n\}$  and the tape alphabet  $\Gamma$  as integers and the rule / and direction symbols L and R as the uppercase integer literals **SLASH**, **LEFT** and **RIGHT** was presented in [11]. As ROOPL contains no array or stack primitives, the transition table  $\delta$  was suggested to be represented as a linked list of objects containing four integers  $\mathbf{q1}$ ,  $\mathbf{s1}$ ,  $\mathbf{s2}$  and  $\mathbf{q2}$  each, where  $\mathbf{s1}$  equals **SLASH** for shift rules. In ROOPL++, we do, however, have an array primitive and as such, we can simply simulate transitions by having rules  $\mathbf{q1}$ ,  $\mathbf{s1}$ ,  $\mathbf{s2}$  and  $\mathbf{q2}$  represented as arrays, where the number of cells in each array is **PC\_MAX**, in a similar fashion as shown in [29].

#### 2.11.2 Tape Representation

As with regular Turing machines, the Reversible Turing machines also have tapes of infinite length. Therefore, we must simulate tape growth in either direction. Yokoyama et al. represented the tape using two stack primitives in the Janus RTM interpreter and Haulund used list of objects. In ROOPL++, we could implement a stack, as objects are not statically scoped as in ROOPL. However, in terms of easy of use, a doubly linked list implementation similar to the one presented in section 2.7.3, of simple cell objects containing value, left, right and self fields, is more intuitive.

As such, the tape head finds a tape cell by inspecting a specific element of the doubly linked list tape representation. When we move in either direction, we simply set the neighbour element as the new tape head and allocate a new neighbour for the new tape head cell, if we are at the end of the list, to simulate the infinitely-length tape. Reversibly, this means that when we move in the opposite direction, blank cells are deallocated if we are moving the tape head away from the cell currently neighbouring either end of the tape.

Figure 2.29 shows the *moveRight* method for moving the tape head right. If the current tape head has no instantiated right neighbour we construct one using the **new** statement. Uncalling this method will move the tape head left. If the tape head is empty after moving left, we simply allocate a new cell, thus allowing tape growth in both directions.

```
1 method moveRight(int symbol, Cell tapeHead)
      local Cell right = nil
2
3
      local Cell tmp = nil
4
      uncall tapeHead::getSymbol(symbol) // Put symbol back in current cell
                                            // Get right neighbour
5
      call tapeHead::getRight(right)
6
7
      if right = nil && symbol = BLANK then
          symbol ^= BLANK
                                           // Zero clear symbol
8
          new Cell right
                                            // Init new neighbour
9
                                            // Copy reference to self
10
          copy Cell right tmp
          uncall right::getSelf(tmp)
11
                                            // Store self reference
          uncall right::getLeft(tapeHead) // Set tape head as left of new cell
          right <=> tapeHead
13
14
      else
15
          call right::getLeft(tmp)
                                            // Get copy of tape head reference
          uncopy Cell tmp tapeHead
                                            // Clear reference to tape head
16
17
          if tapeHead = nil && symbol = BLANK
18
19
              call tmp::getSelf(tapeHead) // rev: set self pointer
                                           // rev: new self pointer
20
              uncopy Cell tmp tapeHead
              delete Cell tmp
                                           // rev: new left neighbour
21
22
              symbol ^= BLANK
23
          else skip
                                           // In reverse:
                                           // Allocate new left if current is nil
          fi tmp = nil
24
25
26
          uncall right::getLeft(tmp)
                                            // Put tape head reference back
27
          tapeHead <=> right
          call tapeHead::getRight(right) // Get right of new tape head
          call tapeHead::getSymbol(symbol) // Get symbol of new tape head
29
30
      fi right = nil
31
      uncall tapeHead::getRight(right)
                                           // Set right neighbour
      delocal Cell right = nil
32
      delocal Cell tmp = nil
33
```

Figure 2.29: Method for moving the tape head in the RTM simulation

#### 2.11.3 Reversible Turing Machine Simulation

Figure 2.30 shows the modified method *inst* from [29], which executes a single instruction given the tape head, the current state, symbol, program counter and the four arrays representing the transition rules. As described above, we **call** *moveRight* to move the tape head right and **uncall** to move the tape head left.

Figure 2.31 shows the simulate method which is the main method responsible for running the RTM simulation. The tape is extended in either direction when needed, and the program counter is incremented.

Unlike the ROOPL simulation, ROOPL++ is not limited by stack allocated, statically-scoped objects. Due to this limitation, the ROOPL RTM simulator cannot finish with the TM tape as its program output when the RTM halts, as the call stack of the simulation must unwind before termination. As objects in ROOPL++ are not bound by this limitation, the TM tape will exist as the program output when the RTM halts.<sup>1</sup>

Instantiating a RTM simulation consists of initializing an initial tape head cell, as well as the

<sup>&</sup>lt;sup>1</sup>We are here breaking the rule that a **new** statement must eventually be followed by a **delete** statement to free the data.

```
1 method inst(int state, int symbol, int[] q1, int[] s1,
              int[] s2, int[] q2, int pc, Cell tapeHead)
      if state = q1[pc] && symbol = s1[pc] then
3
                                                 // Symbol rule:
4
          state += q2[pc]-q1[pc]
                                                   // set state to q2[pc]
          symbol += s2[pc]-s1[pc]
                                                   // set symbol to s2[pc]
5
      fi state = q2[pc] && symbol = s2[pc]
      if state = q1[pc] && s1[pc] = SLASH then
7
                                                   // Move rule:
          state += q2[pc]-q1[pc]
                                                   // set state to q2[pc]
8
          if s2[pc] = RIGHT then
              call moveRight(symbol, tapeHead)
10
                                                   // Move tape head right
          fi s2[pc] = RIGHT
11
          if s2[pc] = LEFT then
13
              uncall moveRight(symbol, tapeHead)
                                                  // Move tape head left
14
          fi s2[pc] = LEFT
      fi state = q2[pc] && s1[pc] = SLASH
15
```

Figure 2.30: Method for executing a single TM transition

```
1 method simulate(Cell tapeHead, int state, int[] q1, int[] s1, int[] s2, int[] q2, int pc)
      from state = Qs do
2
          pc += 1
3
                                                 // Increment pc local int symbol = 0
          call tapeHead::getSymbol(symbol)
                                                 // Fetch current symbol
          call inst(state, symbol, q1, s1, s2, q2, pc, tapeHead)
5
6
          uncall tapeHead::getSymbol(symbol)
                                                // Zero-clear symbol delocal symbol = 0
          if pc = PC_MAX then
                                                 // Reset pc
              pc ^= PC_MAX
          else skip
          fi pc = 0
10
11
      loop skip
      until state = Qf
12
```

Figure 2.31: Main RTM simulation method

transition rule arrays. After initialization, the *simulate* method is simply called and the simulation begins.

# Dynamic Memory Management

In order to allow objects to live outside of static scopes, we need to utilize a different memory management technique, such that objects are not allocated on the stack. Dynamic memory management presents a method of storing objects in different memory structures, most commonly, a memory heap. Most irreversible, modern programming languages uses dynamic memory management in some form for allocating space for objects in memory.

However, reversible, native support for complex data structures is a non-trivial matter to implement. Variable-sized records and frames need to be stored efficiently in a structured heap, while avoiding garbage build-up to maintain reversibility. A reversible heap manager layout has been proposed for a simplified version of the reversible functional language RFun and later expanded to allow references to avoid deep copying values [2, 30, 18].

This chapter presents a brief introduction to fragmentation, garbage and linearity and how these respectively are handled reversibly, and a discussion of various heap manager layouts considered for ROOPL++, along with their advantages and disadvantages in terms of implementation difficulty, garbage build-up and the OOP paradigm.

# 3.1 Fragmentation

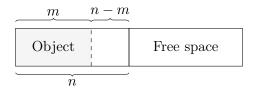
Efficient memory usage is an important matter to consider when designing a heap layout for a dynamic memory manager. In a stack allocating memory layout, the stack discipline is in effect, meaning only the most recently allocated data can be freed. This is not the case with heap allocation, where data can be freed regardless of allocation order. A potential side effect of this freedom, comes as a consequence of memory fragmentation. We distinguish different types of fragmentation as internal or external fragmentation.

Internal fragmentation refers to unused space inside a memory block used to store an object, if, say, the object is smaller than the block it has been allocated to. External fragmentation occurs as blocks freed throughout execution are spread across the memory heap, resulting in *fragmented* free space [20].

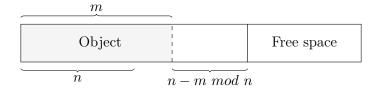
### 3.1.1 Internal Fragmentation

Internal fragmentation occurs in the memory heap when part of an allocated memory block is unused. This type of fragmentation can arise from a number of different scenarios, but mostly it originates from cases of *over-allocation*, which occurs when the memory manager delegates memory larger than required to fit an object, due to e.g. fixed-block sizing.

For an example, consider a scenario, in which we allocate memory for an object of size m onto a simple, fixed-sized block heap. The fixed block size is n and  $m \neq n$ . If n > m, internal fragmentation would occur of size n - m for every object of size m allocated in said heap. If n < m, numerous blocks would be required for allocation to fit our object. In this case the internal fragmentation would be of size  $n - m \mod n$  per allocated object of size m.



**Figure 3.1a:** Creation of internal fragmentation of size n-m due to over-allocation



**Figure 3.1b:** Creation of internal fragmentation of size  $n-m \mod n$  due to over-allocation

Figure 3.1a and 3.1b visualize the examples of internal fragmentation build-up from *over-allocating* memory.

It is difficult for the memory manager to reclaim wasted memory caused by internal fragmentation, as it usually originates from a design choice. Intuitively, internal fragmentation can best be prevented by ensuring that the size of block(s) being used for allocating space for an object of size m either match or sums to this exact size, when designing the layout.

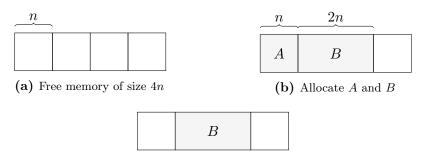
### 3.1.2 External Fragmentation

External fragmentation materializes in the memory heap when a freed block becomes partly or completely unusable for future allocation if, say, it is surrounded by allocated blocks but the size of the freed block is too small to contain objects on its own.

This type of fragmentation is generally a more substantial cause of problems than internal fragmentation, as the amount of wasted memory typically is larger and less predictable in external fragmentation blocks than in internal fragmentation blocks. Depending on the heap implementation, i.e. a layout using variable-sized blocks of, say, size  $2^n$ , the internal fragment size becomes considerable for large values of n.

Non-allocatable external fragments become a problem when it is impossible to allocate space for a large object as a result of too many non-consecutive blocks scattered around the heap, caused by the external fragmentation. Physically, there is enough space to store the object, but not in the current heap state. In this scenario we would need to relocate blocks in such a manner that the fragmentation disperses, which is not possible to do reversibly.

Allocation and deallocation order is important in order to combat external fragmentation. For example, if we have a class A, which fit on one memory block of size n, and we have a class B, which fit on two memory blocks of size n and limited memory space, we can easily reach a situation, where we cannot fit more B objects due to external fragmentation.



(c) Free A. Cannot fit another B due to external fragmentation

Figure 3.2: Example of external fragmentation caused for allocation and deallocation order

Figure 3.2 shows this example, where the allocation and deallocation order causes a situation, in which we cannot allocate any more B objects, even though we physically have the required amount of free space in memory.

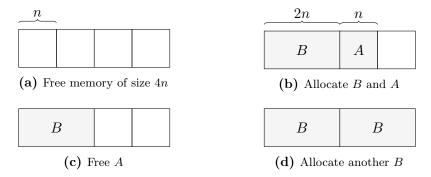


Figure 3.3: Example of avoiding external fragmentation using allocation and deallocation order

Figure 3.3 shows how changing allocation and deallocation order can combat external fragmentation.

# 3.2 Memory Garbage

A reversible computation should be garbage-free and as such it should be our goal to return the memory to its original state after program termination.

Traditionally, in non-reversible programming languages, freed memory blocks are simply re-added to the free list during deallocation and no modification of the actual data stored in the block is performed, as it simply is overwritten when the block is used later on. In the reversible setting we must return the memory block to its original state after the block has been freed (e.g. zero-cleared), to uphold the time-invertible and two-directional computational model. Figure 3.4 illustrates how the output data (or garbage) of an injective function f is the input to its inverse function  $f^{-1}$ .

In heap allocation layouts, we maintain one or more free lists to keep track of free blocks during program execution, which are stored in memory, besides the heap representation itself. These free lists can essentially be considered garbage and as such, they must also be returned to their original state after execution. Furthermore, the heap itself can also be considered garbage and if it grows during execution, it should also be returned to its original size.

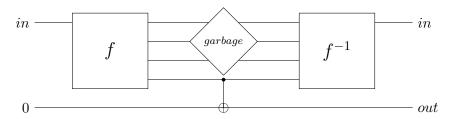


Figure 3.4: The "garbage" output of an injective function f is the input to its inverse function  $f^{-1}$ 

Returning the free list(s) to their original states is a non-trivial matter, which is highly dependent on the heap layout and free list design. Axelsen and Glück introduced a dynamic memory manager which allowed heap allocation and deallocation, but without restoring the free list to its original state in [2]. Axelsen and Glück argue that an unrestored free list can be considered harmless garbage in the sense that the free list residing in memory after termination is equivalent to a restored free list, as it contains the same blocks, but linked in a different order, depending on the order of allocation and deallocation operations performed during program execution. Figure 3.5 illustrates how an inverse, injective function  $f^{-1}$ , whose non-inverse function f computes something which modifies a given free lists, does not require the *exact* output free list of f, but any free list of same layout as input for the inverse function  $f^{-1}$ . The output free list of  $f^{-1}$  will naturally be a further modified free list.

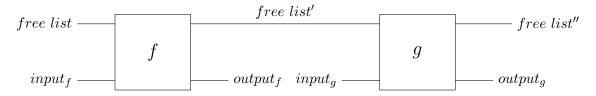


Figure 3.5: All free lists are considered equivalent "garbage" in terms of injective functions

This intuitively leads to the question of garbage classification. In the reversible setting all functions are injective. Thus, given some  $input_f$ , in a reversible computation using heap allocation, the injective function f produces some  $output_f$  and some modified free list  $free\ list'$ , obtained after storing or freeing data in the heap during the execution of f with an input  $free\ list$ . A future injective function in the program, function g, must thus take any modification of the original

free list in addition to its input to produce its output  $output_g$  and a potentially further modified free list,  $free\ list''$ . However, in the context of reversible heaps, we must consider all free lists as of equivalent, harmless garbage class and thus freely substitutable with each other, as injective functions still can drastically change the block layout, free list order, etc. during its execution in either direction. Figure 3.5 shows how any free list can be passed between a function f and a function g further in the program.

# 3.3 Linearity and Reference Counting

Programming languages use different approaches for storing and synchronizing variables and objects in memory. Typing *linearity* is a distinction, which can reduce storage management and synchronization costs [4].

Reversible programming languages such as JANUS and ROOPL are linear in the sense that object and variable pointers cannot be copied and are only deleted during deallocation. Pointer copying greatly increases the flexibility of programming, especially in a reversible setting where zero-clearing is critical, at the cost of increased management in form of reference counting for e.g. objects. For variables, pointer copying is not particular interesting, nor would it add much flexibility as the values of a variable simply can be copied into statically-scoped local blocks. For objects however, tedious amounts of boilerplate work must be done if object A and B need to work on the same object C and only one reference to each object is allowed. Copying is not an option as field modification in one copy does not affect the other copies.

Mogensen presented the reversible functional language RCFUN which use reference counting to allow multiple pointers to the same memory nodes as well as a translation from RCFUN into JANUS in [18]. In RCFUN, reference counting is used to manage and trace the number of pointer copies made by respectively incrementing and decrementing a reference count stored in the memory node, whenever the original node pointer is copied or a copy pointer is deleted. For the presented heap manage, deletion of object nodes was only allowed when no references to a node remained.

In non-reversible languages, reference counting is also used in garbage collection by automatically deallocating unreachable objects and variables which contains no referencing.

# 3.4 Heap Manager Layouts

Heap managers can be implemented in numerous ways. Different layouts yield different advantages when allocating memory, finding a free block or when collecting garbage. As our goal is to construct a garbage-free heap manager, our finalized design should emphasize and reflect this objective in particular. Furthermore, we should attempt to allocate and deallocate memory as efficiently as possible, as merging and splitting of blocks is a non-trivial problem in a reversible setting and to avoid problematic fragmentation.

For the sake of simplicity, we will not consider the issue of retrieving memory pages reversibly. A reversible operating system is a long-term dream of the reversible researcher and as reversible programming language designers, we assume that ROOPL++ will be running in an environment,

in which an operating system will be supplying memory pages and their mappings. As such, the following heap memory designs reflect this preliminary assumption, that we can always query the operating system for more memory.

Historically, most object-oriented programming languages utilize a dynamic memory manager during program execution. In lower-level languages such as C or C++, memory management is manual and allocation has to be stated explicitly and with the requested size through the **malloc** statement and deallocated using the **free** statement. Modern languages, such as JAVA and PYTHON, *automagically* allocates and frees space for objects and variable-sized arrays by utilizing their dynamic memory manager and garbage collector to dispatch **malloc**- and **free**-like operations to the operating system and managing the obtained memory blocks in private heap(s) [15, 25, 21]. The heap layout of these managers vary from language to language and compiler to compiler.

Previous work on reversible heap manipulation has been done for reversible functional languages in [2, 10, 19].

Axelsen and Glück presented a static heap structure consisting of LISP-inspired constructor cells of fixed size and a single free list for the reversible function language RFUN in [2]. Mogensen presented an implementation in JANUS of reversible reference counting under the assumption of Axelsen and Glück's heap manager in [18]. Building on the previous work, Mogensen later presented a reversible intermediate language RIL and an implementation in RIL of a reversible heap manager, which uses reference counting and hash-consing to achieve garbage collection in [19].

We do not consider reference counting or garbage collection in the layouts presented in the following sections, but we later show how the selected layout for ROOPL++ is extended with reference counting in section 4.7.

#### 3.4.1 Memory Pools

The simplest heap layout we can design uses fixed-sized blocks. This design is also known as memory pools, as memory is allocated from "pools" of fixed-sized blocks regardless of the record size. To model these pools of fixed-sized blocks, we simply use a linked list of identically sized free block cells, which we maintain over execution. While the fixed-block layout is simple and relatively easy in terms of implementation it is also largely uninteresting as it provides little to no options, besides sizing of the fixed-blocks, to combat fragmentation.

This layout comes with a few options in terms of the actual heap layout. If we only allow allocation of consecutive, adjacent free blocks, we should keep the free list sorted. If the free list is not sorted, and we have to allocate an object which requires n blocks, we have to iterate the free list  $n^2$  times in the worst case to find a chain of consecutive blocks large enough to fit the object. The sorting part itself is non-trivial matter. Furthermore, we need some overhead storage inside the object to contains the references of the blocks occupied by the object, or some other structure which can be used when deallocating the object and returning all the blocks to the free list. If we allow allocation of non-consecutive blocks, larger amounts of bookkeeping is required as we need to store knowledge of when and where the object is split.

Figures 3.2 and 3.3 from earlier in this chapter, in section 3.1.2 on page 47 illustrates examples with consecutive, fixed-sized block allocation.

### 3.4.2 One Heap Per Record Size

Instead of allocating space for objects from a single free list and heap, we could design an approach which uses one heap per record size, known as a multi-heap layout. The respective classes and their sizes are easily identified during compile time from which the amount of heaps and free list will be initialized. This means the layout is very dynamic and potentially can change drastically in terms of the amount of heaps utilized depending on the input program.

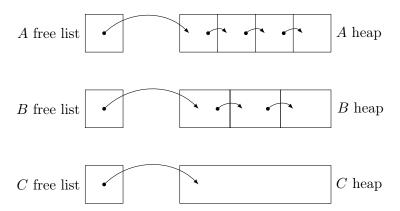


Figure 3.6: Memory layout using one heap per record size

Figure 3.6 illustrates three heaps with respective free lists for three classes A, B and C of size n, 2n and 4n. Each heap is represented as a simple linked list with the free list simply being a pointer to the first free block in the heap.

The advantage of this approach would be effective elimination of internal and external fragmentation, as each heap fits their targeted record perfectly, making each allocation and deallocation tailored to the size of the record obtained from a static analysis during compilation, resulting in no over-allocation and no unusable chunks of freed memory appearing during varying deallocation order. Implementation-wise, allocation of an object of a given class simply becomes the task of popping the head of the respective free list, which can easily be determined at compile time. The deallocation is simply adding a new head to the free list.

Listing 3.1 outlines the allocation algorithm for this layout written in extended Janus from [29]. We assume that the heads of the free lists are stored in a single array primitive, such that the free list for records of size n are indexed at n-2 and n>2 (as every record needs some overhead) and that we have heaps for continuous size range with no gaps.

The algorithm consists of an entry point named **malloc** and a recursion body named **malloc1**. Given a zero-cleared pointer p, the size of the object we are allocating  $o_{size}$  and the array of free lists primitive, the recursion body is called after initializing a counter, which is an index into the free lists array and a counter size,  $c_{size}$ , which is the block size of the current free list the counter is indexed in. The recursion body first updates the free list index until we find a free list with a size greater or equal to the size of the object we are allocating. Once such a free list

has been found, the head of the free list is simply popped and the next block is set as the new head.

```
procedure malloc(int p, int osize, int freelists[])
       local int counter = 0
      local int csize = 2
3
       call malloc1(p, osize, freelists, counter, csize)
4
5
       delocal int csize = 2
      delocal int counter = 0
6
    procedure malloc1(int p, int osize, int freelists[], int counter, int csize)
8
9
      if (csize < osize) then</pre>
        counter += 1
10
         csize += 1
11
12
        call malloc1(p, osize, freelists, counter, csize)
13
        csize -= 1
        counter -= 1
14
15
       else
        p += freelists[counter]
16
17
         freelists[counter] -= p
18
         // Swap head of free list with next block of p
19
20
         freelists[counter] ^= M(p)
21
        M(p) ^= freelists[counter]
         freelists[counter] ^= M(p)
22
       fi csize < osize
23
```

Listing 3.1: Allocation algorithm for one heap per record size implemented in extended Janus

The obvious disadvantage to this layout is the amount of bookkeeping and workload associated with growing and shrinking a heap and its neighbours, in case the program requests additional memory from the operating system. In real world object-oriented programming, most classes feature a small number of fields, very rarely more than 16.

Additionally, helper classes of other sizes would spawn additional heaps and bookkeeping work, making the encapsulation concept of OOP rather unattractive, for the optimization-oriented reversible programmer.

Finally, while internal and external fragmentation is effectively eliminated, we are left with additional and considerable amounts of garbage in forms of all the heaps and free lists initialized in memory. If two record types only differ one word in size, two heaps would be initialized. Each heap intuitively need to be initialized with a chunk of memory from the underlying operating system such that objects can be allocated on their respective heaps, regardless of the number of times the heap is used during program execution. This is an obvious space requirement increase over the previously presented layout, and on average, the amount of required memory for a program compiled using this approach would probably be larger, than some of the following layouts, due to unoptimized heap utilization and sharing. Heap of some sizes may be mostly empty when another is full, resulting in wasted memory.

#### 3.4.3 One Heap Per Power-Of-Two

To address the issues of the previous heap manager layout, we can optimize the amounts of heaps required by introducing a relatively small amount of internal fragmentation. Instead of having a heap per record size, we could have a heap per power-of-two. Records would be stored in the heap closest to their respective size and, as such, we reduce the number of heaps needed, as

many different records can be stored in the same heap. Records of size 5, 6, 7 and 8 would in the former layout be stored in four different heaps, where they would be stored in a single heap using this layout. Figure 3.7 illustrates the free lists and heaps up to  $2^n$ .

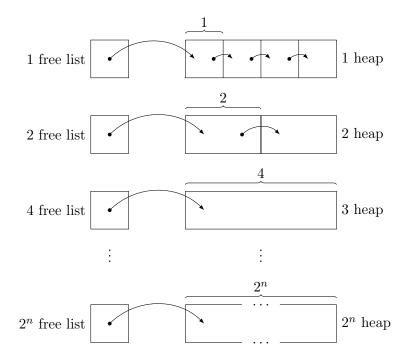


Figure 3.7: Memory layout using one heap per power-of-two

Internal fragmentation does become a problem for very large records, as blocks are only of size  $2^n$ . An object of size 65 would fit in a 128 sized block, resulting in considerable amounts of wasted memory space in form of internal fragmentation. However, in the real world, most records are small and allocation of records causing this much amount of fragmentation is an unlikely scenario. To avoid large amounts of internal fragmentation building up when allocating large records, we could allocate space for large objects using smaller blocks. If a record exceeds some limit, which has been determined the cutoff point, one kilobyte for an example, we could split it into  $\sqrt{n}$  sized chunks and use blocks of that size instead. This would reduce the amount of internal fragmentation at the cost of increased bookkeeping. For smaller records, very minimal amounts of internal fragmentation occur.

The number of heaps needed for a computation can be determined at compile time by finding the smallest and largest record sizes and ensuring we have heaps to fit these effectively. The allocation process consists of determining the closest  $2^n$  to the size of the record we are allocating and then simply popping the head of the respective free list.

Listing 3.2 shows a modified **malloc1** recursion body for the power-of-two approach. Once again, we assume our array of free lists contains the head of each free list, such that index n is the head of the free list of size  $2^{n+1}$ . Instead of incrementing the counter size by one, as in the former layout algorithm, we double it, using the shown **double** procedure. Besides this change, the algorithm remains unchanged and still assumes each heap has been initialized along with the free lists.

```
procedure double(int target)
      local int current = target
2
3
      target += current
4
      delocal int current = target / 2
5
    procedure malloc1(int p, int osize, int freelists[], int counter, int csize)
6
7
      if (csize < osize) then</pre>
           counter += 1
8
           call double(csize)
9
           call malloc1(p, osize, freelists, counter, csize)
10
11
           uncall double(csize)
12
           counter -= 1
13
       else
         if freelists[counter] != 0 then
14
15
           p += freelists[counter]
           freelists[counter] -= p
16
17
           // Swap head of free list with next block of p
18
19
           freelists[counter] ^= M(p)
20
           M(p) ^= freelists[counter]
           freelists[counter] ^= M(p)
21
22
         else
23
           counter += 1
           call double(csize)
24
           call malloc1(p, osize, freelists, counter, csize)
25
26
           uncall double(csize)
27
           counter -= 1
         fi freelists[counter] = 0 || p != freelists[counter]
28
      fi csize < osize
29
```

Listing 3.2: Allocation algorithm for one heap per power-of-two implemented in extended Janus

### 3.4.4 Shared Heap, Record Size-Specific Free Lists

A natural proposal, considering the disadvantages of the previously presented designs, would be using a shared heap instead of record-specific heaps. This way, we ensure minimal fragmentation when allocating and freeing as the different free lists ensure that allocation of an object wastes as little memory as possible. By only keeping one heap, we eliminate the growth/shrinking issues of the multiple heap layout.

There is, however, still a considerable amount of bookkeeping involved in maintaining multiple free lists. Having mixed-size blocks in a single heap is also a task which might prove difficult to accomplish reversibly. How initialization and destruction of said heap should work is not clear. As with the multiple heap version of this layout, we are still left with the issues surrounding two records which only differs one word in size. In the former layout, two heaps were required to store records of these types. In this layout, we need to store two block sizes in our heap to allocate these records, with no internal fragmentation. We could allow these objects to be allocated on similarly-sized blocks, if we round the calculated class sizes up to, say, a power-of-two. We would essentially have a shared heap, power-of-two-specific free lists layout.

As the only change in this design are the heaps themselves, the allocation process remains unchanged from the one presented in listing 3.1 or listing 3.2 if we use the power-of-two approach. Figure 3.8 visualizes the shared heap and the free lists of this layout.

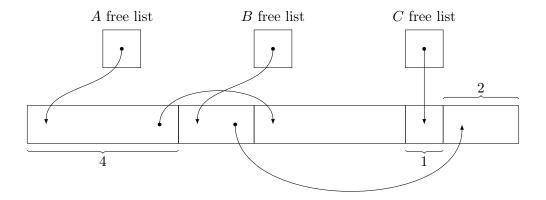


Figure 3.8: Record size-specific free lists on a shared heap (powers of two)

### 3.4.5 Buddy Memory

The Buddy Memory layout utilizes blocks of variable-sizes of the power-of-two, typically with one free list per power-of-two using a shared heap. When allocating an object of size m, we simply check the free lists for a free block of size n, where  $n \ge m$ . Is such a block found and if n > m, we split the block into two halves recursively, until we obtain the smallest block capable of storing m. When deallocating a block of size m, we do the action described above in reverse, thus merging the blocks again, where possible [13].

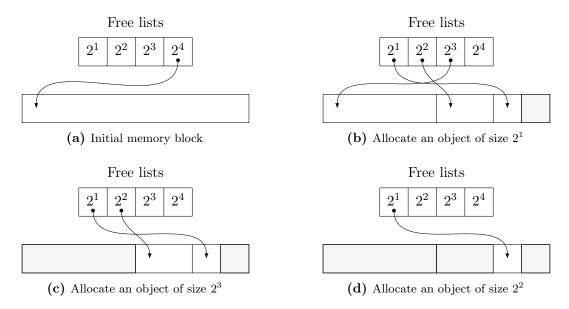


Figure 3.9: Buddy Memory block allocation example

Figure 3.9 illustrates an example of block splitting during allocation in the buddy system. Originally, one block of free memory is available. When allocating a record three factors smaller than the original block, three splits occurs.

This layout is somewhat of a middle ground between the previous three designs, addressing a number of problems found in these. The Buddy Memory layout uses a single heap for all

record-types, thus eliminating the problems related to moving adjacent heaps reversibly in a multi-heap layout. To optimize the problems around initializing a usable amount of variable-sized blocks in a shared heap, we simply initialize one large block in the buddy system, which we will split into smaller parts during execution and merge where possible when freed.

The main drawback from this layout is the amount of internal fragmentation. As we only allocate blocks of a power-of-two size, substantial internal fragmentation follows when allocating large records, i.e. allocating a block of size 128 for a record of size 65. However, as most real world programs uses much smaller sized records, we do not consider this a very frequent scenario. As discussed in section 3.4.3, we would split large records into chunks of  $\sqrt{n}$  at the cost of additional bookkeeping.

Implementation-wise, this design would require doubling and halving of numbers related to the power-of-two. This action translates well into the reversible setting, as a simply bit-shifting directly gives us the desired result.

```
procedure malloc1(int p, int osize, int freelists[], int counter, int csize)
2
      if (csize < osize) then</pre>
3
           counter += 1
4
           call double (csize)
           call malloc1(p, osize, freelists, counter, csize)
6
           uncall double(csize)
7
           counter -= 1
8
9
           if freelists[counter] != 0 then
               p += freelists[counter]
10
11
               freelists[counter] -= p
12
13
               // Swap head of free list with next block of p
               freelists[counter] ^= M(p)
14
15
               M(p) ^= freelists[counter]
16
               freelists[counter] ^= M(p)
17
           else
               counter += 1
               call double(csize)
19
               call malloc1(p, osize, freelists, counter, csize)
20
               uncall double(csize)
               counter -= 1
22
23
               freelists[counter] += p
24
               p += csize
           fi freelists[counter] = 0 || p - csize != freelists[counter]
25
       fi csize < osize
26
```

Listing 3.3: The Buddy Memory algorithm implemented in extended Janus

Listing 3.3 shows the Buddy Memory algorithm implemented in the extended Janus variant with local blocks from [29]. For simplification, object sizes are rounded to the nearest power-of-two during compile-time. The algorithm extends on the one heap per power-of-two algorithm presented in listing 3.2, page 53. The body of the allocation function is still executed recursively until a free list for a  $2^n$  larger than the size of the object has been found. Once found, we continue searching until we have found a non-empty free list. If the non-empty free list for a  $2^n$  larger than the object is found, the head of the list is popped and the popped block is split recursively, until a block the desired size is obtained. Throughout the splitting process, empty free lists are updated when a larger free block is split into a block which fits into those lists.

Since a split block is always added as two blocks to an empty free list, we can only merge adjacent

blocks if they are the only two blocks in a free list.								

# Compilation

The following chapter presents the considerations and translation schemas used in the process of translating ROOPL++ to the reversible low-level machine language PISA. As ROOPL++ is an extension of ROOPL, many techniques are carried directly over, and have as such been left out.

Before presenting the ROOPL++ compiler, a brief overview of the memory layout and modeling of the ROOPL compiler, which the ROOPL++ compiler is a continuation of, is provided.

# 4.1 The ROOPL to PISA Compiler

Haulund presented a proof-of-concept compiler along with the design for ROOPL. The compiler translates well-typed ROOPL programs into the reversible machine language PISA in [11]. The ROOPL compiler (ROOPLC) is written in HASKELL and hosted at https://github.com/TueHaulund/ROOPLC.

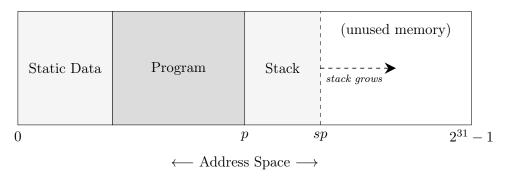


Figure 4.1: Memory layout of a ROOPL program, originally from [11]

Figure 4.1 shows the memory layout of a compiled ROOPL program. The layout consists of a static storage segment, the program segment and the stack.

The object model is simple and only features one additional word for storing the address of the virtual table for the object class. Figure 4.2 shows the prefixing for three simple classes modeling geometric shapes.

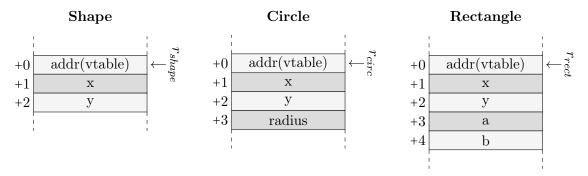


Figure 4.2: Illustration of prefixing in the memory layout of 3 ROOPL objects, originally from [11]

# 4.2 ROOPL++ Memory Layout

ROOPL++ builds upon the memory layout of its predecessor with dynamic memory management. The reversible Buddy Memory heap layout presented in section 3.4.5 is utilized in ROOPL++ as it is an interesting layout, addressing a number of disadvantages found in other considered layouts, naturally translates into a reversible setting with one simple restriction (i.e only blocks which are heads of their respectable free lists are allocatable) and since its only drawback is dismissible in most real world scenarios.

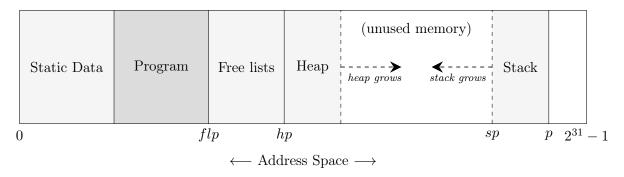


Figure 4.3: Memory layout of a ROOPL++ program

Figure 4.3 shows the full layout of a ROOPL++ program stored in memory.

- As with ROOPL, the static storage segment contains load-time labelled **DATA** pseudo-instructions, initialized with virtual function tables and other static data needed by the translated program.
- The program segment is stored right after the static storage and contains the translated ROOPL++ program instructions.
- The free lists maintained by the Buddy Memory heap layout is placed right after the program segment, with the *free list pointer flp* pointing at the first free list. The free lists are simply the address pointing to the first block of its respective size. The free lists are stored such that the free list at address flp + i corresponds to the free list of size  $2^{i+1}$ .
- The heap begins directly following the free lists. Its beginning is marked by the *heap* pointer (hp).

• Unlike in ROOPL, where the stack grows upwards, the ROOPL++ stack grows downwards and begins at address p. The stack remains a LIFO structure, analogously to ROOPL.

As mentioned in the previous chapter, we assume an underlying reversible operating system providing us with additional memory when needed. With no real way of simulating this, the ROOPL++ compiler places the stack at a fixed address p and sets one free block in the largest  $2^n$  free list initially. The number of free lists and the address p is configurable in the source code, but defaults to 10 free lists, meaning initially one block of size 1024 is available and the stack is placed at address 1024 words after the heap.

In traditional compilers, the heap pointer usually points to the end of the heap. For reasons stated above, we never grow the heap as we start with a heap of fixed size. As such, the heap pointer simply points to the beginning of the heap.

The heap can simply be expanded by adding another block of the largest possible size and storing the address of the respective free list.

In the following sections of this chapter, we will present various translation techniques. In these translations, we will make use of a number of PISA pseudo-instructions to subtract integer values from registers and pushing/popping to the program stack. The pseudo-instructions are shown in figure 4.4 and are modified from [11], as the direction of the program stack is flipped in ROOPL++.

Figure 4.4: Definition of pseudoinstructions SUBI, PUSH and POP, modified from [11]

#### 4.3 Inherited ROOPL features

As mentioned, a number of features from ROOPL carries over to ROOPL++.

The dynamic dispatching mechanism presented in [11] is inherited. As such, the invocation of a method implementation is based on the type of the object at run time. Virtual function tables are still the implementation strategy used in the dynamic dispatching implementation.

Evaluation of expressions and control flow remains unchanged.

For completeness, object blocks are included and still stack allocated as their life time is limited to the scope of their block and the dynamic allocation process is quite expensive in terms of register pressure and number of instructions compared to the stack allocated method implemented in the ROOPL compiler.

## 4.4 Program Structure

The program structure of a translated ROOPL++ is analogous to the program structure of a ROOPL program with the addition of free lists and heap initialization. The full structure is shown in figure 4.5.

```
; Static data declarations
(1)
(2)
                    . . . . . .
                                                    ; Code for program class methods
(3)
       start:
                   START
                                                    ; Program starting point
(4)
                                                   ; Initialize free lists pointer
                   ADDI
                               r_{flps}
(5)
                                                    ; Initialize heap pointer
                   XOR
                               r_{hp}
                                      r_{flps}
(6)
                   ADDI
                               r_{hp}
                                                    ; Initialize heap pointer
                   XOR
                                                    ; Store address of initial free memory block in r_b
(7)
                                       r_{hp}
                                                    ; Index to end of free lists
(8)
                   ADDI
                                      size_{fls}
                               r_{flps}
(9)
                   SUBI
                                                    ; Index to last element of free lists
                               r_{flps}
(10)
                   EXCH
                               rb
                                                    ; Store address of first block in last element of free lists
                                       r_{flps}
                                                    ; Index to end of free lists
(11)
                   ADDI
                                      1
                               r_{flps}
(12)
                   SUBI
                                                    ; Index to beginning of free lists
                               r_{flps}
(13)
                   XOR
                                                    ; Initialize stack pointer
                               r_{sp}
                                       offset_{stack}; Initialize stack pointer
(14)
                   ADDI
                               r_{sp}
(15)
                   SUBI
                                                    ; Allocate space for main object
                               r_{sp}
                                                    ; Store address of main object in r_m
(16)
                   XOR
                               r_m
                                       r_{sp}
                                       label_{vt}
                                                   ; Store address of vtable in r_v
(17)
                   XORI
                               r_v
                   PUSH
                                                   ; Push address of vtable onto stack
(18)
                               r_v
(19)
                   PUSH
                               r_m
                                                   ; Push 'this' onto stack
(20)
                               label_m
                                                   ; Call main procedure
                   BRA
                   POP
                                                   ; Pop 'this' from stack
(21)
                               r_m
(22)
                   POP
                               r_v
                                                   ; Pop vtable address into r_v
(23)
                                       label_{vt}
                                                   ; Clear r_v
                   XORI
                               r_v
(24)
                                                    ; Clear r_m
                   XOR
                                       r_{sp}
(25)
                   ADDI
                                                    ; Deallocate space of main object
                                       size_m
                               r_{sp}
(26)
                   SUBI
                                       offset<sub>stack</sub>; Clear stack pointer
(27)
                   XOR
                                                    ; Clear stack pointer
                   FINISH
(28)
       finish:
                                                    ; Program exit point
```

Figure 4.5: Overall layout of a translated ROOPL++ program

The following PISA code block initializes the free lists pointer, the heap pointer, the stack pointer, allocates the main object on the stack, calls the main method, deallocates the main object and finally clears the free lists, heap and stack pointers.

The free lists pointer is initialized by adding the base address, which varies with the size of the translated program, to the register  $r_{flps}$ . In figure 4.5 the base address is denoted by p.

The heap pointer is initialized directly after the free lists pointer by adding the size of the free lists. One free list is the size of one word and the full size of the free lists is configured in the source code (defaulted to 10, as described earlier).

Once the heap pointer and free lists pointer is initialized, the initial block of free memory is placed in the largest free lists by indexing to said list, by adding the length of the list of free lists, subtracting 1, writing the address of the first block (which is the same address as the heap

pointer, which points to the beginning of the heap) to the last free list and then resetting the free lists pointer to point to the first list again, afterwards.

The stack pointer is initialized simply by adding the stack offset to the heap pointer register  $r_{hp}$ . The stack offset is configured in the source code and defaults to 1024, as described earlier in this chapter. As such, the heap and the stack each have 1024 words of space to utilize. Once the stack pointer has been initialized, the main object is allocated on the stack and the main method called, analogously to the ROOPL program structure.

When the program terminates and the main method returns, the main object is popped from the stack and deallocated and the stack pointer is cleared. The heap and free list pointer not intentionally not cleared to simulate future program simulation using these pointers. The contents of the free lists and whatever is left on the heap is untouched at this point. It is the programmers responsibility to free dynamically allocated objects in their ROOPL++ program. Furthermore, depending on the deallocation order, we might not end up with exactly one fully merged block in the end and as such, we do not invert the steps taken to initialize this initial free memory block. Analogously to ROOPL, the values of the main object are left in the stack section of memory.

## 4.5 Buddy Memory Translation

As briefly mentioned in section 4.2, the Buddy Memory layout was selected as the memory manager layout as it addressed a number of problems related to fragmentation and initialization. The Buddy Memory layout could be converted to a reversible section with only a few restrictions and side effects, which will be described in this section. Firstly, we present the algorithm translated to PISA. As the algorithm is quite lengthy, it will be broken down into smaller chunks. The full translation is shown in appendix A.

The Buddy Memory algorithm consists of three Janus procedures; the entry point **malloc**, the recursion body **malloc1** and a helper function **double**. The entry point is omitted for now, as it differs depending on which type of memory object we are allocating and will be presented in sections 4.6 and 4.8.1. The helper function can be implemented using a single instruction in PISA for our specific case of doubling number in the power-of-two, which we will show later.

(1)	$malloc1_{top}$ :	BRA	$malloc1_{bot}$	; Receive jump		
(2)		POP	$r_{ro}$	; Pop return offset from the stack		
(3)				; Inverse of (7)		
(4)	$malloc1_{entry}$ :	SWAPBR	$r_{ro}$	; Malloc1 entry and exit point		
(5)		NEG	$r_{ro}$	; Negate return offset		
(6)		PUSH	$r_{ro}$	; Store return offset on stack		
(7-63)				; Allocation code		
(64)	$malloc1_{bot}$ :	BRA	$malloc1_{ton}$	; Jump		

Figure 4.6: Dynamic dispatch approach for entering the allocation subroutine

Before we go into depth with the translation of the algorithm, we consider the mechanism for triggering the allocation subroutine. Naively, we could generate the entire block of code required

for allocation for every **new** or **delete** statement in the target program. This approach would severely limit the amount of objects we could allocate as the register pressure of the Buddy Memory implementation is quite high, as we be shown in this section. Instead, we can utilize the dynamic dispatching technique, which also is used for method invocations. This way, we only generate the allocation instructions once, and then simply jump to the entry point from different locations in the program. Figure 4.6 outlines the structure for this approach. By using the **SWAPBR** instruction we can jump from multiple points of origin in the compiled program and internally for the recursive needs of the algorithm itself.

```
; Code for r_{fl} \leftarrow addr(fl[c])
                                                                (8)
                                                                                                                  ; Code for r_{block} \leftarrow \llbracket fl[c] \rrbracket
                                                                (9)
                                                                                                                 ; Code for r_{e1_o} \leftarrow [c_{size} < object_{size}]
                                                                (10)
                                                                                         {\tt XOR} \quad r_t \quad r_{e1_o}
                                                                                                                 ; Copy value of c_{size} < object_{size} into r_t
                                                                                                                 ; Inverse of (9)
                                                                (11)
                                                                                         BEQ r_t r_0 o_{test_f} ; Receive jump
                                                                (12)
                                                                (13)
                                                                                         \mathbf{XORI}\ r_t \quad 1
                                                                                                                 ; Clear r_t
                                                                (14-21)
                                                                                                                  ; Code for outer if-then statement
                                                                                                                 ; Set r_t = 1
                                                                (22)
                                                                                         XORI r_t 1
                                                                (23)
                                                                            o_{assert_t}: BRA o_{assert}
                                                                                                                 : Jump
                                                                                         BRA o_{test}
                                                                (24)
                                                                                                                 ; Receive jump
                                                                (25)
                                                                                                                 : Code for r_{e1} \leftarrow [addr(fl[c]) \neq 0]
                                                                (26)
                                                                                         XOR r_{t2} r_{e1_i}
                                                                                                                  ; Copy value of r_{e1_i} into r_{t2}
                                                                (27)
                                                                                                                 ; Inverse of (25)
                                                                (28)
                                                                                         BEQ r_{t2} r_0 i_{test_f} ; Receive jump
                                                                                                                 ; Clear r_{t2}
1 if (csize < osize) then
2 // outer if-then
                                                                (29)
                                                                                         {f xori}\ r_{t2}\ 1
                                                                                                                  ; Code for inner if-then statement
                                                                (30-34)
                                                                (35)
        if freelists[counter] != 0 then
                                                                                         {\tt XORI}\ r_{t2}\ 1
                                                                                                                 ; Set r_{t2} = 1
              // inner if-then
                                                                                       BRA i_{assert}
                                                                (36)
                                                                                                                 ; Jump
                                                                            i_{assert_t}:
        else
              // inner else
                                                                                         BRA i_{test}
                                                                (37)
                                                                                                                  ; Receive jump
        fi (freelists[counter] = 0 ||
                                                                (38-47)
                                                                                                                 ; Code for inner else statement
(48)
                                                                                         BNE r_{t2} r_0 i_{assert_t}; Receive jump
                                                                (49)
                                                                                         EXCH r_{tmp}r_{fl}
                                                                                                                ; Load address of head of current free list
                                                                                                                 ; Set p to previous block address
                                                                (50)
                                                                                         {\tt SUB} \quad r_p \quad r_{cs}
                                                                (51)
                                                                                                                 ; r_{e2:1} \leftarrow [p - c_{size} \neq addr(fl[c])]
                                                                (52)
                                                                                                                 ; \, r_{e2_{i2}} \,\, \leftarrow \,\, \llbracket addr(fl[c]) = 0 \rrbracket
                                                                (53)
                                                                                                                 ; r_{e2_{i3}} \leftarrow \llbracket (p - c_{size} \neq addr(fl[c])) \lor (addr(fl[c]) = 0) \rrbracket
                                                                (54)
                                                                                         {\tt XOR} \quad r_{r2} \quad r_{e2_{i3}}
                                                                                                             ; Copy value of r_{e2_{i3}} into r_{t2}
                                                                (55)
                                                                                                                 : Inverse of (53)
                                                                (56)
                                                                                                                  ; Inverse of (52)
                                                                (57)
                                                                                                                 : Inverse of (51)
                                                                (58)
                                                                                         ADD r_p r_{cs}
                                                                                                                 ; Inverse of (50)
                                                                                         EXCH r_{tmp}r_{fl}
                                                                (59)
                                                                                                                 : Inverse of (49)
                                                                (60)
                                                                            (61)
                                                                                                                ; Code for r_{e2_o} \leftarrow [c_{size} < object_{size}]
                                                                (62)
                                                                                         {\tt XOR} \quad r_t \quad r_{e2_o}
                                                                                                                 ; Copy value of c_{size} < object_{size} into r_t
                                                                (63)
```

Figure 4.7: PISA translation of the nested conditionals in the Buddy Memory algorithm

The main recursion body of the algorithm, **malloc1** from listing 3.3, page 56 consists of two conditionals, in which one is nested in the else branch of the outer conditional. Figure 4.7 shows the translation structure of the nested conditional pair, using the translation techniques for conditionals presented in [1].

The nested conditionals contain large amounts of boilerplate code for evaluating the various expressions of the conditionals. As these conditionals requires comparisons with contents of the free lists, we must be careful with extracting and storing the values in the free list.

We have three statements to translate from here. The outer **if-then** statement, the inner **if-then** statement and the inner **else** statement.

```
(14) ADDI r_c 1
                                                                                       : Counter++
1 counter += 1
                                                                (15) RL r<sub>sc</sub> 1
                                                                                       ; Call double(c<sub>size</sub>)
2 call double(csize)
                                                                (16) .....
                                                                                       ; Inverse of (7)
3 call malloc1(p, osize, freelists,
                                                                (17) .....
                                                                                       ; Code for pushing temp reg values to stack
                                                                (18) BRA malloc1<sub>entry</sub>
                                                                                       ; Call malloc1())
4
                  counter, csize)
                                                                (19)
                                                                                       ; Inverse of (17)
5 uncall double(csize)
                                                                (20) RR r_{sc} 1
6 counter -= 1
                                                                (21) SUBI r_c 1
                                                                                       ; Inverse of (14)
```

Figure 4.8: PISA translation of the outer if-then statement for the Buddy Memory algorithm

Figure 4.8 shows the translation of the outer **if-then** statement. As briefly mentioned, we can utilize the right bit shift instruction of PISA, **RL**, in place of the **double** helper procedure from the JANUS implementation. By using a simple bit shift, we are able to maintain reversibility elegantly when doubling or halving numbers in the power-of-two. This statement also contains one of the careful storage operations of the free list values, in instruction (16). Before we recursively branch to the entry point, we must place the previously extracted address of the head of the free list back into the free list. This is also the reason for instruction (3) in figure 4.6. Furthermore, we must push all temporary evaluated expression values to the stack, so they can be popped when we return.

Figure 4.9: PISA translation of the inner if-then statement for the Buddy Memory algorithm

Figure 4.9 shows the translation of the inner **if-then** statement. This statement translates easily using the **EXCH** instructions to swap with memory locations as simulated in the JANUS code.

```
(38) ADDI r_c 1
                                                                               ; Counter + +
                                                     (39) RL r_{sc} 1
                                                                               ; Call double(c_{size})
1 counter += 1
                                                    (40) · · · · · ·
                                                                               ; Push temp reg values to stack
2 call double(csize)
                                                    (41) BRA malloc1_{entry}
                                                                               ; Call malloc1())
3 call malloc1(p, osize, freelists,
                                                    (42) ·····
                                                                               ; Inverse of (40)
                  counter, csize)
5 uncall double(csize)
                                                    (43) RR r_{sc} 1
                                                                               ; Inverse of (39)
6 counter -= 1
                                                    (44) SUBI r_c 1
                                                                               : Inverse of (38)
7 freelists[counter] += p
                                                    (45) XOR r_{tmp}r_p
                                                                               ; Copy current address of p
8 p += csize
                                                    (46) EXCH r_{tmp}r_{fl}
                                                                               ; Store address of p in free list
                                                    (47) ADD r_p r_{cs}
                                                                               ; Split block by p= other half of block
```

Figure 4.10: PISA translation of the inner else statement for the Buddy Memory algorithm

The last statement translation is the inner **else** statement shown in figure 4.10. This statement is almost identical to the outer **if-then** with the addition of the block splitting code. The block

splitting is done in three instructions. First, the current block we are examining is set as the new head of the current free list. Afterwards the current free list block size is added to out pointer p, resulting in an effectively split block.

During the design of the reversible Buddy Memory algorithm limitations on the merging and splitting conditions were required to ensure reversibility. Since a split block is always added as two blocks to an empty free list, we can only merge adjacent blocks if they are the only two blocks in a free list. In the irreversible Buddy Memory algorithm block merging can occur in any place of the free list, but in the reversible version, we can only merge blocks at the start of the free list to maintain reversibility. The effect of this limitation prevents us from returning to one final block of free memory, if the deallocation order is not exactly opposite of the allocation order.

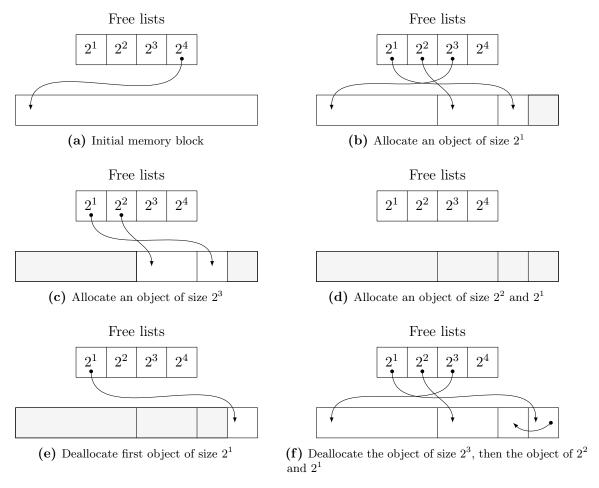


Figure 4.11: Non-opposite deallocation results in a different free list after termination

Figure 4.11 shows how alternative deallocation orders results in different free lists, compared to the original given to some function. However, as discussed in section 3.2, we can consider every collection of Buddy Memory free lists equivalent, as a later computation can take another set of free lists and still execute its function, as long as the free lists have the required blocks available.

## 4.6 Object Allocation and Deallocation

Now that we have the main allocation mechanism in place and a method of accessing it through a label and a **SWAPBR** instruction, we can continue translating the **malloc** procedure entry point from listing 3.3 on page 56.

```
(1)
                                                                 l_{malloc\_top} : BRA
                                                                                                    ; Receive jump
                                                                                       l_{malloc\_bot}
                                                                              SWAPBR r_o
                                                                                                    ; Entry and exit point
                                                            (2)
                                                            (3)
                                                                              NEG
                                                                                                     ; Negate return offset
1 procedure malloc(int p, int osize,
                                                            (4)
                                                                              ADDI
                                                                                       c_{size}
                                                                                                  2 : Init Coize
                            int freelists[])
                                                            (5)
                                                                              XOR
                                                                                                  r_0; Init counter
                                                                                        r_{counter}
        local int counter = 0
3
                                                            (6)
                                                                                                     ; Pop r_p and object_{size} from stack
4
         local int csize = 2
                                                            (7)
                                                                              PIISH
                                                                                                    : Push r<sub>o</sub>
        call malloc1(p, osize, freelists,
                                                                                                    ; call malloc1()
                                                                                       l_{malloc1}
                           counter, csize)
                                                            (9)
                                                                              POP
                                                                                                     ; Inverse of (7)
                                                                                       r_0
         delocal int csize = 2
                                                            (10)
                                                                                                     ; Inverse of (6)
         delocal int counter = 0
                                                            (11)
                                                                              XOR
                                                                                                  r_0; Inverse of (5)
                                                                                       r_{counter}
                                                            (12)
                                                                              SUBI
                                                                                                  2 ; Inverse of (4)
                                                            (13) l_{malloc\_bot}:
                                                                                       l_{malloc\_top}
                                                                                                     ; Jump
```

Figure 4.12: PISA translation of the malloc procedure entry point of Buddy Memory algorithm

Figure 4.12 shows the translated **malloc** procedure. In addition to the original procedure, we also push the current return offset register value to the stack before we branch to the **malloc1** implementation, to ensure we have a zero-cleared register before starting the allocation process. The translated procedure assumes that the pointer to the object we are allocating and its size are on top of the stack before entering the block. This translated procedure serves as the entry point for the allocation subroutine as it is also only generated once. Each **new** and **delete** statement branches to the  $l_{malloc}$  label to begin an allocation or a deallocation.

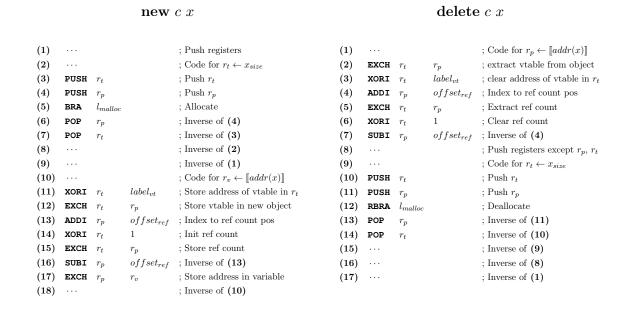


Figure 4.13: PISA translation of heap allocation and deallocation for objects

Figure 4.13 shows how each **new** and **delete** statement for objects are translated during compilation. They are simply inverse of each other. For allocation, the object pointer and its size are pushed to the stack and then a jump to the malloc entry point is executed. After allocation, the virtual table and reference count are stored in the first two words of the allocated memory. Note how deallocation jumps and flips the direction of execution using the **RBRA** instruction, which then runs the allocation process in reverse. In the figure  $x_{size}$  denotes the computed size of objects with class c, plus two, to account for the virtual table pointer and reference count space, rounded up to nearest power-of-two.

#### construct $c \ x - s -$ destruct x

```
(1) XOR r_x r_{sp} ; Store address of new object x in r_x (2) PUSH r_x ; Push r_x to the stack (3) ..... ; Code for new c x (4) ..... ; Code for statement s (5) ..... ; Code for delete c x (6) POP r_x ; Pop r_x from the stack (7) XOR r_x r_{sp} ; Clear r_x
```

Figure 4.14: PISA translation of a ROOPL++ object block

Figure 4.14 shows the updated translation technique for object blocks. In ROOPL, the object blocks allocated their objects on the stack, but in ROOPL++, we can now allocate them on the heap. to facilitate this, we simply execute the exact same instructions as in **new** and **delete** statements, with body statement execution code in between. As described in section 2.6, the **construct/desctruct** block can be considered syntactic sugar, and its usage in a real world example would probably be limited.

# 4.7 Referencing

As mentioned, one of the main strengths of ROOPL++ in terms of increased expressiveness is allowance of multiple references to objects and arrays. When an object or array is constructed we allocate enough space to hold an additional reference counter which is initialized to 1. For each reference copied using the **copy**-statement, we incrementally increase the reference counter by one. When we **uncopy** a reference, the reference counter is decreased by one. The object or array cannot be deconstructed until its reference counter has been returned to 1 as we would have a reference pointer to cleared memory in the heap. Such references are known as dangling pointers.

Figure 4.15 shows the object layout of ROOPL++ objects with the added space for the reference counting from the original ROOPL model in figure 4.2 on page 59.

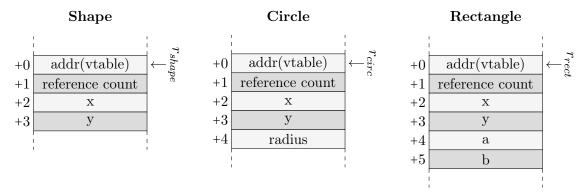


Figure 4.15: Illustration of prefixing in the memory layout of three ROOPL++ objects

Figure 4.16 shows the translated PISA code for the **copy** and **uncopy** statements. As shown, they are both very simple and each others inverse. For copying, the address of the passed variable x is simply copied into the zero-cleared value of x' and the reference count incremented by one. For deletion, the address is cleared and the reference count decremented. Copying and clearing is done through the **XOR** instruction. These translations features no error handling, but a solution is discussed in section 4.9.

```
copy c x x'
                                                                                                        uncopy c \ x \ x'
(1)
                              ; Code for r_p \leftarrow addr(x)
                                                                                    (1)
                                                                                                                  ; Code for r_p \leftarrow addr(x)
(2)
                              ; Code for r_{cp} \leftarrow value(x')
                                                                                    (2)
                                                                                                                  ; Code for r_{cp} \leftarrow value(x')
                              ; Copy address of x into x'
(3)
      XOR
                                                                                    (3)
                                                                                           XOR
                                                                                                                  ; Clear address of x from x'
(4)
      ADDI
                    offset_{ref}; Index to reference count address
                                                                                    (4)
                                                                                           ADDI
                                                                                                        offset_{ref}; Index to reference count address
                             : Extract reference count
                                                                                                                  : Extract reference count
(5)
      EXCH
                                                                                    (5)
                                                                                           EXCH
(6)
      ADDI
                              : Increment reference count
                                                                                    (6)
                                                                                           SUBI
                                                                                                                  : Decrement reference count
(7)
      EXCH r_t
                             ; Store updated reference count
                                                                                    (7)
                                                                                           EXCH
                                                                                                                  ; Store updated reference count
                                                                                                  r_t
                                                                                                        offset_{ref}; Inverse of (3)
(8)
      SUBI
                    offset_{ref}; Inverse of (3)
                                                                                    (8)
                                                                                           SUBI
                             ; Inverse of (2)
                                                                                                                  ; Inverse of (2)
(9)
                                                                                    (9)
(10)
                              : Inverse of (1)
                                                                                    (10)
                                                                                                                  : Inverse of (1)
```

Figure 4.16: Pisa translation of the reference copying and deletion statements

# 4.8 Arrays

The fixed-sized arrays in ROOPL++ are also heap allocated to allow dynamic lifetime. The array memory layout is presented in figure 4.17. As shown, the arrays feature two additional fields to store the size of the array and the reference count. Additionally, integer arrays store their values directly in the array, while object arrays are a simple pointer stores.

As the size of a ROOPL++ array is determined by a passed expression evaluation, it is unknown at compile time. This also means that out-of-bounds checking cannot be conducted during compilation. A possible solution for this is presented in section 4.9.

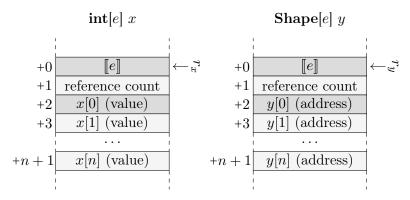


Figure 4.17: Illustration of prefixing in the memory layout of two ROOPL++ arrays

### 4.8.1 Construction and Destruction

As ROOPL++ arrays also are heap allocated, the buddy allocation implementation is also used for allocating arrays. The only difference between object and array allocation is that no virtual table is stored in the allocated space while the offsets for the reference counter are shared for both types. Due to this fact, **copy** and **uncopy** PISA blocks generated during compile time are exactly the same for arrays and objects, as shown in the previous section.

$\mathbf{new}\ a[e]\ x$			<b>delete</b> $a[e]$ $x$				
(1) (2) (3) PUSH $r_t$ (4) PUSH $r_t$ (5) BRA $l_r$ (6) POP $r_t$ (7) POP $r_t$ (10) (11) SUBI $r_t$ (13) ADDI $r_t$ (14) XORI $r_t$ (15) EXCH $r_t$ (16) SUBI $r_t$ (17) EXCH $r_t$ (18)	$r_p$ $r_p$ $r_t$ $r_t$ $r_t$ $r_t$ $r_p$ $r_t$	; Push registers ; Code for $r_t \leftarrow \llbracket e \rrbracket + 2$ ; Push $r_t$ ; Push $r_p$ ; Allocate array ; Inverse of (4) ; Inverse of (3) ; Inverse of (1) ; Code for $r_v \leftarrow \llbracket addr(x) \rrbracket$ ; $r_t \leftarrow \llbracket e \rrbracket$ ; Store size in new array ; Index to ref count pos ; Init ref count ; Store ref count ; Store ref count ; Inverse of (13) ; Store address in variable ; Inverse of (10)	(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14) (15) (16) (17) (18) (19)	ADDI EXCH XORI SUBI EXCH XORI ADDI PUSH PUSH RBRA POP POP SUBI	$r_t$ $r_t$ $r_v$ $r_v$	$offset_{ref}$ $r_p$ 1 $offset_{ref}$ $r_p$ $r_v$ 2	; Code for $r_p \leftarrow \llbracket addr(x) \rrbracket$ ; Code for $r_v \leftarrow \llbracket e \rrbracket$ ; Index to ref count pos; Extract ref count; Clear ref count; Clear ref count; Inverse of (3); extract size from object; clear size in $r_t$ ; Push registers except $r_p$ , $r_v$ ; Actual size of array; Push $r_v$ ; Push $r_p$ ; Deallocate array; Inverse of (12); Inverse of (11); Inverse of (10); Inverse of (9); Inverse of (2); Inverse of (1)

Figure 4.18: PISA translations of array allocation and deallocation statements

Figure 4.18 shows the translation schemes used for array allocation and deallocation. As said, these are almost identical to the object allocation and deallocation schemes presented in figure 4.13 on page 66. Classes are analyzed during a compilation phase and their allocation size, the object size + 2 (for virtual table and reference counter) rounded up to nearest power-of-two. The

size of arrays cannot be determined during compilation, as that would require evaluating the expression passed to the initialization call, and as such, we add the overhead needed directly in the allocation and deallocation instructions. While the two blocks are code are not exact opposites they are functionally inverse of each other. An extra **XORI** instruction on line (8) in the deallocation block has been included to clear the stored array size using the value of the passed expression and further use this size for the inverse **malloc** subroutine.

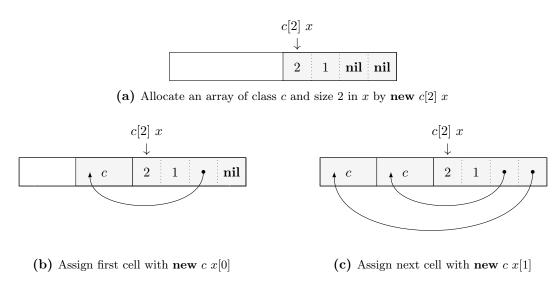


Figure 4.19: Illustration of array memory storage layout

Figure 4.19 shows how object arrays simply contain pointers to allocated objects. For integer arrays, the cell values would stored directly in the allocated array space instead.

### 4.8.2 Array Element Access

Array elements are simply passed as any other variable to methods or statements. Based on the variable type, compilation of various statements individually determines whether the address or the value of the passed variable should be used for the compiling the statement. For arrays, this is no different. If an integer array element is passed, it is treated just liked a regular integer variable. For an object array element, it is treated just like a regular object variable.

# 4.9 Error Handling

While a program written in ROOPL++ might be syntactically valid and well-typed, this is not a guarantee that it executes successfully. A number of conditions exist, which cannot be determined at compile time, which in turn results in erroneous executed code. Haulund describes the following conditions:

• If the entry expression of a conditional is **true**, then the exit assertion should also be **true** after executing the then-branch.

- If the entry expression of a conditional is **false**, then the exit assertion should also be **false** after executing the else-branch.
- The entry expression of a loop should initially be **true**.
- If the exit assertion of a loop is **false**, then the entry expression should also be **false** after executing the loop-statement.
- All instance variables should be zero-cleared within an object block before the object is deallocated.
- The value of a local variable should always match the value of the delocal-expression after the block statement has executed [11].

The extensions made to ROOPL in ROOPL++ brings forth a number of additional conditions:

- All fields of an object instance should be zero-cleared before the object is deallocated using the **delete** statement.
- All cells of an instance should be zero-cleared before the array is deallocated using the **delete** statement.
- Local object blocks should have their fields zero-cleared after the execution of the block statement.
- Local array blocks should have their cells zero-cleared after the execution of the block statement.
- If the value of a local object variable is exchanged during its block statement and the new value is an object reference, this object must have its fields zero-cleared after the execution of the block statement.
- If the value of a local array variable is exchanged during its block statement and the new value is an array reference, this array must have its cell zero-cleared after the execution of the block statement.
- The variable in the **new** statement must be zero-cleared beforehand.
- The variable in the **copy** statement must be zero-cleared beforehand.
- An object variable must be initialized using **new** or **copy** before its methods can be called.
- An array variable must be initialized using **new** or **copy** before its fields can be accessed.
- Array cell indices must be within bounds defined in the expression passed during initialization
- Only one reference to an object or an array must exist when executing the **delete** statement.
- Swapping cell values between a subtype A variable and parent-type B array is only allowed if the value stored in the variable is also A afterwards.

It is the programmer's responsibility to meet these conditions. As these conditions, in general, cannot be determined at compile time, undefined program behaviour will occur as the termination will continue silently, resulting in erroneous program state. We can insert run time error checks in the generated instructions such that the program is terminated if one of the conditions does not

hold. The run time error checks can be added as dynamic error checks using error routines defined at labels, such as  $label_{uninitialized\_object}$  which the program can jump to, if such a condition is unmet. Haulund presented an example for dynamic error checking for local blocks in [11]. PISA and its simulator PendVM is, however, limited and does not support exit codes natively. To fully support dynamic error checking, PendVM could be extended to read from a value from a designated register to supply a more meaningful message for the programmer in the case of a run time exit.

### 4.10 Implementation

The ROOPL++ compiler (ROOPLPPC) was implemented using techniques and translation schemes presented in this chapter, expanding upon the work of the original ROOPL compiler (ROOPLC). The compiler serves as a proof-of-concept and simply performs one-to-one translations of ROOPL++ code to PISA code without any optimizations along the way. The compiler is written in HASKELL 7.10 and the translated output was tested on the Pendulum simulator, PendVM [5].

As with the ROOPL compiler, the ROOPL++ compiler is structured around the same six separate compilation phases.

- 1. **Parsing** consists of constructing an abstract syntax tree from the input program text using parser combinators from the PARSEC library in HASKELL.
- 2. Class Analysis verifies inheritance cycles, duplicated method names or fields and base classes. In this phase, we also compute the allocation size of each class
- 3. **Scope Analysis** constructs the virtual and symbol tables and maps every identifier to a unique variable or method.
- 4. **Type Checking** verifies that the parsed program is well-typed.
- 5. **Code Generation** translates the abstract syntax tree to blocks of PISA code in a recursive descent.
- 6. **Macro Expansion** expands macros left by the code generator for i.e. configuration variables, etc.

Compiled ROOPL programs have a size increase by a factor of 10 to 15 in terms of the lines of code. For ROOPL++ the size increase is much larger, partially due to the increase of static code included in form of the memory manager using the buddy layout described in this chapter and partially because heap allocations are more costly than stack allocations in terms of lines of code.

The ROOPL compiler was implemented in 1403 lines of HASKELL and the ROOPL++ compiler was extended to 2046 lines of HASKELL.

The entire compiler source code as well as example programs and their compiled versions are provided in the appendices and in the supplied ZIP archive. It is also hosted on Github as open source software under the MIT license at https://github.com/cservenka/ROOPLPPC.

Building and usage of the compiler is supplied in the README.md file found in the ZIP archive and in appendix B.

#### 4.11 Evaluation

For evaluating the results of the implemented compiler, it was tested against example code provided throughout this thesis. Tests programs utilizing the linked list, doubly-linked list and binary tree data structures and the RTM implementation are found in appendix C.

Program	ROOPL++ LOC	Pisa LOC	Number of executed instructions
Linked List	61	1280	18015
Doubly-Linked List	66	1339	21825
Binary Tree	86	2056	6065
RTM Simulation	211	6716	64922

Figure 4.20: Lines of code comparison between target and compiled ROOPL++ programs

The linked list test programs simply instantiates ten cells and links them in their respective lists. The binary tree test program instantiates three nodes and adds them to the tree structure, which afterwards is traversed to determine the sum of the nodes and finally mirroring the tree. The Reversible Turing Machine implementing incrementation of a non-negative *n*-bit binary number by 1 originally described in [29] has been implemented in ROOPL++ and successfully converts its initial tape value in little endian form of 1101 to 0011 after termination. It should be noted that these test programs require additional stack space during their lengthy computations and as such has been compiled with twice the length between the stack and heap to allow further stack growth.

As discussed, the compiler is considered proof-of-concept and no noteworthy optimizations has been implemented. However, for the sake of giving the reader an idea of the size blowup of a compiled ROOPL++ program, figure 4.20 details this difference. The lines of translated PISA instructions includes the 204 instructions needed for the **malloc** and **malloc1** PISA-equivalent mechanisms. The last row of the table shows how many instructions are execution during simulation using PendVM.

# Conclusions

We formally presented a dynamic memory management extension for the reversible object-oriented programing language, ROOPL, in the form of the superset language ROOPL++. The extension expands upon the previously presented static typing system defining well-typedness. The language successfully extends the expressiveness of its predecessor by allowing more flexibility within the domain of reversible object-oriented programming. With ROOPL++ we, as reversible programmers, can now define and model non-trivial dynamic data structures in a reversible setting, such as lists, trees and graphs. We illustrated this by example programs such as a new reversible Turing machine simulator along with implementations for linked lists, doubly-linked lists and binary trees as well as techniques for traversing these. Besides expanding the expressiveness of ROOPL, we have also shown that complex dynamic data structures are not only feasible, but furthermore do not contradict the reversible computing paradigm.

We presented various dynamic memory management layouts and how each would translate into the reversible allocation algorithms. Weighing the advantages and disadvantages of each, the Buddy Memory layout was found to translate into reversible code very naturally with few side effects and addressed a number of disadvantages found in other considered layouts. With dynamically lifetimed objects the allocation and deallocation order is important in terms of a entirely garbage-free computation. In most cases with ROOPL++, we only obtain partially garbage-free computations, as our free lists might not be restored to their original form, without an effective garbage collector design for the memory manager.

Techniques for clean translations of extended parts of the language, such as the memory manager, the new fixed-sized array type and reference counting have been demonstrated and implemented in a proof-of-concept compiler for validation.

With the dynamic memory manager for reversible object-oriented programming languages allowing dynamic object-scopes and multiple references, exemplified by ROOPL++, we have successfully taking an additional step in the direction towards high-level abstractions reversible computations.

#### 5.1 Future Work

Naturally with the discovery of feasibility of non-trivial, reversible data structures with the introduction of ROOPL++, further study of design and implementation of reversible algorithms

working with these data structures are an obvious contender for future research. Data structures such as lists, graphs and trees could potentially provide very interesting future reversible programs.

In terms of the future of reversible object-oriented languages, additional work could be made to extend the fixed-sized array type with a fully dynamic array supporting multiple dimensions. This addition could further help the discovery and research of reversible data structures such as trees and graphs. Such an extension could perhaps be added via a **put** and **take** statement pair, being each others inverse. After a dynamic array has been declared, it could automatically reallocate or upscale its internal space when putting new data outside of its current bounds. In reverse, the space could shrink or reallocate when removing the largest indexed value. The current memory management layout will still suffice for this extension.

Finally, more research could be conducted into reversible heap managers. We provided a simple manager which translated to our problem domain naturally. To obtain completely garbage free computations, a garbage collector could be designed to work with the reversible Buddy Memory memory manager. A reversible garbage collector for non-mutable objects has been designed and shown feasible for the reversible functional language RCFUN in [19]. Additionally, experimentation with implementing the Buddy Memory layout into other reversible languages with dynamic allocation and deallocation such as R-WHILE and R-CORE provides an interesting opportunity [8, 9].

## References

- [1] Axelsen, H. B. "Clean Translation of an Imperative Reversible Programming Language". In: Compiler Construction 20th International Conference, CC 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings. Ed. by J. Knoop. Vol. 6601. Lecture Notes in Computer Science. Springer, 2011, pp. 144–163.
- [2] Axelsen, H. B. and Glück, R. "Reversible Representation and Manipulation of Constructor Terms in the Heap". In: Reversible Computation - 5th International Conference, RC 2013, Victoria, BC, Canada, July 4-5, 2013. Proceedings. Ed. by G. W. Dueck and D. M. Miller. Vol. 7948. Lecture Notes in Computer Science. Springer, 2013, pp. 96–109.
- [3] Axelsen, H. B., Glück, R., and Yokoyama, T. "Reversible Machine Code and Its Abstract Processor Architecture". In: Computer Science Theory and Applications, Second International Symposium on Computer Science in Russia, CSR 2007, Ekaterinburg, Russia, September 3-7, 2007, Proceedings. Ed. by V. Diekert, M. V. Volkov, and A. Voronkov. Vol. 4649. Lecture Notes in Computer Science. Springer, 2007, pp. 56–69.
- [4] Baker, H. G. 'Use-Once' Variables and Linear Objects Storage Management, Reflection and Multi-Threading. URL: http://home.pipeline.com/~hbaker1/Use1Var.html (visited on 01/22/2018).
- [5] Clark, C. R. Improving the Reversible Programming Language R and its Supporting Tools. URL: http://www.cise.ufl.edu/research/revcomp/users/cclark/pendvm-fall2001/ (visited on 01/22/2018).
- [6] Digiconomist. "Bitcoin Energy Consumption Index". In: *Digiconomist* (2017-12-06). URL: https://digiconomist.net/bitcoin-energy-consumption (visited on 01/22/2018).
- [7] Frank, M. P. "The R Programming Language and Compiler". MIT Reversible Computing Project Memo #M8. 1997.
- [8] Glück, R. and Yokoyama, T. "A Linear-Time Self-Interpreter of a Reversible Imperative Language". In: *Computer Software* 33.3 (2016), pp. 108–128.
- [9] Glück, R. and Yokoyama, T. "A Minimalist's Reversible While Language". In: *IEICE Transactions* 100-D.5 (2017), pp. 1026–1034.
- [10] Hansen, J. S. K. "Translation of a Reversible Functional Programming Language". Master's Thesis. University of Copenhagen, DIKU, 2014.
- [11] Haulund, T. "Design and Implementation of a Reversible Object-Oriented Programming Language". In: *CoRR* abs/1707.07845 (2017).
- [12] Haulund, T., Mogensen, T. Æ., and Glück, R. "Implementing Reversible Object-Oriented Language Features on Reversible Machines". In: Reversible Computation - 9th International Conference, RC 2017, Kolkata, India, July 6-7, 2017, Proceedings. Ed. by I. Phillips and H. Rahaman. Vol. 10301. Lecture Notes in Computer Science. Springer, 2017, pp. 66-73.

References 76 of 231

- [13] Knuth, D. E. The Art of Computer Programming: Fundamental Algorithms. The Art of Computer Programming. Addison-Wesley, 1998, pp. 435–455. ISBN: 0-201-89683-4.
- [14] Landauer, R. "Irreversibility and Heat Generation in the Computing Process". In: *IBM Journal of Research and Development* 5.3 (1961), pp. 183–191.
- [15] Lee, W. H. and Chang, M. "A study of dynamic memory management in C++ programs". In: Comput. Lang. 28.3 (2002), pp. 237–272.
- [16] Lutz, C. "Janus: a time-reversible language". Letter to R. Landauer. 1986.
- [17] Mitchell, J. C. and Apt, K. Concepts in Programming Languages. New York, NY, USA: Cambridge University Press, 2001. ISBN: 0521780985.
- [18] Mogensen, T. Æ. "Reference Counting for Reversible Languages". In: Reversible Computation 6th International Conference, RC 2014, Kyoto, Japan, July 10-11, 2014. Proceedings. Ed. by S. Yamashita and S. Minato. Vol. 8507. Lecture Notes in Computer Science. Springer, 2014, pp. 82–94.
- [19] Mogensen, T. Æ. "Garbage Collection for Reversible Functional Languages". In: Reversible Computation - 7th International Conference, RC 2015, Grenoble, France, July 16-17, 2015, Proceedings. Ed. by J. Krivine and J. Stefani. Vol. 9138. Lecture Notes in Computer Science. Springer, 2015, pp. 79–94.
- [20] Mogensen, T. Æ. "Programming Language Design and Implementation (Unpublished)".
- [21] Python Software Foundation: Memory Management. URL: https://docs.python.org/ 2/c-api/memory.html (visited on 01/22/2018).
- [22] Schultz, U. P. and Axelsen, H. B. "Elements of a Reversible Object-Oriented Language Work-in-Progress Report". In: Reversible Computation 8th International Conference, RC 2016, Bologna, Italy, July 7-8, 2016, Proceedings. Ed. by S. J. Devitt and I. Lanese. Vol. 9720. Lecture Notes in Computer Science. Springer, 2016, pp. 153–159.
- [23] Thomsen, M. K., Axelsen, H. B., and Glück, R. "A Reversible Processor Architecture and Its Reversible Logic Design". In: Reversible Computation Third International Workshop, RC 2011, Gent, Belgium, July 4-5, 2011. Revised Papers. Ed. by A. D. Vos and R. Wille. Vol. 7165. Lecture Notes in Computer Science. Springer, 2011, pp. 30–42.
- [24] Vance, A. "Inside the Arctic Circle, Where Your Facebook Data Lives". In: *Bloomberg* (2013-12-04). URL: https://www.bloomberg.com/news/articles/2013-10-04/facebooks-new-data-center-in-sweden-puts-the-heat-on-hardware-makers (visited on 01/22/2018).
- [25] Venners, B. The Java Virtual Machine. URL: http://www.artima.com/insidejvm/ed2/jvmP.html (visited on 01/22/2018).
- [26] Vieri, C. J. "Pendulum: A Reversible Computer Architecture". Master's Thesis. University of California at Berkeley, 1993.
- [27] A. D. Vos and R. Wille, eds. Reversible Computation Third International Workshop, RC 2011, Gent, Belgium, July 4-5, 2011. Revised Papers. Vol. 7165. Lecture Notes in Computer Science. Springer, 2012.
- [28] Winskel, G. The Formal Semantics of Programming Languages: An Introduction. Cambridge, MA, USA: MIT Press, 1993.

References 77 of 231

- [29] Yokoyama, T., Axelsen, H. B., and Glück, R. "Principles of a reversible programming language". In: *Proceedings of the 5th Conference on Computing Frontiers*, 2008, Ischia, Italy, May 5-7, 2008. Ed. by A. Ramírez, G. Bilardi, and M. Gschwind. ACM, 2008, pp. 43–54.
- [30] Yokoyama, T., Axelsen, H. B., and Glück, R. "Towards a Reversible Functional Language". In: Reversible Computation Third International Workshop, RC 2011, Gent, Belgium, July 4-5, 2011. Revised Papers. Ed. by A. D. Vos and R. Wille. Vol. 7165. Lecture Notes in Computer Science. Springer, 2011, pp. 14–29.
- [31] Yokoyama, T. and Glück, R. "A reversible programming language and its invertible self-interpreter". In: *Proceedings of the 2007 ACM SIGPLAN Workshop on Partial Evaluation and Semantics-based Program Manipulation, 2007, Nice, France, January 15-16, 2007.* Ed. by G. Ramalingam and E. Visser. ACM, 2007, pp. 144–153.

References 78 of 231

# Pisa Translated Buddy Memory

```
malloc1_{top} :
                                    malloc1_{bot}
(2)
                          POP
                                                           ; Pop return offset from the stack
(3)
                                                          ; Malloc1 entry and exit point
(4)
       malloc1_{entry} : SWAPBR
                                                          ; Negate return offset
(5)
                          PUSH
(6)
                                                           ; Store return offset on stack
(7)
                                                           ; Code for r_{fl} \leftarrow addr(freelists[counter])
(8)
                                                           ; Code for r_{block} \leftarrow [[freelists[counter]]]
(9)
                                                           ; Code for r_{e1_0} \leftarrow \llbracket c_{size} < object_{size} \rrbracket
                                                           ; Copy value of c_{size} < object_{size} into r_t
(10)
                          XOR
                                           r_{e1_o}
                                                           ; Inverse of (9)
(11)
                                           r_0 = o_{test_f} ; Receive jump
(12)
                                                           ; Clear r_t
(13)
                          XORT
(14)
                                                           ; Counter + +
(15)
                          RL
                                         1
                                                          ; Call double(c_{size})
(16)
                                                           ; Inverse of (7)
(17)
                                                           ; Code for pushing temp reg values to stack
                          BRA
                                    malloc1_{entry}
                                                          ; Call malloc1())
(18)
(19)
                                                          ; Inverse of (17)
                                                           ; Inverse of (15)
(20)
                          RR
                                          1
(21)
                                                           ; Inverse of (14)
                                                           ; Set r_t = 1
(22)
                          XORI
                                           1
(23)
                                                           ; Jump
       o_{assert_t} :
                                    o_{assert}
(24)
                                                           ; Receive jump
                                    o_{test}
                                                           ; Code for r_{e1_i} \leftarrow [addr(freelists[counter]) \neq 0]
(25)
(26)
                          XOR
                                    r_{t2}
                                          r_{e1_i}
                                                           ; Copy value of r_{e1_i} into r_{t2}
(27)
                                                           ; Inverse of (25)
(28)
                          BEQ
                                           r_0 = i_{test_f}; Receive jump
                                    r_{t2}
(29)
                          XORI
                                           1
                                                           ; Clear r_{t2}
                                   r_{t2}
(30)
                          ADD
                                           r_{block}
                                                           ; Copy address of the current block to p
                                                          ; Clear r_{block}
                                    r_{block} r_p
(32)
                          EXCH
                                                          ; Load address of next block
                                                           ; Set address of next block as new head of free list
(33)
                                    r_{tmp} r_{fl}
(34)
                                                           ; Clear address of next block
                                    r_{tmp} r_p
(35)
                          XORT
                                                           ; Set r_{t2} = 1
(36)
                                                           ; Jump
       i_{assert_t}:
                                    i_{assert}
(37)
                                                           ; Receive jump
       i_{test_f} :
                                    i_{test}
                                                           ; Counter + +
(38)
                          ADDI
(39)
                                                           ; Call double(c<sub>size</sub>)
                          RL
(40)
                                                           ; Code for pushing temp reg values to stack
(41)
                          BRA
                                    malloc1_{entry}
                                                           : Call malloc1())
(42)
                                                           ; Inverse of (40)
                                                           ; Inverse of (39)
(43)
                          RR
(44)
                          SUBI
                                                           ; Inverse of (38)
```

```
(45)
                             XOR
                                                                  ; Copy current address of p
                                        r_{tmp} r_p
(46)
                                                                  ; Store current address of p in current free list
                             EXCH
                                        r_{tmp}
                                                r_{fl}
                                                                  ; Split block by setting p to second half of current block
(47)
                             ADD
                                         r_p
                                                r_{cs}
                                                        i_{assert_t}; Receive jump
(48)
                             BNE
        i_{assert}:
                                                r_0
                                        r_{t2}
                                                                  ; Load address of head of current free list
(49)
                             EXCH
                                        r_{tmp} r_{fl}
                                                                  ; Set p to previous block address
(50)
                             SUB
                                                                  ; Code for {r_e}_{2i1} \ \leftarrow \ [\![p-c_{size} \neq addr(free lists[counter])]\!]
(51)
(52)
                                                                  ; Code for r_{e2}_{i2} \leftarrow [addr(freelists[counter]) = 0]
(53)
                                                                  ; Code for r_{e2}_{i3} \leftarrow \llbracket (p - c_{size} \neq addr(freelists[counter])) \lor (addr(freelists[counter]) = 0) \rrbracket
                                                                  ; Copy value of {r_{e2}}_{i3} into {r_{t2}}
(54)
                             XOR
                                                r_{e2}{}_{i3}
                                        r_{r2}
                                                                  ; Inverse of (53)
(55)
(56)
                                                                  ; Inverse of (52)
                                                                  ; Inverse of (51)
(57)
(58)
                             ADD
                                                 r_{cs}
                                                                  ; Inverse of (50)
                                        r_p
(59)
                                                                  ; Inverse of (49)
                             EXCH
                                                r_{fl}
                                                        o_{assert_t}; Receive jump
(60)
                                                 r_0
(61)
                                                                  ; Code for r_{e2o} \ \leftarrow \ \llbracket c_{size} < object_{size} \rrbracket
                                                                  ; Copy value of c_{size} < object_{size} into r_t
(62)
                             XOR
                                                 r_{e2_o}
(63)
                                                                  ; Inverse of (61)
(64)
       malloc1_{bot} :
                                        malloc1_{top}
                                                                  ;\;\mathrm{Jump}
```

# ROOPLPPC Source Code

#### README.md

```
1 # ROOPLPPC
 3 *ROOPLPP* is a compiler translating source code written in **Reversible Object Oriented
      Programming Language++** (ROOPL++) to the reversible assembly language Pendulum ISA (PISA)
 5 The compiler is to be considered proof-of-concept in connection with my Master's Thesis on the
       ROOPL++ language.
 6
 7 ## Requirements
 8 ROOPLPPC uses [Stack](https://docs.haskellstack.org/en/stable/README/) to manage all
      dependencies and requirements.
10 ## Building
11 Simply invoke
12
13 stack build
14
15 which compiles an executable into the '.stack-work' folder
16
18 To compile a ROOPL++ program simply run
19
20 stack exec ROOPLPPC input.rplpp
21
22 which compiles the input program into Pisa and stores the compiled file as 'input.pal' in the
      current directory.
24 To specify an output file name, simply provide it as an additional argument
25
26 stack exec ROOPLPP input.rplpp output.pal
28
29 ## Examples
30 To see usage examples, please refer to 'test/' for example programs.
32 ## Running compiled programs
33 The PendVM simulator executes compiled Pisa code and is hosted on Github [here](https://github
      .com/TueHaulund/PendVM).
```

#### AST.hs

```
1 module AST where
3 import Text.Show.Pretty
5 {-- AST Primitives --}
6 type TypeName = String
8 type MethodName = String
10 data DataType = IntegerType
                  | ObjectType TypeName
11
                  | CopyType TypeName
12
                  |Â ObjectArrayType TypeName
                  | IntegerArrayType
14
15
                  | ArrayType
                  | ArrayElementType
                  | NilType
17
18
    deriving (Show)
19
20 -- Types
21 instance Eq DataType where
    IntegerType == IntegerType = True
23
    IntegerArrayType == IntegerArrayType = True
    NilType == NilType = True
    NilType == (ObjectType _) = True
25
26
     (ObjectType _) == NilType = True
     (ObjectType t1) == (ObjectType t2) = t1 == t2
27
     (CopyType t1) == (CopyType t2) = t1 == t2
28
29
     (ObjectArrayType t1) == (ObjectArrayType t2) = t1 == t2
    (CopyType t1) == (ObjectType t2) = t1 == t2
(ObjectType t1) == (CopyType t2) = t1 == t2
30
31
    ArrayType == (ObjectArrayType _) = True
    (ObjectArrayType _) == ArrayType = True
33
34
    ArrayType == IntegerArrayType = True
    IntegerArrayType == ArrayType = True
35
    _ == _ = False
36
37
38 -- Binary Operators
39 data BinOp = Add
              | Sub
               | Xor
41
42
              | Mul
              | Div
43
              1 Mod
44
45
              | BitAnd
46
              | BitOr
47
              | And
               | Or
              | Lt
49
50
               | Gt
               | Eq
51
52
               | Neg
53
               | Lte
54
               | Gte
55
    deriving (Show, Eq, Enum)
57 data ModOp = ModAdd
58
              | ModSub
               | ModXor
59
    deriving (Show, Eq, Enum)
60
62 {-- Generic AST Definitions --}
63 --Expressions
```

```
64 data GExpr v = Constant Integer
                 | Variable v
65
66
                 | ArrayElement (v, GExpr v)
67
                 | Binary BinOp (GExpr v) (GExpr v)
68
69
     deriving (Show, Eq)
70
71 --Statements
72 data GStmt m v = Assign v ModOp (GExpr v)
                   |Â AssignArrElem (v, GExpr v) ModOp (GExpr v)
73
74
                   | Swap (v, Maybe (GExpr v)) (v, Maybe (GExpr v))
                   | Conditional (GExpr v) [GStmt m v] [GStmt m v] (GExpr v)
75
                   | Loop (GExpr v) [GStmt m v] [GStmt m v] (GExpr v)
76
77
                   | ObjectBlock TypeName v [GStmt m v]
                   | LocalBlock DataType v (GExpr v) [GStmt m v] (GExpr v)
78
                   | LocalCall m [(v, Maybe (GExpr v))]
79
80
                   | LocalUncall m [(v, Maybe (GExpr v))]
                   | ObjectCall (v, Maybe (GExpr v)) MethodName [(v, Maybe (GExpr v))]
81
82
                   | ObjectUncall (v, Maybe (GExpr v)) MethodName [(v, Maybe (GExpr v))]
83
                   | ObjectConstruction TypeName (v, Maybe (GExpr v))
                   | ObjectDestruction TypeName (v, Maybe (GExpr v))
84
                   | CopyReference DataType (v, Maybe (GExpr v)) (v, Maybe (GExpr v))
85
86
                   | UnCopyReference DataType (v, Maybe (GExpr v)) (v, Maybe (GExpr v))
                   \mid ArrayConstruction (TypeName, GExpr v) v
87
                   | ArrayDestruction (TypeName, GExpr v) v
89
                   | Skip
90
     deriving (Show, Eq)
92 --Field/Parameter declarations
93 data GDecl v = GDecl DataType v
     deriving (Show, Eq)
95
96 -- Method: Name, parameters, body
97 data GMDecl m v = GMDecl m [GDecl v] [GStmt m v]
98
     deriving (Show, Eq)
100 --Class: Name, fields, methods
101 data GCDecl m v = GCDecl TypeName (Maybe TypeName) [GDecl v] [GMDecl m v]
102
     deriving (Show, Eq)
103
104 --Program
105 newtype GProg m v = GProg [GCDecl m v]
    deriving (Show, Eq)
106
108 {-- Specific AST Definitions --}
109 -- Plain AST
110 type Identifier = String
111
112 type Expression = GExpr Identifier
113
114 type Statement = GStmt MethodName Identifier
115
116 type VariableDeclaration = GDecl Identifier
117
118 type MethodDeclaration = GMDecl MethodName Identifier
119
120 type ClassDeclaration = GCDecl MethodName Identifier
121
122 type Program = GProg MethodName Identifier
123
124 -- Scoped AST
125 type SIdentifier = Integer
127 type SExpression = GExpr SIdentifier
128
129 type SStatement = GStmt SIdentifier SIdentifier
```

```
131 type SVariableDeclaration = GDecl SIdentifier
132
133 type SMethodDeclaration = GMDecl SIdentifier SIdentifier
134
135 type SProgram = [(TypeName, GMDecl SIdentifier SIdentifier)]
136
137 {-- Other Definitions --}
138 type Offset = Integer
139
140 data Symbol = LocalVariable DataType Identifier
141
               | ClassField DataType Identifier TypeName Offset
               | MethodParameter DataType Identifier
142
143
               | Method [DataType] MethodName
    deriving (Show, Eq)
144
145
146 type SymbolTable = [(SIdentifier, Symbol)]
147
148 type Scope = [(Identifier, SIdentifier)]
150 printAST :: (Show t) => t -> String
151 printAST = ppShow
```

### PISA.hs

```
1 {-# LANGUAGE FlexibleInstances, TypeSynonymInstances #-}
3 module PISA where
5 import Data.List (intercalate)
6 import Control.Arrow
8 import AST (TypeName, MethodName)
10 type Label = String
11
12 newtype Register = Reg Integer
      deriving (Eq)
14
15 {-- Generic PISA Definitions --}
17 data GInstr i = ADD Register Register
18
                 | ADDI Register i
19
                 | ANDX Register Register Register
                 | ANDIX Register Register i
20
21
                 | NORX Register Register Register
                 | NEG Register
23
                 | ORX Register Register Register
                 | ORIX Register Register i
                 | RL Register i
25
26
                 | RLV Register Register
27
                 | RR Register i
                 | RRV Register Register
28
29
                 | SLLX Register Register i
30
                 | SLLVX Register Register Register
                 | SRAX Register Register i
31
                 | SRAVX Register Register Register
                 | SRLX Register Register i
33
                 | SRLVX Register Register Register
34
                 | SUB Register Register
35
36
                 | XOR Register Register
37
                 | XORI Register i
                 | BEQ Register Register Label
38
39
                 | BGEZ Register Label
40
                 | BGTZ Register Label
                 | BLEZ Register Label
41
42
                 | BLTZ Register Label
43
                 | BNE Register Register Label
                 | BRA Label
44
45
                 | EXCH Register Register
46
                 | SWAPBR Register
47
                 | RBRA Label
                 | START
                 | FINISH
49
50
                 I DATA i
                 | SUBI Register i --Pseudo
51
      deriving (Eq)
52
53
54 newtype GProg i = GProg [(Maybe Label, GInstr i)]
55
56 {-- Macro PISA Definitions --}
57
58 data Macro = Immediate Integer
              | AddressMacro Label
              | SizeMacro TypeName
60
              | OffsetMacro TypeName MethodName
              | ProgramSize
62
              | FreeListsSize
63
```

```
| StackOffset
                | InitialMemoryBlockSize
 65
 66
                | ReferenceCounterIndex
 67
                | ArrayElementOffset
 68
        deriving (Show, Eq)
 69
 70 type MInstruction = GInstr Macro
71 type MProgram = GProg Macro
 73 invertInstructions :: [(Maybe Label, MInstruction)] -> [(Maybe Label, MInstruction)]
 74 invertInstructions = reverse . map (second invertInstruction . first (fmap (++ "_i")))
        where invertInstruction (ADD r1 r2) = SUB r1 r2
76
               invertInstruction (SUB r1 r2) = ADD r1 r2
 77
               invertInstruction (ADDI r i) = SUBI r i
               invertInstruction (SUBI r i) = ADDI r i
 78
               invertInstruction (RL r i) = RR r i
 79
               invertInstruction (RLV r1 r2) = RRV r1 r2
               invertInstruction (RR r i) = RL r i
 81
 82
               invertInstruction (RRV r1 r2) = RLV r1 r2
               invertInstruction (BEQ r1 r2 l) = BEQ r1 r2 $ 1 ++ "_i"
               invertInstruction (BGEZ r l) = BGEZ r $ l ++ "_i"
 84
               invertInstruction (BGTZ r 1) = BGTZ r $ 1 ++ "_i"
               invertInstruction (BLEZ r l) = BLEZ r $ 1 ++ "
 86
               invertInstruction (BLTZ r l) = BLTZ r $ 1 ++ "_i"
 87
               invertInstruction (BNE r1 r2 l) = BNE r1 r2 $ 1 ++ "_i"
               invertInstruction (BRA 1) = BRA $ 1 ++ "_i"
 89
               invertInstruction (RBRA 1) = RBRA $ 1 ++ "_i"
 90
               invertInstruction inst = inst
 92
 93 {-- Output PISA Definitions --}
 95 type Instruction = GInstr Integer
 96 type Program = GProg Integer
98 instance Show Register where
99
        show (Reg r) = "$" ++ show r
100
101 instance Show Instruction where
        show (ADD r1 r2) = unwords ["ADD
                                               ", show r1, show r2]
102
        show (ADDI r i) = unwords ["ADDI ", show r, show i]
103
        show (ANDX r1 r2 r3) = unwords ["ANDX ", show r1, show r2, show r3]
104
        show (ANDIX r1 r2 i) = unwords ["ANDIX ", show r1, show r2, show i]
105
        show (NORX r1 r2 r3) = unwords ["NORX ", show r1, show r2, show r3]
106
        show (NEG r) = unwords ["NEG ", show r]
107
        show (ORX r1 r2 r3) = unwords ["ORX ", show r1, show r2, show r3]
show (ORIX r1 r2 i) = unwords ["ORIX ", show r1, show r2, show i]
108
109
                                            ", show r, show i]
", show r1, show r2]
110
        show (RL r i) = unwords ["RL
        show (RLV r1 r2) = unwords ["RLV
111
        show (RR r i) = unwords ["RR ", show r, show i]
112
        show (RRV r1 r2) = unwords ["RRV ", show r1, show r2]
show (SLLX r1 r2 i) = unwords ["SLLX ", show r1, show r2, show i]
show (SLLVX r1 r2 r3) = unwords ["SLLVX ", show r1, show r2, show r3]
113
114
115
        show (SRAX r1 r2 i) = unwords ["SRAX ", show r1, show r2, show i]
show (SRAVX r1 r2 r3) = unwords ["SRAVX ", show r1, show r2, show r3]
116
117
        show (SRLX r1 r2 i) = unwords ["SRLX ", show r1, show r2, show i]
118
        show (SRLVX r1 r2 r3) = unwords ["SRLVX ", show r1, show r2, show r3]
119
        show (SUB r1 r2) = unwords ["SUB ", show r1, show r2]
120
                                               ", show r1, show r2]
121
        show (XOR r1 r2) = unwords ["XOR
                                              ", show r, show i]
        show (XORI r i) = unwords ["XORI
122
        show (BEQ r1 r2 1) = unwords ["BEQ ", show r1, show r2, 1]
123
        show (BGEZ r l) = unwords ["BGEZ ", show r, l]
124
                                              ", show r, 1]
125
        show (BGTZ r l) = unwords ["BGTZ
        show (BLEZ r l) = unwords ["BLEZ
                                              ", show r, 1]
        show (BLTZ r l) = unwords ["BLTZ ", show r, l] show (BNE r1 r2 l) = unwords ["BNE ", show r1, show r2, l]
127
128
        show (BRA 1) = unwords ["BRA ", 1]
129
```

```
show (EXCH r1 r2) = unwords ["EXCH ", show r1, show r2]
        show (SWAPBR r) = unwords ["SWAPBR", show r]
131
        show (RBRA 1) = unwords ["RBRA ", 1]
132
133
        show START = "START "
        show FINISH = "FINISH"
134
        show (DATA i) = unwords ["DATA ", show i] show (SUBI r i) = unwords ["ADDI ", show r, show $ -i] --Expand pseudo
135
136
137
138 showProgram :: Program -> String
139 showProgram (GProg p) = ";; pendulum pal file\n" ++ intercalate "\n" (map showLine p)
        where showLine (Nothing, i) = spaces 25 ++ show i
    showLine (Just 1, i) = 1 ++ ":" ++ spaces (24 - length 1) ++ show i
140
141
                spaces :: (Int -> String)
spaces n = [1..n] >> " "
142
143
144
145 writeProgram :: String \rightarrow Program \rightarrow IO ()
146 writeProgram output p = writeFile output $ showProgram p
```

#### Parser.hs

```
1 module Parser (parseString) where
 3 import Control.Monad.Except
 4 import Data.Functor. Identity
 5 import Data.Bifunctor
 7 import Text.Parsec
 8 import Text.Parsec.String
9 import Text.Parsec.Expr
10 import Text.Parsec.Language
11 import qualified Text.Parsec.Token as Token
13 import Debug.Trace (trace, traceShow)
14
15 import AST
17 {-- Language Definition --}
18 keywords :: [String]
19 keywords =
       ["class",
"inherits",
20
21
         "method",
22
23
         "call",
24
         "uncall"
         "construct",
25
         "destruct",
26
         "skip",
27
         "from",
28
29
         "do",
         "loop",
30
         "until",
31
         "int",
         "nil",
33
         "if",
34
         "then",
35
         "else",
36
         "fi",
37
         "local",
38
         "delocal",
39
40
         "new",
         "delete",
41
42
         "copy",
         "uncopy"]
43
44
45 --Operator precedence identical to C
46 operatorTable :: [[(String, BinOp)]]
47 operatorTable =
       [ [("*", Mul), ("/", Div), ("%", Mod)],
  [("+", Add), ("-", Sub)],
  [("<", Lt), ("<=", Lte), (">", Gt), (">=", Gte)],
49
50
          [("=", Eq), ("!=", Neq)],
51
          [("&", BitAnd)],
[("^", Xor)],
52
53
          [("|", BitOr)],
54
          [("&&", And)],
[("||", Or)]]
55
57
58 languageDef :: Token.LanguageDef st
59 languageDef =
       emptyDef {
60
                                     = "//",
61
            Token.commentLine
            Token.nestedComments = False,
62
                                      = letter,
            Token.identStart
63
```

```
Token.identLetter
                                 = alphaNum <|> oneOf "_'",
65
           Token.reservedOpNames = concatMap (map fst) operatorTable,
66
           Token.reservedNames = keywords,
67
           Token.caseSensitive
                                  = True }
68
69 tokenParser :: Token.TokenParser st
70 tokenParser = Token.makeTokenParser languageDef
72 {-- Parser Primitives --}
73 identifier :: Parser String
74 identifier = Token.identifier tokenParser
76 arrElemIdentifier :: Parser (String, Expression)
77 arrElemIdentifier = do x <- identifier
                           y <- brackets expression
78
79
                           return (x, y)
81 anyIdentifier :: Parser (String, Maybe Expression)
82 anyIdentifier = do x <- identifier
                      y <- optionMaybe $ brackets expression
84
                       return (x, y)
86 reserved :: String -> Parser ()
87 reserved = Token.reserved tokenParser
89 reservedOp :: String -> Parser ()
90 reservedOp = Token.reservedOp tokenParser
92 integer :: Parser Integer
93 integer = Token.integer tokenParser
95 symbol :: String -> Parser String
96 symbol = Token.symbol tokenParser
98 parens :: Parser a -> Parser a
99 parens = Token.parens tokenParser
100
101 brackets :: Parser a -> Parser a
102 brackets = Token.brackets tokenParser
103
104 colon :: Parser String
105 colon = Token.colon tokenParser
106
107 commaSep :: Parser a -> Parser [a]
108 commaSep = Token.commaSep tokenParser
109
110 typeName :: Parser TypeName
111 typeName = identifier
112
113 arrayTypeName :: Parser (TypeName, Expression)
114 arrayTypeName = do x <- try typeName <|> string "int"
                       y <- brackets expression
115
116
                       return (x, y)
117
118 methodName :: Parser MethodName
119 methodName = identifier
120
121 {-- Expression Parsers --}
122 constant :: Parser Expression
123 constant = Constant <$> integer
124
125 variable :: Parser Expression
126 variable = Variable <$> identifier
127
128 arrayElementVariable :: Parser Expression
129 arrayElementVariable = ArrayElement <$> arrElemIdentifier
```

```
131 nil :: Parser Expression
132 nil = Nil <$ reserved "nil"
134 expression :: Parser Expression
135 expression = buildExpressionParser opTable $ constant <|> try arrayElementVariable <|>
       variable <|> nil
       where binop (t, op) = Infix (Binary op <$ reservedOp t) AssocLeft</pre>
136
137
             opTable = (map . map) binop operatorTable
138
139 {-- Statement Parsers --}
140 modOp :: Parser ModOp
141 modOp = ModAdd < $ symbol "+="
       <|> ModSub <$ symbol "-="
142
       <|> ModXor <$ symbol "^="
143
144
145 assign :: Parser Statement
146 assign = Assign <> identifier <*> modOp <*> expression
147
148 assignArrElem :: Parser Statement
149 assignArrElem = AssignArrElem <$> arrElemIdentifier <*> modOp <*> expression
150
151 swap :: Parser Statement
152 swap = Swap <$> anyIdentifier <* symbol "<=>" <*> anyIdentifier
154 conditional :: Parser Statement
155 conditional =
      reserved "if"
157
       >> Conditional
158
       <$> expression
       <* reserved "then"
159
160
       <*> block
161
       <* reserved "else"
162
       <*> block
       <* reserved "fi"
163
164
       <*> expression
165
166 loop :: Parser Statement
167 loop =
       reserved "from"
168
169
       >> Loop
170
       <$> expression
       <* reserved "do"
171
172
       <*> block
       <* reserved "loop"
173
174
       <*> block
       <* reserved "until"
175
       <*> expression
176
177
178 localCall :: Parser Statement
179 localCall =
180
       reserved "call"
       >> LocalCall
181
182
       <$> methodName
       <*> parens (commaSep anyIdentifier)
183
184
185 localUncall :: Parser Statement
186 localUncall =
       reserved "uncall"
187
188
       >> LocalUncall
       <$> methodName
189
190
       <*> parens (commaSep anyIdentifier)
192 objectCall :: Parser Statement
193 objectCall =
      reserved "call"
194
```

```
>> ObjectCall
       <$> anyIdentifier
196
197
       <* colon
198
        <* colon
       <*> methodName
199
200
       <*> parens (commaSep anyIdentifier)
201
202 objectUncall :: Parser Statement
203 objectUncall =
       reserved "uncall"
204
       >> ObjectUncall
205
       <$> anyIdentifier
       <* colon
207
208
       <* colon
       <*> methodName
209
       <*> parens (commaSep anyIdentifier)
210
211
212 objectConstruction :: Parser Statement
213 objectConstruction =
214
       reserved "new"
       >> ObjectConstruction
215
^{216}
       <$> typeName
217
       <*> anyIdentifier
218
219 objectDestruction :: Parser Statement
220 objectDestruction =
       reserved "delete"
221
       >> ObjectDestruction
       <$> typeName
223
       <*> anyIdentifier
224
226 localBlock :: Parser Statement
227 localBlock =
      reserved "local"
228
       >> LocalBlock
229
230
       <$> dataType
       <*> identifier
231
232
       <* symbol "="
       <*> expression
233
       <*> block
234
235
       <* reserved "delocal"</pre>
       <* dataType
236
       <* identifier
237
238
       <* symbol "="
       <*> expression
239
240
241 objectBlock :: Parser Statement
242 objectBlock =
       reserved "construct"
243
       >> ObjectBlock
244
245
       <$> typeName
246
       <*> identifier
       <*> block
247
       <* reserved "destruct"
248
       <* identifier
249
250
251 skip :: Parser Statement
252 skip = Skip <$ reserved "skip"
253
254 copyReference :: Parser Statement
255 copyReference =
256 reserved "copy"
257
       >> CopyReference
       <$> dataType
258
259
       <*> anyIdentifier
       <*> anyIdentifier
260
```

```
262 unCopyReference :: Parser Statement
263 unCopyReference =
264
       reserved "uncopy"
       >> UnCopyReference
265
266
       <$> dataType
267
       <*> anyIdentifier
       <*> anyIdentifier
268
270 arrayConstruction :: Parser Statement
271 arrayConstruction =
       reserved "new"
       >> ArrayConstruction
273
274
       <$> arrayTypeName
       <*> identifier
275
276
277 arrayDestruction :: Parser Statement
278 arrayDestruction =
279
       reserved "delete"
280
       >> ArrayDestruction
       <$> arrayTypeName
281
282
       <*> identifier
283
284 statement :: Parser Statement
285 statement = try assign
           <|> try assignArrElem <|> swap
286
           <|> conditional
287
           <|> loop
           <|> try localCall
289
290
           <|> try localUncall
           <|> objectCall
291
292
           <|> objectUncall
293
           <|> localBlock
294
           <|> objectBlock
           <|> try arrayConstruction <|> objectConstruction
295
296
           <|> try arrayDestruction <|> objectDestruction
           <|> skip
297
298
           <|> copyReference
299
           <|> unCopyReference
300
301 block :: Parser [Statement]
302 block = many1 statement
303
304 {-- Top Level Parsers --}
305 dataType :: Parser DataType
306 dataType = try (IntegerArrayType <$ reserved "int" <* symbol "[" <* symbol "]")
              <|> IntegerType <$ reserved "int"
          <|> try (ObjectArrayType <$> typeName <* symbol "[" <* symbol "]")</pre>
308
309
               <|> ObjectType <$> typeName
310
311
312 variableDeclaration :: Parser VariableDeclaration
313 variableDeclaration = GDecl <$> dataType <*> identifier
314
315 methodDeclaration :: Parser MethodDeclaration
316 methodDeclaration =
^{317}
       reserved "method"
       >> GMDecl
318
       <$> methodName
319
       <*> parens (commaSep variableDeclaration)
320
       <*> block
321
322
323 classDeclaration :: Parser ClassDeclaration
324 classDeclaration =
325
       reserved "class"
326
       >> GCDecl
```

### ClassAnalyzer.hs

```
1 {-# LANGUAGE GeneralizedNewtypeDeriving, FlexibleContexts #-}
3 module ClassAnalyzer
    ( classAnalysis
    , printCAState
    , CAState(..)
    ) where
9 import Data.List
10 import Data.Maybe
12 import Control.Monad
13 import Control.Monad.Except
14 import Control.Monad.State
15 import Text.Pretty.Simple (pPrint)
17 import Debug.Trace (trace, traceShow)
19 import AST
20
21 type Size = Integer
23 -- | The Class Analyzer State consists of a list of classes, sizes, methods
24 -- | and a main class
25 data CAState = CAState {
      classes :: [(TypeName, ClassDeclaration)],
27
      subClasses :: [(TypeName, [TypeName])],
      superClasses :: [(TypeName, [TypeName])],
28
      classSize :: [(TypeName, Size)],
29
30
      classMethods :: [(TypeName, [MethodDeclaration])],
      mainClass :: Maybe TypeName
31
32 } deriving (Show, Eq)
33
34 -- | The Class Analyzer monad
35 newtype ClassAnalyzer a = ClassAnalyzer { runCA :: StateT CAState (Except String) a }
      deriving (Functor, Applicative, Monad, MonadState CAState, MonadError String)
36
38 -- | Initializes the Class Analyzer State with empty lists and Nothing for the mainClass
39 initialState :: CAState
40 initialState = CAState {
     classes = [],
41
42
      subClasses = [],
43
      superClasses = [],
      classSize = [],
44
45
      classMethods = [];
46
      mainClass = Nothing
47 }
49 -- | Returns a class from the Class Analyzer State if passed typename matches
50 getClass :: TypeName -> ClassAnalyzer ClassDeclaration
51 getClass n = gets classes >>= \cs ->
52
      case lookup n cs of
53
          (Just c) -> return c
          Nothing -> throwError $ "ICE: Unknown class " ++ n
56 -- | Returns the base class inherited from
57 getBaseClass :: TypeName -> ClassAnalyzer (Maybe TypeName)
58 getBaseClass n = getClass n >>= getBase
      where getBase (GCDecl _ b _ _) = return b
61 -- | Throws error if class is defined multiple times
62 checkDuplicateClasses :: ClassDeclaration -> ClassAnalyzer ()
63 checkDuplicateClasses (GCDecl n \_ \_ ) = gets classes >>= \cs ->
```

```
when (count cs > 1) (throwError $ "Multiple definitions of class " ++ n)
       where count = length . filter ((== n) . fst)
65
66
67 -- | Ensures legal inheritance
68 checkBaseClass :: ClassDeclaration -> ClassAnalyzer ()
69 checkBaseClass (GCDecl _ Nothing _ _) = return ()
71
          cs <- gets classes
72
          when (isNothing \$ lookup b cs) (throwError \$ "Class " ++ n ++ " cannot inherit from
73
               unknown class " ++ b)
75 -- |Â Checks duplicated field declarations
76 checkDuplicateFields :: ClassDeclaration -> ClassAnalyzer ()
77 checkDuplicateFields (GCDecl n _ fs _) = mapM_ checkField fs
       where count v = length . filter (\((GDecl \_v') \rightarrow v' == v) $ fs
78
79
             checkField (GDecl \_ v) = when (count v > 1) (throwError $ "Multiple declarations of
                  field " ++ v ++ " in class " ++ n)
80
81 -- | Checks duplicated method declaration in classes
82 checkDuplicateMethods :: ClassDeclaration -> ClassAnalyzer ()
83 checkDuplicateMethods (GCDecl n _ _ ms) = mapM_ checkMethod ms'
84 where ms' = map (\((GMDecl n' _ _) -> n')\) ms
85 count m = length . filter (== m) $ ms'
              \texttt{checkMethod} \ \texttt{m} \ = \ \textbf{when} \ (\texttt{count} \ \texttt{m} \ > \ 1) \ (\texttt{throwError} \ \$ \ \texttt{"Multiple} \ \texttt{definitions} \ \texttt{of} \ \texttt{method} \ \texttt{"}
                  ++ m ++ " in class " ++ n)
87
88 -- | Checks cyclic inheritance
89 checkCyclicInheritance :: ClassDeclaration -> ClassAnalyzer ()
90 checkCyclicInheritance (GCDecl _ Nothing _ _) = return ()
91 checkCyclicInheritance (GCDecl n b \_ \_) = checkInheritance b [n]
       where checkInheritance Nothing \_ = return ()
92
              checkInheritance (Just b') visited =
93
                  do when (b' 'elem' visited) (throwError $ "Cyclic inheritance involving class "
94
                      ++ n)
                     next <- getBaseClass b'</pre>
                     checkInheritance next (b' : visited)
96
98 -- | Sets the main class in the Class Analyzer State
99 setMainClass :: ClassDeclaration -> ClassAnalyzer ()
100 setMainClass (GCDecl n _ _ ms) = when ("main" 'elem' ms') (gets mainClass >>= set)
101
       where
           ms' = map (\(GMDecl n' _ _) -> n') ms
102
           set (Just m) = throwError $ "Method main already defined in class " ++ m ++ " but
                redefined in class " ++ n
           set Nothing = modify $ \s -> s {mainClass = Just n}
104
106 -- | Adds classes to the state
107 setClasses :: ClassDeclaration -> ClassAnalyzer ()
108 setClasses c@(GCDecl n \_ \_ ) = modify \ \s -> s {classes = (n, c) : classes s}
109
110 -- \mid Add subclasses to the state
111 setSubClasses :: ClassDeclaration -> ClassAnalyzer ()
112 setSubClasses (GCDecl n b \_ ) = modify (\s -> s { subClasses = (n, []) : subClasses s }) >>
       addSubClass n b
113
114 -- | Adds a subclass to the list of subclasses
115 addSubClass :: TypeName -> Maybe TypeName -> ClassAnalyzer ()
116 addSubClass _ Nothing = return ()
117 addSubClass n (Just b) = gets subClasses >>= \sc ->
       case lookup b sc of
118
           Nothing \rightarrow modify \ \s \rightarrow s { subClasses = (b, [n]) : sc }
119
            121
122 -- | Sets super classes in the state
123 setSuperClasses :: ClassDeclaration -> ClassAnalyzer ()
```

```
124 setSuperClasses (GCDecl n _ _ _) = gets subClasses >>= \sc ->  
125 modify \ \s -> s { superClasses = (n, map fst \ filter (\( (_, sub) -> n 'elem' sub) sc) :
            superClasses s }
127 -- | Returns the nearest 2^n as size for given class
128 getClassSize :: ClassDeclaration -> ClassAnalyzer Size
129 getClassSize (GCDecl _ Nothing fs _) =
       return $ 2 ^ (ceiling :: Double -> Integer) (logBase 2 (2 + genericLength fs))
130
131 getClassSize (GCDecl _ (Just b) fs _) =
       getClass b >>= getClassSize >>= \sz ->
132
            return $ 2 ^ (ceiling :: Double -> Integer) (logBase 2 (fromIntegral $ sz +
133
                 genericLength fs))
134
135 -- | Set class size in state
136 setClassSize :: ClassDeclaration -> ClassAnalyzer ()
137 setClassSize c@(GCDecl n \_ \_) =
     getClassSize c \gg sz \rightarrow modify <math>s \sim s classSize = (n, sz) : classSize s
139
140 -- \mid Returns class methods of a passed class
141 resolveClassMethods :: ClassDeclaration -> ClassAnalyzer [MethodDeclaration]
142 resolveClassMethods (GCDecl \_ Nothing \_ ms) = return ms
143 resolveClassMethods (GCDecl n (Just b) _ ms) = getClass b >>= resolveClassMethods >>= combine
       where checkSignature (GMDecl m ps \_, GMDecl m' ps' \_) = when (m == m' && ps /= ps') ( throwError $ "Method " ++ m ++ " in class " ++ n ++ " has invalid method signature")
              compareName (GMDecl m \_ \_) (GMDecl m' \_ \_) = m == m'
              combine ms' = mapM_ checkSignature ((,) <$> ms <*> ms') >> return (unionBy)
146
                   compareName ms ms')
148 -- | Adds the methods of a class in the Class Analyzer State
149 setClassMethods :: ClassDeclaration -> ClassAnalyzer ()
150 setClassMethods c@(GCDecl n \_ \_ ) = resolveClassMethods c >>= \cm ->
        modify \ \s -> s { classMethods = (n, cm) : classMethods s }
151
153 -- | Class Analyzes a program
154 caProgram :: Program -> ClassAnalyzer Program
155 caProgram (GProg p) = do
       mapM_ setClasses p
156
157
        {\tt mapM}\_ setSubClasses p
158
        mapM_ setSuperClasses p
       mapM_ setClassSize p
159
        {\tt mapM}\_ setClassMethods p
160
       mapM_ checkDuplicateClasses p
161
162
        mapM_ checkDuplicateFields p
        mapM_ checkDuplicateMethods p
163
        \boldsymbol{mapM}\_ \text{ checkBaseClass p}
164
165
       mapM_ checkCyclicInheritance p
166
        mapM_ setMainClass p
        mc <- gets mainClass</pre>
167
168
        when (isNothing mc) (throwError "No main method defined")
        return $ GProg rootClasses
169
170
        where
171
            rootClasses = filter noBase p
            noBase (GCDecl _ Nothing _ _) = True
172
173
            noBase _ = False
175 -- | Performs Class Analysis on the program
176 classAnalysis :: Program -> Except String (Program, CAState)
177 classAnalysis p = runStateT (runCA $ caProgram p) initialState
178
179 -- | Pretty prints the Class Analyzer State
180 printCAState :: (Program, CAState) -> IO ()
181 printCAState (\_, s) = pPrint s
```

### ScopeAnalyzer.hs

```
1 {-# LANGUAGE GeneralizedNewtypeDeriving, FlexibleContexts #-}
3 module ScopeAnalyzer
    ( scopeAnalysis
    , printSAState
    , SAState(..)
    ) where
9 import Data.Maybe
10 import Data.List
11 import Data. Typeable
13 import Control.Monad.State
14 import Control.Monad.Except
16 import Debug.Trace (trace, traceShow)
17
18 import Text.Pretty.Simple (pPrint)
19
20 import AST
21 import ClassAnalyzer
23 data SAState =
      SAState {
           symbolIndex :: SIdentifier,
25
26
           symbolTable :: SymbolTable,
27
           scopeStack :: [Scope],
           virtualTables :: [(TypeName, [SIdentifier])],
28
           caState :: CAState,
29
30
           mainMethod :: SIdentifier
31
       } deriving (Show, Eq)
33 newtype ScopeAnalyzer a = ScopeAnalyzer { runSA :: StateT SAState (Except String) a }
       deriving (Functor, Applicative, Monad, MonadState SAState, MonadError String)
36 initialState :: CAState -> SAState
37 initialState s = SAState { symbolIndex = 0, symbolTable = [], scopeStack = [], virtualTables =
        [], caState = s, mainMethod = 0 }
39 -- | Add an empty scope to the scope stack
40 enterScope :: ScopeAnalyzer ()
41 enterScope = modify $ \s -> s { scopeStack = [] : scopeStack s }
43 -- | Leaves the current scope by removing it from the scope stack
44 leaveScope :: ScopeAnalyzer ()
45 leaveScope = modify $ \s -> s { scopeStack = drop 1 $ scopeStack s }
47 -- | Returns the top scope at the scope stack
48 topScope :: ScopeAnalyzer Scope
49 topScope = gets scopeStack >>= \ss ->
      case ss of
50
          (s:_) -> return s
51
52
           [] -> throwError "ICE: Empty scope stack"
54 \ -- \ | \ \mbox{Add} a symbol to the current scope
55 addToScope :: (Identifier, SIdentifier) -> ScopeAnalyzer ()
56 addToScope b =
57
      do ts <- topScope</pre>
         modify $ \s -> s { scopeStack = (b : ts) : drop 1 (scopeStack s) }
58
59
60 -- | Inserts an identifier and symbol pair into the symbol table and current scope
61 saInsert :: Symbol -> Identifier -> ScopeAnalyzer SIdentifier
62 saInsert sym n =
```

```
do ts <- topScope
           when (isJust $ lookup n ts) (throwError $ "Redeclaration of symbol: " ++ n)
 64
 65
           i <- gets symbolIndex
           modify  \s -> s { symbolTable = (i, sym) : symbolTable s, <math>symbolIndex = 1 + i  }
           addToScope (n, i)
 67
 68
           return i
 69
70 \ \text{--}\ |\ \text{Looks up} an identifier in the scope
 71 saLookup :: Identifier -> ScopeAnalyzer SIdentifier
 72 saLookup n = gets scopeStack >>= \ss ->
        case listToMaybe $$ mapMaybe (lookup n) ss of
 73
            Nothing -> throwError $ "Undeclared symbol: " ++ n
            Just i -> return i
 75
 76
 77 -- | Scope Analyses Expressions
 78 saExpression :: Expression -> ScopeAnalyzer SExpression
 79 saExpression (Constant v) = pure $ Constant v
 80 saExpression (Variable n) = Variable <$> saLookup n
 81 saExpression Nil = pure Nil
 82 saExpression (ArrayElement (n, e)) =
       do n' <- saLookup n</pre>
83
           e' <- saExpression e
 84
 85
          return $ ArrayElement (n', e')
 86 saExpression (Binary binop e1 e2) =
       Binary binop
       <$> saExpression e1
 88
        <*> saExpression e2
 89
 91 -- | Scope Analyses Statements
 92 saStatement :: Statement -> ScopeAnalyzer SStatement
 93 saStatement s =
94
       case s of
95
            (Assign n modop e) ->
96
                when (elem n $ var e) (throwError "Irreversible variable assignment")
97
                >> Assign
 98
                <$> saLookup n
                <*> pure modop
99
100
                <*> saExpression e
101
            (AssignArrElem (n, e1) modop e2) ->
102
                when (elem (n, e1) $ varArr e2) (throwError "Irreversible variable assignment")
103
                >> AssignArrElem
104
                <$> saArrayCell n e1
105
                <*> pure modop
106
                <*> saExpression e2
107
108
            (Swap (n1, e1) (n2, e2)) ->
109
110
                Swap
111
                <$> maybeArrayCell n1 e1
                <*> maybeArrayCell n2 e2
112
113
114
            (Conditional e1 s1 s2 e2) ->
115
                Conditional
116
                <$> saExpression el
117
                <*> mapM saStatement s1
                <*> mapM saStatement s2
118
119
                <*> saExpression e2
120
            (Loop e1 s1 s2 e2) ->
121
                Loop
122
                <$> saExpression el
123
124
                <*> mapM saStatement s1
                <*> mapM saStatement s2
126
                <*> saExpression e2
127
            (LocalBlock t n e1 stmt e2) ->
128
```

```
do e1' <- saExpression e1</pre>
129
130
                   enterScope
                   n^{\prime} <- saInsert (LocalVariable t n) n
131
132
                   stmt' <- mapM saStatement stmt
133
                   leaveScope
134
                   e2' <- saExpression e2
                   return $ LocalBlock t n' e1' stmt' e2'
135
136
            (LocalCall m args) ->
                LocalCall
138
139
                <$> saLookup m
                <*> localCall m args
140
141
142
            (LocalUncall m args) ->
143
                LocalUncall
                <$> saLookup m
144
145
                <*> localCall m args
146
147
            (ObjectCall (o, e) m args) ->
                do when (args /= nub args || (o, e) 'elem' args) (throwError $ "Irreversible
                    invocation of method " ++ m)
149
                   >> ObjectCall
150
                   <$> maybeArrayCell o e
                    <*> pure m
151
                   <*> saArgs args
152
153
            (ObjectUncall (o, e) m args) ->
154
                when (args /= nub args || (o, e) 'elem' args) (throwError $ "Irreversible
                    invocation of method " ++ m)
156
                >> ObjectUncall
157
                <$> maybeArrayCell o e
158
                <\star> pure m
159
                <*> saArgs args
160
            (ObjectConstruction tp (n, e)) ->
161
162
                ObjectConstruction
                <$> pure tp
163
164
                <*> maybeArrayCell n e
165
166
            (ObjectDestruction tp (n, e)) ->
167
                ObjectDestruction
                <$> pure tp
168
                <*> maybeArrayCell n e
169
170
            (ObjectBlock tp n stmt) ->
171
172
                do enterScope
                   n' <- saInsert (LocalVariable (ObjectType tp) n) n</pre>
173
                   stmt' <- mapM saStatement stmt
174
175
                    leaveScope
                   return $ ObjectBlock tp n' stmt'
176
177
178
            Skip -> pure Skip
179
180
            (CopyReference tp (n, e1) (m, e2)) ->
                CopyReference
181
                <$> pure tp
182
183
                <*> maybeArrayCell n e1
                <*> maybeArrayCell m e2
184
185
            (UnCopyReference tp (n, e1) (m, e2)) ->
186
                UnCopyReference
187
188
                <$> pure tp
189
                <*> maybeArrayCell n el
190
                <*> maybeArrayCell m e2
191
            (ArrayConstruction (tp, e) n) ->
192
```

```
do n' <- saLookup n
193
                    e' <- saExpression e
194
195
                    return $ ArrayConstruction (tp, e') n'
196
             (ArrayDestruction (tp, e) n) \rightarrow
197
198
                 do n' <- saLookup n</pre>
                    e' <- saExpression e
199
                    return $ ArrayDestruction (tp, e') n'
200
201
        where var (Variable n) = [n]
202
203
               var (Binary \_ e1 e2) = var e1 ++ var e2
204
               var _ = []
205
206
               varArr (ArrayElement (n, e)) = [(n, e)]
207
               varArr _ = []
208
209
               isCF ClassField{} = True
              isCF _ = False
210
211
               rlookup = flip lookup
212
213
               localCall :: MethodName -> [(Identifier, Maybe Expression)] -> ScopeAnalyzer [(
214
                   SIdentifier, Maybe SExpression)]
               localCall m args =
215
                 \textbf{do when } (\texttt{args} \ / = \ \texttt{nub} \ \texttt{args}) \ (\texttt{throwError} \ \$ \ \texttt{"Irreversible invocation of method} \ \texttt{"} \ + + \ \texttt{m}
216
                     )
                    args' <- saArgs args
217
                    st <- gets symbolTable
218
                    when (any isCF \$ mapMaybe (rlookup st . fst) args') (throwError \$ "Irreversible
219
                          invocation of method " ++ m)
220
                    return args'
221
               saArgs :: [(Identifier, Maybe Expression)] -> ScopeAnalyzer [(SIdentifier, Maybe
222
                  SExpression) 1
223
               saArqs arqs =
224
                 do (ns, es) <- pure $ unzip args</pre>
                    ns' <- mapM saLookup ns
225
226
                    es' <- mapM (mapM saExpression) es
227
                    return $ zip ns' es'
228
               maybeArrayCell :: Identifier -> Maybe Expression -> ScopeAnalyzer (SIdentifier,
                   Maybe SExpression)
230
               maybeArrayCell n e =
                 do n' <- saLookup n
                    e' <- mapM saExpression e
232
233
                    return (n', e')
234
               saArrayCell :: Identifier -> Expression -> ScopeAnalyzer (SIdentifier, SExpression)
235
236
               saArrayCell n e =
                 do n' <- saLookup n</pre>
237
                    e' <- saExpression e
238
                    return (n', e')
239
240
241 -- | Set the main method in the Scope Analyzer state
242 setMainMethod :: SIdentifier -> ScopeAnalyzer ()
243 setMainMethod i = modify \ \s -> s { mainMethod = i }
244
245 -- | Scope Analyses Methods
246 saMethod :: (TypeName, MethodDeclaration) -> ScopeAnalyzer (TypeName, SMethodDeclaration)
247 saMethod (t, GMDecl m ps body) =
        \textbf{do} \text{ m'} \text{ <- saLookup m}
248
           when (m == "main") (setMainMethod m')
249
250
           enterScope
           ps' <- mapM insertMethodParameter ps
251
           body' <- mapM saStatement body
252
253
           leaveScope
```

```
return (t, GMDecl m' ps' body')
254
255
       where insertMethodParameter (GDecl tp n) = GDecl tp <$> saInsert (MethodParameter tp n) n
256
257 -- | Returns subclasses for a given type name
258 getSubClasses :: TypeName -> ScopeAnalyzer [ClassDeclaration]
259 getSubClasses n =
       do cs <- gets $ classes . caState</pre>
260
          sc <- gets $ subClasses . caState
261
          case lookup n sc of
262
              Nothing -> throwError $ "ICE: Unknown class " ++ n
263
264
               (Just sc') -> return $ mapMaybe (rlookup cs) sc'
       where rlookup = flip lookup
266
267 -- | Returns method name at given index
268 getMethodName :: SIdentifier -> ScopeAnalyzer (SIdentifier, MethodName)
269 getMethodName i = gets symbolTable >>= \st ->
270
       case lookup i st of
          (Just (Method _ m)) -> return (i, m)
271
272
           _ -> throwError $ "ICE: Invalid method index " ++ show i
274 -- | Prefixes the virtual table
275 prefixVtable :: [(SIdentifier, MethodName)] -> (SIdentifier, MethodName) -> [(SIdentifier,
       MethodName)]
276 prefixVtable [] m' = [m']
277 prefixVtable (m:ms) m' = if comp m m' then m':ms else m : prefixVtable ms m'
278
       where comp (\_, n) (\_, n') = n == n'
279
280 -- | Scope Analyses a passed class
281 saClass :: Offset -> [SIdentifier] -> ClassDeclaration -> ScopeAnalyzer [(TypeName,
       SMethodDeclaration)]
282 saClass offset pids (GCDecl c _ fs ms) =
283
       do enterScope
          mapM_ insertClassField $ zip [offset..] fs
284
285
          m1 <- mapM getMethodName pids
286
          m2 <- mapM insertMethod ms
287
          let m3 = map fst $ foldl prefixVtable m1 m2
              offset' = genericLength fs + offset
288
289
          modify $ \s -> s { virtualTables = (c, m3) : virtualTables s }
290
          sc <- getSubClasses c</pre>
          ms' <- concat <$> mapM (saClass offset' m3) sc
291
          ms'' <- mapM saMethod $ zip (repeat c) ms
292
293
          leaveScope
          return $ ms' ++ ms''
294
       where insertClassField (o, GDecl tp n) = saInsert (ClassField tp n c o) n
295
296
             insertMethod (GMDecl n ps _) = saInsert (Method (map getType ps) n) n >>=
                  getMethodName
             getType (GDecl tp _) = tp
298
299 -- | Analyses Programs
300 saProgram :: Program -> ScopeAnalyzer SProgram
301 saProgram (GProg cs) = concat <$> mapM (saClass 2 []) cs
303 -- | Performs scope analysis on the entire program
304 scopeAnalysis :: (Program, CAState) -> Except String (SProgram, SAState)
305 scopeAnalysis (p, s) = runStateT (runSA $ saProgram p) $ initialState s
306
307 -- | Pretty prints the current Scope Analysis State Monad
308 printSAState :: (Show a, MonadIO m) => (t, a) \rightarrow m ()
309 printSAState (_, s) = pPrint s
```

## TypeChecker.hs

```
1 {-# LANGUAGE GeneralizedNewtypeDeriving #-}
 3 module TypeChecker (typeCheck) where
 5 import Data.List
 6 import Data.Maybe
 8 import Control.Monad.Reader
 9 import Control.Monad.Except
10 import Control.Exception
11
12 import Debug.Trace (trace, traceShow)
14 import AST
15 import ClassAnalyzer
16 import ScopeAnalyzer
18 newtype TypeChecker a = TypeChecker { runTC :: ReaderT SAState (Except String) a }
19
             deriving (Functor, Applicative, Monad, MonadReader SAState, MonadError String)
20
21 getType :: SIdentifier -> TypeChecker DataType
22 getType i = asks symbolTable >>= \st ->
23
             case lookup i st of
                      (Just (LocalVariable t _)) -> return t
                      (Just (ClassField t _ _ _)) -> return t
(Just (MethodParameter t _)) -> return t
25
26
27
                     _ -> throwError $ "ICE: Invalid index " ++ show i
28
29 getParameterTypes :: SIdentifier -> TypeChecker [DataType]
30 getParameterTypes i = asks symbolTable >>= \st ->
31
             {\color{red}\textbf{case lookup}} \ {\color{blue}\textbf{i}} \ {\color{blue}\textbf{st}} \ {\color{blue}\textbf{of}}
                     (Just (Method ps _)) -> return ps
                     _ -> throwError $ "ICE: Invalid index " ++ show i
33
34
35 expectType :: DataType -> DataType -> TypeChecker ()
36 \text{ expectType t1 t2} = \textbf{unless} \text{ (t1 == t2)} \text{ (throwError $ "Expected type: " ++ <math>\textbf{show} \text{ t1 ++ "} \setminus \texttt{nActual} \text{ to the state of the 
              type: " ++ show t2)
38 getClassMethods :: TypeName -> TypeChecker [MethodDeclaration]
39 getClassMethods n = asks (classMethods . caState) >>= \cm ->
             case lookup n cm of
41
                     Nothing -> throwError $ "ICE: Unknown class " ++ n
                      (Just ms) -> return ms
42
44 getDynamicParameterTypes :: TypeName -> MethodName -> TypeChecker [DataType]
45 getDynamicParameterTypes n m = getClassMethods n >>= \mbox{ms} ->
             case find (\(GMDecl m' _ _ ) \rightarrow m == m') ms of
46
                     Nothing -> throwError \ "Class " ++ n ++ " does not support method " ++ m
                      (Just (GMDecl \_ ps \_)) -> return $ map (\((GDecl tp \_) -> tp) ps
48
50 getArrayType :: DataType -> DataType
51 getArrayType tp = case tp of
                                              IntegerArrayType -> IntegerType
                                              ObjectArrayType t -> ObjectType t
53
54
55 checkCall :: [(SIdentifier, Maybe SExpression)] -> [DataType] -> TypeChecker ()
56 checkCall args ps =
57
             when (la /= lp) (throwError err)
58
             >> mapM (mapM tcExpression . snd) args
             >> mapM (getType . fst) args
59
60
             >>= \arrowvert as -> mapM_ checkArgument (zip as ps)
61
             where la = length args
                         lp = length ps
62
```

```
err = "Passed " ++ show la ++ " argument(s) to method expecting " ++ show lp ++ "
                  argument(s)"
65 checkArgument :: (DataType, DataType) -> TypeChecker ()
66 checkArgument (ObjectType ca, ObjectType cp) = asks (superClasses . caState) >>= \sc ->
       unless (ca == cp || maybe False (elem cp) (lookup ca sc)) (throwError $ "Class " ++ ca ++
            " not a subtype of class " ++ cp)
68 checkArgument (ObjectType ca, ObjectArrayType cp) = asks (superClasses . caState) >>= \sc ->
       unless (ca == cp || maybe False (elem cp) (lookup ca sc)) (throwError $ "Class " ++ ca ++
           " not a subtype of class " ++ cp)
70 checkArgument (ObjectArrayType ca, ObjectType cp) = asks (superClasses . caState) >>= \sc ->
       unless (ca == cp || maybe False (elem cp) (lookup ca sc)) (throwError $ "Class " ++ ca ++
            " not a subtype of class " ++ cp)
72 checkArgument (IntegerArrayType, tp) = expectType (getArrayType IntegerArrayType) tp
73 checkArgument (ta, IntegerArrayType) = expectType (getArrayType IntegerArrayType) ta
74 checkArgument (ta, tp) = expectType tp ta
76 tcExpression :: SExpression -> TypeChecker DataType
77 tcExpression (Constant _) = pure IntegerType
78 tcExpression (Variable n) = getType n
79 tcExpression Nil = pure NilType
80 tcExpression (ArrayElement (n, e)) =
81
       do t <- getType n</pre>
82
          expectType ArrayType t
          e' <- tcExpression e
          expectType IntegerType e'
84
85
          return $ getArrayType t
86 tcExpression (Binary binop el e2)
       | binop == Eq || binop == Neq =
87
88
           do t1 <- tcExpression e1</pre>
              t2 <- tcExpression e2
89
90
               expectType t1 t2
              pure IntegerType
91
92
       | otherwise =
93
           do t1 <- tcExpression e1</pre>
94
               t2 <- tcExpression e2
               expectType t1 IntegerType
95
96
               expectType t2 IntegerType
              pure IntegerType
98
99 tcStatement :: SStatement -> TypeChecker ()
100 tcStatement s =
101
       case s of
102
            (Assign n _ e) ->
103
                getType n
104
                >>= expectType IntegerType
105
                >> tcExpression e
               >>= expectType IntegerType
106
107
            (AssignArrElem (n, e1) _ e2) ->
108
109
                getType n
                >>= expectType IntegerArrayType
110
                >> tcExpression el
111
112
                >>= expectType IntegerType
113
                >> tcExpression e2
                >>= expectType IntegerType
114
115
            (Swap (n1, e1) (n2, e2)) ->
116
                do t1 <- getType n1</pre>
117
                   t2 <- getType n2
118
                   if isNothing e1 /= isNothing e2
119
                     then catchError (checkArgument (t1, t2)) (\setminus -> checkArgument (t2, t1))
120
                     else expectType (if isNothing e1 then t1 else getArrayType t1) (if isNothing
                         e2 then t2 else getArrayType t2)
122
            (Conditional e1 s1 s2 e2) ->
123
```

```
tcExpression e1
124
                 >>= expectType IntegerType
125
126
                 >> mapM_{\_} tcStatement s1
127
                 >> mapM_ tcStatement s2
                 >> tcExpression e2
128
129
                 >>= expectType IntegerType
130
             (Loop e1 s1 s2 e2) ->
131
                 tcExpression el
                 >>= expectType IntegerType
133
134
                 >> mapM_ tcStatement s1
135
                 >> mapM_ tcStatement s2
136
                 >> tcExpression e2
137
                 >>= expectType IntegerType
138
             ({\tt ObjectBlock} \ \_ \ \_ \ {\tt stmt}) \ -\!\!\!>
139
140
                 mapM_ tcStatement stmt
141
142
             (LocalBlock t n e1 stmt e2) \rightarrow
143
                 getType n
                 >> tcExpression e1
144
145
                 >>= expectType (if t == IntegerType then IntegerType else NilType)
146
                 >> mapM_ tcStatement stmt
147
                 >> tcExpression e2
                 >>= expectType (if t == IntegerType then IntegerType else NilType)
148
149
150
             (LocalCall m args) ->
                 getParameterTypes m
151
                 >>= checkCall args
152
153
154
             (LocalUncall m args) ->
                 getParameterTypes m
155
156
                 >>= checkCall args
157
             (ObjectCall (o, e) m args) ->
158
159
                 do t <- getType o</pre>
                     e' <- mapM tcExpression e
160
161
                     {\tt case}\ {\tt t}\ {\tt of}
162
                          (ObjectType tn) -> getDynamicParameterTypes tn m >>= checkCall args
                          (ObjectArrayType tn) ->
163
                           {\tt case} \ {\tt e'} \ {\tt of}
164
                               Nothing -> throwError $ "Non-object type " ++ show t ++ " does not
165
                                    support method invocation"
                                 -> getDynamicParameterTypes tn m >>= checkCall args
166
                          _ -> throwError $ "Non-object type " ++ show t ++ " does not support method
167
                               invocation"
168
             (ObjectUncall (o, e) m args) ->
169
170
                 do t <- getType o</pre>
                     e' <- mapM tcExpression e
171
                     {\tt case}\ {\tt t}\ {\tt of}
172
173
                          (ObjectType tn) -> getDynamicParameterTypes tn m >>= checkCall args
                          (ObjectArrayType tn) ->
174
175
                           {\tt case} \ {\tt e'} \ {\tt of}
                               Nothing -> throwError $ "Non-object type " ++ show t ++ " does not
176
                                   support method invocation"
177
                                 -> getDynamicParameterTypes tn m >>= checkCall args
                          _ -> throwError $ "Non-object type " ++ show t ++ " does not support method
178
                               invocation"
179
             Skip -> pure ()
180
181
182
             (ObjectConstruction tp (n, e)) ->
                 do t <- getType n</pre>
183
                     e^{\prime} <- mapM tcExpression e
184
                     {\tt case} \ {\tt e'} \ {\tt of}
185
```

```
Nothing -> expectType t (ObjectType tp)
                               -> checkArgument (ObjectType tp, t)
187
188
189
             (ObjectDestruction tp (n, e)) ->
190
                 do t <- getType n</pre>
                    _ <- mapM tcExpression e
191
                    case t of
192
                     (ObjectType _) -> expectType t (ObjectType tp)
193
                     (ObjectArrayType _) -> checkArgument (ObjectType tp, t)
                     _ -> throwError $ "Expected type: " ++ show (ObjectType tp) ++ " Actual type:
195
                          " ++ show t
197
            -- Allow copying with a copy type
198
            CopyReference \_ (n, e1) (m, e2) ->
199
                 do t1 <- getType n</pre>
                    t2 <- getType m
200
201
                    e1' <- mapM tcExpression e1
                    e2' <- mapM tcExpression e2
202
203
                    when (t1 == IntegerType || t2 == IntegerType) (throwError "Integer types does
                        not support reference copying")
                    if isNothing e1 /= isNothing e2
204
205
                      then catchError (checkArgument (t1, t2)) (\_ -> checkArgument (t2, t1))
206
                      else expectType (if isNothing el then tl else getArrayType tl) (if isNothing
                           e2 then t2 else getArrayType t2)
207
             -- Allow uncopying with two identical copies
208
209
            UnCopyReference _ (n, e1) (m, e2) ->
                 do t1 <- getType n</pre>
                    t2 <- getType m
211
                    e1' <- mapM tcExpression e1
212
                    e2' <- mapM tcExpression e2
213
                    \textbf{when} \ (\texttt{t1} == \texttt{IntegerType} \ | \ | \ \texttt{t2} == \texttt{IntegerType}) \ (\texttt{throwError} \ \texttt{"Integer} \ \texttt{types} \ \texttt{does}
214
                         not support reference copying")
215
                    if isNothing e1 /= isNothing e2
                      then catchError (checkArgument (t1, t2)) (\ -> checkArgument (t2, t1))
216
217
                      else expectType (if isNothing e1 then t1 else getArrayType t1) (if isNothing
                           e2 then t2 else getArrayType t2)
218
219
             (ArrayConstruction (tp, e) n) ->
220
                 do t <- getType n</pre>
                    _ <- tcExpression e
222
223
                    case tp of
                      "int" -> expectType t IntegerArrayType
                            -> expectType t (ObjectArrayType tp)
225
226
             (ArrayDestruction (tp, e) n) ->
                 do t <- getType n</pre>
228
229
                    _ <- tcExpression e
                    case tp of
230
                      "int" -> expectType t IntegerArrayType
231
                             -> checkArgument (ObjectArrayType tp, t)
232
234 getMethodName :: SIdentifier -> TypeChecker Identifier
235 getMethodName i = asks symbolTable >>= \st ->
        \textbf{case lookup} \text{ i st } \textbf{of}
236
            (Just (Method _ n)) -> return n
237
            _ -> throwError $ "ICE: Invalid index " ++ show i
238
239
240 tcMethod :: (TypeName, SMethodDeclaration) -> TypeChecker ()
241 tcMethod (_, GMDecl _ [] body) = mapM_ tcStatement body
242 tcMethod (_, GMDecl i (_:_) body) = getMethodName i >>= n \rightarrow m
        when (n == "main") (throwError "Method main has invalid signature")
        >> mapM_ tcStatement body
244
245
246 tcProgram :: SProgram -> TypeChecker (SProgram, SAState)
```

```
247 tcProgram p = (,) p <$> (mapM_ tcMethod p >> ask)
248
249 typeCheck :: (SProgram, SAState) -> Except String (SProgram, SAState)
250 typeCheck (p, s) = runReaderT (runTC $ tcProgram p) s
```

### CodeGenerator.hs

```
1 {-# LANGUAGE GeneralizedNewtypeDeriving #-}
2 {-# LANGUAGE ScopedTypeVariables
4 module CodeGenerator(
      generatePISA,
      showPISAProgram
7 ) where
9 import Data.List
10
11 import Control.Arrow
12 import Control.Monad.Except
13 import Control.Monad.State
14
15 import Debug.Trace (trace, traceShow)
16
17 import Text.Pretty.Simple (pPrint)
19 import AST
20 import ClassAnalyzer
21 import PISA
22 import ScopeAnalyzer
23
24 {-# ANN module "HLint: ignore Reduce duplication" #-}
26 data CGState =
27
      CGState {
          labelIndex :: SIdentifier,
28
29
           registerIndex :: Integer,
           labelTable :: [(SIdentifier, Label)],
30
          registerStack :: [(SIdentifier, Register)],
31
           saState :: SAState
       } deriving (Show, Eq)
33
34
35 newtype CodeGenerator a = CodeGenerator { runCG :: StateT CGState (Except String) a }
       deriving (Functor, Applicative, Monad, MonadState CGState, MonadError String)
36
37
38
39 initialState :: SAState -> CGState
40 initialState s = CGState { labelIndex = 0, registerIndex = 6, labelTable = [], registerStack =
        [], saState = s }
42 -- | Register containing 0
43 registerZero :: Register
44 registerZero = Reg 0
46 -- | Register containing Stack pointer
47 registerSP :: Register
48 \text{ registerSP} = \text{Reg } 1
49
50 -- | Register RO
51 registerRO :: Register
52 registerRO = Reg 2
54 -- | Register holding 'this'
55 registerThis :: Register
56 registerThis = Reg 3
58 -- | Register containing Free list pointers
59 registerFLPs :: Register
60 \text{ registerFLPs} = \text{Reg } 4
62 -- | Register containing Heap pointer
```

```
63 registerHP :: Register
64 registerHP = Reg 5
65
66 -- | Pushes a new register to the register stack
67 pushRegister :: SIdentifier -> CodeGenerator Register
68 pushRegister i = do ri <- gets registerIndex
                       modify $ \ s -> s \{ registerIndex = 1 + ri, registerStack = (i, Reg ri) : 
                           registerStack s }
70
                       return $ Reg ri
71
72 -- | Pop a register from the register stack
73 popRegister :: CodeGenerator ()
74 popRegister = modify \ \s -> s { registerIndex = (-1) + registerIndex s, registerStack = drop
       1 $ registerStack s }
76 -- | Reserve a tmp register
77 tempRegister :: CodeGenerator Register
78 tempRegister =
79
       do ri <- gets registerIndex</pre>
80
          modify $ \s -> s { registerIndex = 1 + ri }
          return $ Reg ri
81
82
83 -- | Clear reverved tmp register
84 popTempRegister :: CodeGenerator ()
87 -- | Lookup register of given identifier
88 lookupRegister :: SIdentifier -> CodeGenerator Register
89 lookupRegister i = gets registerStack >>= \rs ->
90
       case lookup i rs of
91
           Nothing -> throwError $ "ICE: No register reserved for index " ++ show i
           (Just r) -> return r
92
94 -- | Returns the method name of a valid method identifier
95 getMethodName :: SIdentifier -> CodeGenerator MethodName
96 getMethodName i = gets (symbolTable . saState) >>= \st ->
       case lookup i st of
97
98
           (Just (Method _ n)) -> return n
99
           _ -> throwError $ "ICE: Invalid method index " ++ show i
100
101 -- | Inserts a unique method label in the label table for a given method identifier
102 insertMethodLabel :: SIdentifier -> CodeGenerator ()
103 insertMethodLabel m =
       {f do} n <- getMethodName m
          i <- gets labelIndex
105
          modify $ \s -> s { labelIndex = 1 + i, labelTable = (m, "l_" ++ n ++ "_" ++ show i) :
106
              labelTable s }
107
108 -- \mid Returns the Method label for a method identifier
109 getMethodLabel :: SIdentifier -> CodeGenerator Label
110 getMethodLabel m = gets labelTable >>= \lt ->
       case lookup m lt of
111
112
           (Just 1) -> return 1
113
           Nothing -> insertMethodLabel m >> getMethodLabel m
114
115 -- | Returns a unique label by appending the label index to a passed label type
116 getUniqueLabel :: Label -> CodeGenerator Label
117 getUniqueLabel 1 =
       do i <- gets labelIndex</pre>
118
          modify \ \s -> s { labelIndex = 1 + i }
119
          return $ 1 ++ "_" ++ show i
120
121
122 -- | Returns the address to the variable of a given identifier
123 loadVariableAddress :: SIdentifier -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
       CodeGenerator ())
124 loadVariableAddress n = gets (symbolTable . saState) >>= \st ->
```

```
125
       case lookup n st of
            (Just (ClassField _ .
                                 _ _ o)) -> tempRegister >>= \r -> return (r, [(Nothing, ADD r
126
                registerThis), (Nothing, ADDI r $ Immediate o)], popTempRegister)
            (Just (LocalVariable _ _)) -> lookupRegister n >>= \r -> return (r, [], return ())
                                    __)) -> lookupRegister n >>= \r -> return (r, [], return ())
128
            (Just (MethodParameter
           _ -> throwError $ "ICE: Invalid variable index " ++ show n
129
130
131 -- | Returns the value of a variable of given identifier
132 loadVariableValue :: SIdentifier -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
       CodeGenerator ())
133 loadVariableValue n =
134
       do (ra, la, ua) <- loadVariableAddress n</pre>
135
          rv <- tempRegister
          return (rv, la ++ [(Nothing, EXCH rv ra)] ++ invertInstructions la, popTempRegister >>
136
              ua)
137
138 -- | Returns address an array element
139 loadArrayElementVariableAddress :: SIdentifier -> SExpression -> CodeGenerator (Register, [(
       Maybe Label, MInstruction)], CodeGenerator ())
140 loadArrayElementVariableAddress n e =
       do (ra, la, ua) <- loadVariableAddress n</pre>
141
          (re, le, ue) <- cgExpression e
142
143
          rv <- tempRegister
          rt <- tempRegister
144
          return (rv, la ++ le ++ [ (Nothing, EXCH rt ra), (Nothing, XOR rv rt), (Nothing, EXCH rt
               ra), (Nothing, ADDI rv ArrayElementOffset), (Nothing, ADD rv re)] ++
               invertInstructions (la ++ le), popTempRegister >> popTempRegister >> ue >> ua)
147 \ \text{--}\ |\ \text{Returns} the value of an array element
148 loadArrayElementVariableValue :: SIdentifier -> SExpression -> CodeGenerator (Register, [(
       Maybe Label, MInstruction)], CodeGenerator ())
149 loadArrayElementVariableValue n e =
       do (ra, la, ua) <- loadArrayElementVariableAddress n e</pre>
150
151
          rv <- tempRegister
152
          return (rv, la ++ [(Nothing, EXCH rv ra)] ++ invertInstructions la , popTempRegister >>
153
154 -- Â | Returns pointer to free list at given index
155 loadFreeListAddress :: Register -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
       CodeGenerator ())
156 loadFreeListAddress index = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt registerFLPs
       ), (Nothing, ADD rt index)], popTempRegister)
157
158 -- |Â Returns a copy of the pointer to the head of the free list at the given register
159 loadHeadAtFreeList :: Register -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
       CodeGenerator ())
160 loadHeadAtFreeList rFreeList =
161
       do rv <- tempRegister</pre>
162
          rt <- tempRegister
          let copyAddress = [(Nothing, EXCH rt rFreeList),
163
164
                              (Nothing, XOR rv rt),
165
                              (Nothing, EXCH rt rFreeList)]
166
          return (rv, copyAddress, popTempRegister >> popTempRegister)
167
168 -- | Code generation for binary operators
169 cgBinOp :: BinOp -> Register -> Register -> CodeGenerator (Register, [(Maybe Label,
       MInstruction)], CodeGenerator ())
170 cgBinOp Add r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt r1), (Nothing, ADD
       rt r2)], popTempRegister)
171 cqBinOp Sub r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt r1), (Nothing, SUB
       rt r2)], popTempRegister)
172 cgBinOp Xor r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, XOR rt r1), (Nothing, XOR
       rt r2)], popTempRegister)
173 cgBinOp BitAnd r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, ANDX rt r1 r2)],
       popTempRegister)
```

```
174 cgBinOp BitOr r1 r2 = tempRegister >>= \rt -> return (rt, [(Nothing, ORX rt r1 r2)],
        popTempRegister)
175 cgBinOp Lt r1 r2 =
176
        do rt <- tempRegister</pre>
           rc <- tempRegister</pre>
177
178
           l_top <- getUniqueLabel "cmp_top"</pre>
           l_bot <- getUniqueLabel "cmp_bot"
let cmp = [(Nothing, XOR rt r1),</pre>
179
180
                        (Nothing, SUB rt r2),
                        (Just l_top, BGEZ rt l_bot),
182
183
                        (Nothing, XORI rc $ Immediate 1),
                        (Just l_bot, BGEZ rt l_top)]
184
           return (rc, cmp, popTempRegister >> popTempRegister)
185
186 cgBinOp Gt r1 r2 =
187
        do rt <- tempRegister</pre>
           rc <- tempRegister</pre>
188
189
           l_top <- getUniqueLabel "cmp_top"</pre>
           l_bot <- getUniqueLabel "cmp_bot"</pre>
190
191
           let cmp = [(Nothing, XOR rt r1),
192
                        (Nothing, SUB rt r2),
                        (Just l_top, BLEZ rt l_bot),
193
194
                        (Nothing, XORI rc $ Immediate 1),
195
                        (Just l_bot, BLEZ rt l_top)]
           return (rc, cmp, popTempRegister >> popTempRegister)
196
197 cgBinOp Eq r1 r2 =
        do rt <- tempRegister</pre>
198
           l_top <- getUniqueLabel "cmp_top"</pre>
199
            l_bot <- getUniqueLabel "cmp_bot"</pre>
           let cmp = [(Just l_top, BNE r1 r2 l_bot),
201
202
                        (Nothing, XORI rt $ Immediate 1),
203
                        (Just l_bot, BNE r1 r2 l_top)]
204
           return (rt, cmp, popTempRegister)
205 cgBinOp Neq r1 r2 =
        do rt <- tempRegister</pre>
206
           l_top <- getUniqueLabel "cmp_top"</pre>
207
208
           l_bot <- getUniqueLabel "cmp_bot"</pre>
           let cmp = [(Just l_top, BEQ r1 r2 l_bot),
209
210
                        (Nothing, XORI rt $ Immediate 1),
211
                        (Just l_bot, BEQ r1 r2 l_top)]
           return (rt, cmp, popTempRegister)
212
213 cgBinOp Lte r1 r2 =
214
        do rt <- tempRegister</pre>
215
           rc <- tempRegister</pre>
           l_top <- getUniqueLabel "cmp_top"</pre>
216
           l_bot <- getUniqueLabel "cmp_bot"</pre>
217
218
           let cmp = [(Nothing, XOR rt r1),
                        (Nothing, SUB rt r2),
219
220
                        (Just l_top, BGTZ rt l_bot),
                        (Nothing, XORI rc $ Immediate 1),
221
                        (Just l_bot, BGTZ rt l_top)]
222
223
           return (rc, cmp, popTempRegister >> popTempRegister)
224 cgBinOp Gte r1 r2 =
        do rt <- tempRegister</pre>
225
226
           rc <- tempRegister
227
           l_top <- getUniqueLabel "cmp_top"</pre>
           l_bot <- getUniqueLabel "cmp_bot"</pre>
228
229
           let cmp = [(Nothing, XOR rt r1),
230
                        (Nothing, SUB rt r2),
                        (Just l_top, BLTZ rt l_bot),
231
                        (Nothing, XORI rc $ Immediate 1),
232
                        (Just l_bot, BLTZ rt l_top)]
233
234
           return (rc, cmp, popTempRegister >> popTempRegister)
235 cgBinOp _ _ _ = throwError "ICE: Binary operator not implemented"
237 -- | Code generation for expressions
```

```
238 cgExpression :: SExpression -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
       CodeGenerator ())
239 cgExpression (Constant 0) = return (registerZero, [], return ())
240 cgExpression (Constant n) = tempRegister >>= \rt -> return (rt, [(Nothing, XORI rt $ Immediate
        n)], popTempRegister)
241 cgExpression (Variable i) = loadVariableValue i
242 cgExpression (ArrayElement (n, e)) = loadArrayElementVariableValue n e
243 cgExpression Nil = return (registerZero, [], return ())
244 cgExpression (Binary op el e2) =
       do (r1, l1, u1) <- cgExpression e1</pre>
^{245}
246
           (r2, 12, u2) \leftarrow cgExpression e2
          (ro, lo, uo) <- cgBinOp op r1 r2
247
          return (ro, 11 ++ 12 ++ 1o, uo >> u2 >> u1)
248
249
250 -- | Code generation for binary expressions
251 cgBinaryExpression :: SExpression -> CodeGenerator (Register, [(Maybe Label, MInstruction)],
       CodeGenerator ())
252 cgBinaryExpression e =
253
       do (re, le, ue) <- cgExpression e</pre>
254
          rt <- tempRegister
          l_top <- getUniqueLabel "f_top"</pre>
255
          l_bot <- getUniqueLabel "f_bot"</pre>
256
257
          let flatten = [(Just l_top, BEQ re registerZero l_bot),
                           (Nothing, XORI rt $ Immediate 1),
258
                           (Just l_bot, BEQ re registerZero l_top)]
259
          return (rt, le ++ flatten, popTempRegister >> ue)
260
261
262 -- | Code generation for assignments
263 cgAssign :: SIdentifier -> ModOp -> SExpression -> CodeGenerator [(Maybe Label, MInstruction)]
264 cgAssign n modop e =
       do (rt, lt, ut) <- loadVariableValue n</pre>
265
266
          (re, le, ue) <- cgExpression e
267
          ue >> ut
268
          return $ lt ++ le ++ [(Nothing, cgModOp modop rt re)] ++ invertInstructions (lt ++ le)
269
       where cgModOp ModAdd = ADD
270
             cgModOp ModSub = SUB
              cgModOp\ ModXor = XOR
271
272
273 -- | Code generation for assignments
274 cgAssignArrElem :: (SIdentifier, SExpression) -> ModOp -> SExpression -> CodeGenerator [ (Maybe
        Label, MInstruction)]
275 cgAssignArrElem (n, e1) modop e2 =
276
       do (rt, lt, ut) <- loadArrayElementVariableValue n e1</pre>
          (re, le, ue) <- cgExpression e2
277
          1 <- getUniqueLabel "assArrElem"</pre>
278
279
          ue >> ut
280
          return $ 1t ++ 1e ++ [(Just 1, cgModOp modop rt re)] ++ invertInstructions (1t ++ 1e)
       where cgModOp ModAdd = ADD
281
282
             cgModOp ModSub = SUB
             cgModOp ModXor = XOR
283
284
285 -- | Ensures correct loads for swapping
286 loadForSwap :: (SIdentifier, Maybe SExpression) -> CodeGenerator (Register, [(Maybe Label,
       MInstruction)], CodeGenerator ())
287 loadForSwap (n, x) = gets (symbolTable . saState) >>= \st ->
       case lookup n st of
288
            (Just (ClassField IntegerArrayType _ _ _)) -> case x of
289
                                                             Just x' ->
290
                                                                 loadArravElementVariableValue n x'
                                                              _ -> loadVariableValue n
291
            (Just (ClassField (ObjectArrayType _) _ _ _)) -> case x of
292
                                                              Just x' ->
293
                                                                  loadArrayElementVariableValue n x'
294
                                                               -> loadVariableValue n
            (Just ClassField {}) -> loadVariableValue n
295
            (Just (LocalVariable IntegerType _)) -> loadVariableValue n
296
```

```
(Just (LocalVariable (ObjectType _) _)) -> loadVariableValue n
297
            (Just (LocalVariable (CopyType _) _)) -> loadVariableValue n
298
299
            (Just (LocalVariable IntegerArrayType _)) -> case x of
300
                                                              Just x' ->
                                                                  loadArrayElementVariableValue n x'
                                                              _ -> loadVariableValue n
301
302
            (Just (LocalVariable (ObjectArrayType _) _)) -> case x of
                                                               Just x' ->
303
                                                                   loadArrayElementVariableValue n x'
                                                                -> loadVariableValue n
304
            ( \pmb{\textbf{Just}} (MethodParameter IntegerType _)) -> loadVariableValue n
305
            (Just (MethodParameter (ObjectType _) _)) -> loadVariableValue n
            ( \pmb{\textbf{Just}} (MethodParameter (CopyType _) _)) -> loadVariableValue n
307
308
            (Just (MethodParameter IntegerArrayType _)) -> case x of
309
                                                                Just x' ->
                                                                    loadArrayElementVariableValue n x
                                                                 -> loadVariableValue n
310
311
            (Just (MethodParameter (ObjectArrayType _) _)) -> case x of
312
                                                                   Just x' ->
                                                                       loadArrayElementVariableValue
                                                                       n x'
313
                                                                     -> loadVariableValue n
             _ -> throwError $ "ICE: Invalid variable index " ++ show n
314
316 -- | Code generation for swaps
317 cgSwap :: (SIdentifier, Maybe SExpression) -> (SIdentifier, Maybe SExpression) ->
       CodeGenerator [(Maybe Label, MInstruction)]
318 cgSwap n1 n2 = if n1 == n2 then return [] else
319
       do (r1, l1, u1) <- loadForSwap n1
           (r2, 12, u2) <- loadForSwap n2
320
321
           u2 >> u1
           1 <- getUniqueLabel "swap"</pre>
322
323
           let swap = [(Just 1, XOR r1 r2), (Nothing, XOR r2 r1), (Nothing, XOR r1 r2)]
324
           return $ 11 ++ 12 ++ swap ++ invertInstructions (11 ++ 12)
326 -- | Code generation for conditionals
327 cgConditional :: SExpression -> [SStatement] -> [SStatement] -> SExpression -> CodeGenerator
        [(Maybe Label, MInstruction)]
328 cgConditional el s1 s2 e2 =
        do l_test <- getUniqueLabel "test"</pre>
329
           l_assert_t <- getUniqueLabel "assert_true"</pre>
330
           l_test_f <- getUniqueLabel "test_false"</pre>
331
           l_assert <- getUniqueLabel "assert"</pre>
           rt <- tempRegister
333
334
           (rel, lel, uel) <- cgBinaryExpression el
335
           s1' <- concat <$> mapM cgStatement s1
336
337
           s2' <- concat <$> mapM cgStatement s2
           (re2, le2, ue2) <- cgBinaryExpression e2
338
339
           ue2 >> popTempRegister --rt
           return $ le1 ++ [(Nothing, XOR rt re1)] ++ invertInstructions le1 ++
340
                    [(Just l_test, BEQ rt registerZero l_test_f), (Nothing, XORI rt $ Immediate 1)
341
                        ] ++
342
                        ++ [(Nothing, XORI rt $ Immediate 1), (Just l_assert_t, BRA l_assert), (
                        Just l_test_f, BRA l_test)] ++
                    s2' ++ [(Just l_assert, BNE rt registerZero l_assert_t)] ++
343
                    le2 ++ [(Nothing, XOR rt re2)] ++ invertInstructions le2
344
345
346 -- | Code generation for loops
347 cqLoop :: SExpression -> [SStatement] -> [SStatement] -> SExpression -> CodeGenerator [(Maybe
        Label, MInstruction)]
348 cgLoop e1 s1 s2 e2 =
        do l_entry <- getUniqueLabel "entry"</pre>
349
           l_test <- getUniqueLabel "test"</pre>
350
           l_assert <- getUniqueLabel "assert"</pre>
351
```

```
l_exit <- getUniqueLabel "exit"</pre>
          rt <- tempRegister
353
354
           (re1, le1, ue1) <- cgBinaryExpression e1</pre>
355
356
          s1' <- concat <$> mapM cgStatement s1
           s2' <- concat <$> mapM cgStatement s2
357
           (re2, le2, ue2) <- cgBinaryExpression e2
358
          ue2 >> popTempRegister --rt
359
          return $ [(Nothing, XORI rt $ Immediate 1), (Just l_entry, BEQ rt registerZero l_assert
               ) ] ++
                    le1 ++ [(Nothing, XOR rt re1)] ++ invertInstructions le1 ++
361
                    s1' ++ le2 ++ [(Nothing, XOR rt re2)] ++ invertInstructions le2 ++
362
                    [(Just l_test, BNE rt registerZero l_exit)] ++ s2' ++
363
                    [(Just l_assert, BRA l_entry), (Just l_exit, BRA l_test), (Nothing, XORI rt $
364
                        Immediate 1)]
365
   -- | Code generation for object blocks
367 cgObjectBlock :: TypeName -> SIdentifier -> [SStatement] -> CodeGenerator [(Maybe Label,
       MInstruction)]
368 cgObjectBlock tp n stmt =
369
       do rn <- pushRegister n</pre>
          let push = [(Nothing, XOR rn registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
370
371
          malloc <- cgObjectConstruction tp (n, Nothing)</pre>
           stmt' <- concat <$> mapM cgStatement stmt
372
           free <- cgObjectDestruction tp (n, Nothing)</pre>
374
          popRegister
          return $ push ++ malloc ++ stmt' ++ free ++ invertInstructions push
375
377 -- | Code generation for local blocks
378 cgLocalBlock :: SIdentifier -> SExpression -> [SStatement] -> SExpression -> CodeGenerator [(
       Maybe Label, MInstruction)]
379 cgLocalBlock n e1 stmt e2 =
       do rn <- pushRegister n</pre>
380
381
          (rel, lel, uel) <- cgExpression el
382
          rt1 <- tempRegister
383
          popTempRegister >> ue1
          stmt' <- concat <$> mapM cgStatement stmt
384
385
           (re2, le2, ue2) <- cgExpression e2
386
          rt2 <- tempRegister
387
          popTempRegister >> ue2
          popRegister --rn
           1 <- getUniqueLabel "localBlock"</pre>
389
390
          let create re rt = [(Just 1, XOR rn registerSP),
                                (Nothing, XOR rt re),
392
                                (Nothing, EXCH rt registerSP),
393
                                (Nothing, SUBI registerSP $ Immediate 1)]
394
               load = le1 ++ create re1 rt1 ++ invertInstructions le1
               clear = le2 ++ invertInstructions (create re2 rt2) ++ invertInstructions le2
395
396
           return $ load ++ stmt' ++ clear
397
398 -- | Code generation for calls
399 cgCall :: [(SIdentifier, Maybe SExpression)] -> [(Maybe Label, MInstruction)] -> Register ->
       CodeGenerator [(Maybe Label, MInstruction)]
400 cgCall args jump this =
401
       do (ra, la, ua) <- unzip3 <$> mapM loadAddr args
          sequence_ ua
402
           rs <- gets registerStack
403
           let rr = (registerThis : map snd rs) \\ (this : ra)
404
405
               store = concatMap push $ rr ++ ra ++ [this]
           return $ concat la ++ store ++ jump ++ invertInstructions store ++ invertInstructions (
               concat la)
       where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
407
408
              loadAddr (n, e) =
409
                case e of
                    Nothing -> loadVariableAddress n
410
                    Just e' -> loadArrayElementVariableAddress n e'
411
```

```
413 -- | Code generation for local calling
414 cgLocalCall :: SIdentifier -> [(SIdentifier, Maybe SExpression)]-> CodeGenerator [(Maybe Label
       , MInstruction)]
415 cgLocalCall m args = getMethodLabel m >>= \lowerright -> cgCall args [(Nothing, BRA l_m)]
       registerThis
417 -- | Code generation for local uncalling
418 cgLocalUncall :: SIdentifier -> [(SIdentifier, Maybe SExpression)] -> CodeGenerator [(Maybe
       Label, MInstruction)
registerThis
420
421 -- | Returns the type associated with a given identifier
422 getType :: SIdentifier -> CodeGenerator TypeName
423 getType i = gets (symbolTable . saState) >>= \st ->
424
       case lookup i st of
           (Just (LocalVariable (ObjectType tp) _)) -> return tp
425
426
            ({\it Just} (ClassField (ObjectType tp) _ _ _)) -> {\it return} tp
427
            (Just (MethodParameter (ObjectType tp) _)) -> return tp
            (Just (LocalVariable (ObjectArrayType tp) _)) -> return tp
428
            (Just (ClassField (ObjectArrayType tp) _ _ _)) -> return tp
429
430
           (Just (MethodParameter (ObjectArrayType tp) _)) -> return tp
           _ -> throwError $ "ICE: Invalid object variable index " ++ show i
431
432
433 -- | Load the return offset for methods
434 loadMethodAddress :: (SIdentifier, Register) -> MethodName -> CodeGenerator (Register, [(Maybe
        Label, MInstruction)])
435 loadMethodAddress (o, ro) m =
436
       do rv <- tempRegister</pre>
          rt <- tempRegister
437
438
          rtgt <- tempRegister
          offsetMacro <- OffsetMacro <$> getType o <*> pure m
439
440
          1 <- getUniqueLabel "loadMetAdd"</pre>
441
          let load = [(Just 1, EXCH rv ro),
442
                       (Nothing, ADDI rv offsetMacro),
                       (Nothing, EXCH rt rv),
443
444
                       (Nothing, XOR rtgt rt),
445
                       (Nothing, EXCH rt rv),
                       (Nothing, SUBI rv offsetMacro),
446
                       (Nothing, EXCH rv ro)]
447
          return (rtgt, load)
448
449
450 -- | Load address or value needed for calls
451 loadForCall :: (SIdentifier, Maybe SExpression) -> CodeGenerator (Register, [(Maybe Label,
       MInstruction)], CodeGenerator ())
452 loadForCall (n, e) = gets (symbolTable . saState) >>= \st ->
       \textbf{case lookup} \ n \ \texttt{st} \ \textbf{of}
453
454
           (Just (ClassField (ObjectArrayType _) _ _ _)) ->
455
               case e of
                   Just x' -> loadArrayElementVariableValue n x'
456
                    _ -> throwError $ "ICE: Invalid variable index " ++ show n
457
            (Just ClassField {}) -> loadVariableValue n
458
459
            (Just (LocalVariable (ObjectType _) _)) -> loadVariableValue n
460
            (Just (LocalVariable (CopyType _) _)) -> loadVariableValue n
            (Just (LocalVariable (ObjectArrayType _) _)) ->
461
462
               case e of
                   Just x' -> loadArrayElementVariableValue n x'
463
                     -> throwError $ "ICE: Invalid variable index " ++ show n
464
            (Just _) -> loadVariableValue n
465
           _ -> throwError $ "ICE: Invalid variable index " ++ show n
466
467
468 -- | Code generation for object calls
469 cgObjectCall :: (SIdentifier, Maybe SExpression) -> MethodName -> [(SIdentifier, Maybe
       SExpression)] -> CodeGenerator [(Maybe Label, MInstruction)]
470 cgObjectCall (o, e) m args =
```

```
do (ro, lo, uo) <- loadForCall (o, e)</pre>
471
           rt <- tempRegister
472
473
           (rtgt, loadAddress) <- loadMethodAddress (o, rt) m</pre>
474
           l_jmp <- getUniqueLabel "l_jmp"</pre>
           let jp = [(Nothing, SUBI rtgt $ AddressMacro l_jmp),
475
                      (Just l_jmp, SWAPBR rtgt),
476
                      (Nothing, NEG rtgt),
477
478
                      (Nothing, ADDI rtgt $ AddressMacro l_jmp)]
           call <- cgCall args jp rt
479
           \verb|popTempRegister| >> \verb|popTempRegister| -- rv, rt \& rtgt from loadMethod| \\
480
               Addr
481
           popTempRegister >> uo
           let load = lo ++ [(Nothing, XOR rt ro)] ++ loadAddress ++ invertInstructions lo
482
           return $ load ++ call ++ invertInstructions load
483
484
485 -- | Code generation for object uncalls
486 cgObjectUncall :: (SIdentifier, Maybe SExpression) -> MethodName -> [(SIdentifier, Maybe
        SExpression)] -> CodeGenerator [(Maybe Label, MInstruction)]
487 cgObjectUncall (o, e) m args =
488
        do (ro, lo, uo) <- loadForCall (o, e)</pre>
           rt <- tempRegister
489
490
           (rtgt, loadAddress) <- loadMethodAddress (o, rt) m</pre>
491
           l_jmp <- getUniqueLabel "l_jmp"</pre>
           l_rjmp_top <- getUniqueLabel "l_rjmp_top"</pre>
492
           l_rjmp_bot <- getUniqueLabel "l_rjmp_bot"</pre>
493
           let jp = [(Nothing, SUBI rtgt $ AddressMacro l_jmp),
494
495
                      (Just l_rjmp_top, RBRA l_rjmp_bot),
                      (Just l_jmp, SWAPBR rtgt),
                      (Nothing, NEG rtgt),
497
498
                      (Just l_rjmp_bot, BRA l_rjmp_top),
                     (Nothing, ADDI rtgt $ AddressMacro l_jmp)]
499
500
           call <- cgCall args jp rt
           \verb|popTempRegister| >> \verb|popTempRegister| -- rv, rt \& rtgt from loadMethod|
501
               Addr
502
           popTempRegister >> uo
503
           let load = lo ++ [(Nothing, XOR rt ro)] ++ loadAddress ++ invertInstructions lo
           return $ load ++ call ++ invertInstructions load
504
506 -- | Code generation for object construction
507 cgObjectConstruction :: TypeName -> (SIdentifier, Maybe SExpression) -> CodeGenerator [(Maybe
        Label, MInstruction)]
508 cgObjectConstruction tp (n, e) =
509
       do (rv, lv, uv) <- case e of</pre>
                              Nothing -> loadVariableAddress n
                              Just e' -> loadArrayElementVariableAddress n e'
511
512
           rp <- tempRegister
           rt <- tempRegister
513
           popTempRegister >> popTempRegister
514
           1 <- getUniqueLabel "obj_con"</pre>
515
           rs <- gets registerStack
516
517
           let rr = (registerThis : map snd rs) \\ [rp, rt]
518
               store = concatMap push rr
               malloc = [(Just 1, ADDI rt $ SizeMacro tp)] ++ push rt ++ push rp
519
520
               lb = 1 ++ "_bot"
521
               setVtable = [(Nothing, XORI rt $ AddressMacro $ "l_" ++ tp ++ "_vt"),
                             (Nothing, EXCH rt rp),
522
                             (Nothing, ADDI rp ReferenceCounterIndex),
523
                             (Nothing, XORI rt $ Immediate 1),
524
525
                             (Nothing, EXCH rt rp),
                             (Just lb, SUBI rp ReferenceCounterIndex),
526
                             (Nothing, EXCH rp rv)]
527
528
           return $ store ++ malloc ++ [(Nothing, BRA "l_malloc")] ++ invertInstructions malloc ++
                invertInstructions store ++ lv ++ setVtable ++ invertInstructions lv
        where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
530
531
```

```
532 -- | Code generation for object destruction
533 cgObjectDestruction :: TypeName -> (SIdentifier, Maybe SExpression) -> CodeGenerator [(Maybe
        Label, MInstruction)]
534 cgObjectDestruction tp (n, e) =
535
       do (rp, la, ua) <- case e of
                              Nothing -> loadVariableValue n
536
537
                              Just e' -> loadArrayElementVariableValue n e'
           rt <- tempRegister
538
           l <- getUniqueLabel "obj_des"</pre>
           popTempRegister >> ua
540
541
           rs <- gets registerStack
542
           let removeVtable = [(Just lt, EXCH rt rp),
                                 (Nothing, XORI rt $ AddressMacro $ "l_" ++ tp ++ "_vt"),
543
544
                                 (Nothing, ADDI rp ReferenceCounterIndex),
545
                                 (Nothing, EXCH rt rp),
                                 (Nothing, XORI rt $ Immediate 1),
546
547
                                 (Nothing, SUBI rp ReferenceCounterIndex)]
               rr = (registerThis : map snd rs) \\ [rp, rt]
548
549
               store = concatMap push rr
               free = [(Just 1, ADDI rt $ SizeMacro tp)] ++ push rt ++ push rp
550
               lt = 1 ++ "_top"
551
           return $ la ++ removeVtable ++ store ++ free ++ [(Nothing, RBRA "l_malloc")] ++
552
               invertInstructions (la ++ store ++ free)
        where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
553
555 -- | Code generation for reference construction
556 cgCopyReference :: (SIdentifier, Maybe SExpression) -> (SIdentifier, Maybe SExpression) ->
        CodeGenerator [(Maybe Label, MInstruction)]
557 cgCopyReference (n, e1) (m, e2) =
558
        do (rcp, lp, up) <- case e2 of
559
                               Nothing -> loadVariableValue m
                               Just e2' -> loadArrayElementVariableValue m e2'
560
561
           (rp, la, ua) <- case e1 of
562
                              Nothing -> loadVariableValue n
                              Just el' -> loadArrayElementVariableValue n el'
563
564
           rt <- tempRegister
           up >> ua >> popTempRegister
565
566
           1 <- getUniqueLabel "copy"</pre>
           let reference = [(Just 1, XOR rcp rp),
567
568
                             (Nothing, ADDI rp ReferenceCounterIndex),
                             (Nothing, EXCH rt rp),
569
                             (Nothing, ADDI rt $ Immediate 1),
570
571
                             (Nothing, EXCH rt rp),
                             (Nothing, SUBI rp ReferenceCounterIndex)]
572
           return $ lp ++ la ++ reference ++ invertInstructions (lp ++ la)
573
574
575 -- | Code generation for reference destruction
576 cgUnCopyReference :: (SIdentifier, Maybe SExpression) -> (SIdentifier, Maybe SExpression) ->
        CodeGenerator [(Maybe Label, MInstruction)]
577 \text{ cgUnCopyReference (n, e1) (m, e2)} =
578
        do (rcp, la1, ua1) <- case e2 of</pre>
                                 Nothing -> loadVariableValue m
579
                                 Just e2' -> loadArrayElementVariableValue m e2'
580
581
           (rp, la2, ua2) <- case e1 of
                                Nothing -> loadVariableValue n

Just e1' -> loadArrayElementVariableValue n e1'
582
583
           rt <- tempRegister
584
           1 <- getUniqueLabel "uncopy"</pre>
585
586
           ua1 >> ua2 >> popTempRegister
           let reference = [(Just 1, XOR rcp rp),
587
                             (Nothing, ADDI rp ReferenceCounterIndex),
588
589
                             (Nothing, EXCH rt rp),
590
                             (Nothing, SUBI rt $ Immediate 1),
                             (Nothing, EXCH rt rp),
591
                             (Nothing, SUBI rp ReferenceCounterIndex)]
592
593
              removeRegister (m, rcp)
```

```
return $ la1 ++ la2 ++ reference ++ invertInstructions (la1 ++ la2)
594
595
596 -- | Code generation for array construction
597 cgArrayConstruction :: SExpression -> SIdentifier -> CodeGenerator [(Maybe Label, MInstruction
       ) ]
598 cgArrayConstruction e n =
       \textbf{do} \text{ (ra, la, ua) } \leftarrow \text{loadVariableAddress n}
599
           (re, le, ue) <- cgExpression e
600
           rp <- tempRegister</pre>
          rt <- tempRegister
602
603
           popTempRegister >> popTempRegister
          l <- getUniqueLabel "arr_con"</pre>
604
          rs <- gets registerStack
605
606
           let rr = (registerThis : map snd rs) \\ [rp, rt]
607
               store = le ++ [(Just 1, ADDI rt ArrayElementOffset), (Nothing, ADD rt re)] ++
                   invertInstructions le ++ concatMap push rr
608
               malloc = push rt ++ push rp
               lb = 1 ++ "_bot"
609
               initArray = la ++ le ++
610
                            [(Nothing, XOR rt re),
611
                             (Nothing, EXCH rt rp),
612
                             (Nothing, ADDI rp ReferenceCounterIndex),
613
614
                             (Nothing, XORI rt $ Immediate 1),
                             (Nothing, EXCH rt rp),
615
                             (Nothing, SUBI rp ReferenceCounterIndex),
616
                             (Just lb, EXCH rp ra)] ++
617
618
                            invertInstructions (la ++ le)
           ue >> ua
           return $ store ++ malloc ++ [(Nothing, BRA "l_malloc")] ++ invertInstructions (store ++
620
                malloc) ++ initArray
621
        where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
622
623 -- | Code generation for array destruction
624 cgArrayDestruction :: SExpression -> SIdentifier -> CodeGenerator [(Maybe Label, MInstruction)
625 cgArrayDestruction e n =
       do (rp, lp, up) <- loadVariableValue n</pre>
626
627
           (re, le, ue) <- cgExpression e
628
           rt <- tempRegister
          1 <- getUniqueLabel "obj_des"</pre>
629
          popTempRegister >> ue >> up
630
631
           rs <- gets registerStack
           let removeArray = [(Just lt, EXCH rt rp),
632
                               (Nothing, XOR rt re),
633
634
                               (Nothing, ADDI rp ReferenceCounterIndex),
635
                               (Nothing, EXCH rt rp),
636
                               (Nothing, XORI rt $ Immediate 1),
                               (Nothing, SUBI rp ReferenceCounterIndex)]
637
638
               rr = (registerThis : map snd rs) \\ [rp, rt]
               store = concatMap push rr
639
640
               free = [(Just 1, ADDI rt ArrayElementOffset), (Nothing, ADD rt re)] ++ push rt ++
                   push rp
               lt = 1 ++ " top"
641
           return $ lp ++ le ++ removeArray ++ store ++ free ++ [(Nothing, RBRA "l_malloc")] ++
642
               invertInstructions (lp ++ le ++ store ++ free)
       where push r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $ Immediate 1)]
643
645 -- | Code generation for statements
646 cgStatement :: SStatement -> CodeGenerator [(Maybe Label, MInstruction)]
647 cgStatement (Assign n modop e) = cgAssign n modop e
648 cgStatement (AssignArrElem (n, e1) modop e2) = cgAssignArrElem (n, e1) modop e2
649 cgStatement (Swap (n1, e1) (n2, e2)) = cgSwap (n1, e1) (n2, e2)
650 cgStatement (Conditional el sl s2 e2) = cgConditional el sl s2 e2
651 cgStatement (Loop e1 s1 s2 e2) = cgLoop e1 s1 s2 e2
652 cgStatement (ObjectBlock tp n stmt) = cgObjectBlock tp n stmt
653 cgStatement (LocalBlock _ n e1 stmt e2) = cgLocalBlock n e1 stmt e2
```

```
654 cgStatement (LocalCall m args) = cgLocalCall m args
655 cgStatement (LocalUncall m args) = cgLocalUncall m args
656 cgStatement (ObjectCall o m args) = cgObjectCall o m args
657 cgStatement (ObjectUncall o m args) = cgObjectUncall o m args
658 cgStatement (ObjectConstruction tp n) = cgObjectConstruction tp n
659 cgStatement (ObjectDestruction tp n) = cgObjectDestruction tp n
660 cgStatement Skip = return []
661 cgStatement (CopyReference _ n m) = cgCopyReference n m
662 cgStatement (UnCopyReference _ n m) = cgUnCopyReference n m
663 cgStatement (ArrayConstruction (_, e) n) = cgArrayConstruction e n
664 cgStatement (ArrayDestruction (_{,} e) n) = cgArrayDestruction e n
666 -- | Code generation for methods
667 cgMethod :: (TypeName, SMethodDeclaration) -> CodeGenerator [(Maybe Label, MInstruction)]
668 cgMethod (_, GMDecl m ps body) =
        do 1 <- getMethodLabel m</pre>
669
670
           rs <- addParameters
           body' <- concat <$> mapM cgStatement body
671
672
           clearParameters
673
           let lt = l ++ " top"
               lb = 1 ++ " bot"
674
675
               mp = [(Just lt, BRA lb),
676
                      (Nothing, ADDI registerSP $ Immediate 1),
                      (Nothing, EXCH registerRO registerSP)]
677
                      ++ concatMap pushParameter rs ++
678
                     [(Nothing, EXCH registerThis registerSP),
679
680
                      (Nothing, SUBI registerSP $ Immediate 1),
                      (Just 1, SWAPBR registerRO),
                      (Nothing, NEG registerRO),
682
683
                      (Nothing, ADDI registerSP $ Immediate 1),
684
                      (Nothing, EXCH registerThis registerSP)]
685
                      ++ invertInstructions (concatMap pushParameter rs) ++
                     [(Nothing, EXCH registerRO registerSP),
686
687
                      (Nothing, SUBI registerSP $ Immediate 1)]
           return $ mp ++ body' ++ [(Just lb, BRA lt)]
688
689
        where addParameters = mapM (pushRegister . (\(GDecl _ p) -> p)) ps
              clearParameters = replicateM_ (length ps) popRegister
690
691
              pushParameter r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $
                   Immediate 1)]
692
693 cgMalloc1 :: CodeGenerator [(Maybe Label, MInstruction)]
694 cqMalloc1 =
        do -- Temp registers needed for malloc
695
           r_p <- tempRegister -- Pointer to new obj
696
           r_object_size <- tempRegister -- Object size</pre>
697
698
           r_counter <- tempRegister -- Free list index</pre>
           r_csize <- tempRegister -- Current cell size</pre>
700
           rt <- tempRegister
701
           rt2 <- tempRegister
           r_tmp <- tempRegister
702
703
704
           -- Expressions and sub routines
           (r_el_outer, l_el_outer, u_el_outer) <- cgBinOp Lt r_csize r_object_size</pre>
705
706
           (r_e2_outer, l_e2_outer, u_e2_outer) <- cgBinOp Lt r_csize r_object_size
707
           (r_fl, l_fl, u_fl) <- loadFreeListAddress r_counter</pre>
           (r_block, l_block, u_block) <- loadHeadAtFreeList r_fl</pre>
708
           (r_el_inner, l_el_inner, u_el_inner) <- cgBinOp Neq r_block registerZero
709
           (r_e2_i1, 1_e2_i1, u_e2_i1) <- cgBinOp Neq r_p r_tmp (r_e2_i2, 1_e2_i2, u_e2_i2) <- cgBinOp Eq r_tmp registerZero
710
711
           (r_e2_i3, l_e2_i3, u_e2_i3) \leftarrow cgBinOp BitOr r_e2_i1 r_e2_i2
712
713
714
           let tmpRegisterList = [rt, rt2, r_tmp, r_e1_outer, r_e2_outer, r_f1, r_block,
               r_el_inner, r_e2_i1, r_e2_i2, r_e2_i3]
715
716
           -- Update state after evaluating expressions and subroutines
           u_e2_i3 >> u_e2_i2 >> u_e2_i1 >> u_e1_inner
717
```

```
u_block >> u_fl >> u_e2_outer >> u_e1_outer
           popTempRegister >> popTempRegister >> popTempRegister >> popTempRegister >>
719
               popTempRegister >> popTempRegister >> popTempRegister
           let l_o_test = "l_o_test"
721
               l_o_assert_t = "l_o_assert_true"
722
               l_o_test_f = "l_o_test_false"
723
               l_o_assert = "l_o_assert"
724
               l_i_test = "l_i_test"
725
               l_i_assert_t = "l_i_assert_true"
726
               l_i_test_f = "l_i_test_false"
727
               l_i_assert = "l_i_assert"
               l_m_top = "l_malloc1_top"
l_m_bot = "l_malloc1_bot"
729
730
               l_m_entry = "l_malloc1"
731
               malloc = [(Just l_m_top, BRA l_m_bot),
732
733
                          (Nothing, ADDI registerSP $ Immediate 1),
                          (Nothing, EXCH registerRO registerSP)]
                                                                       -- Pop return offset from
734
                              stack
                         ++ invertInstructions l_fl ++
                                                                       -- Malloc1 entry/exit point
                        [(\textbf{Just} l_m_entry, SWAPBR registerRO),
736
737
                          (Nothing, NEG registerRO),
                                                                       -- Restore return offset
738
                          (Nothing, EXCH registerRO registerSP),
                                                                       -- Push return offset to stack
                         (Nothing, SUBI registerSP $ Immediate 1)]
739
                        ++ l_fl
740
                        ++ l_block
741
                                                                       -- Set r_e1 -> c_size <
742
                        ++ l_e1_outer
                            obj_size
                        ++ [(Nothing, XOR rt r_e1_outer)]
                                                                       -- r_t = r_e1_o
743
744
                        ++ invertInstructions l_e1_outer ++
                                                                       -- Clear r_e1_o
745
                        [(Just l_o_test, BEQ rt registerZero l_o_test_f),
                         (Nothing, XORI rt $ Immediate 1),
                                                                      -- S1_outer start
746
747
                          (Nothing, ADDI r_counter $ Immediate 1)]
                                                                      -- counter++
748
                         ++ invertInstructions l_block ++
                                                                       -- call double(csize)
749
                         [(Nothing, RL r_csize $ Immediate 1)]
750
                        ++ concatMap pushRegisterToStack tmpRegisterList
751
752
                        [(Nothing, BRA l_m_entry)]
                                                                       -- call malloc1()
                        ++ invertInstructions(concatMap pushRegisterToStack tmpRegisterList)
753
754
                        [(Nothing, RR r_csize $ Immediate 1),
                                                                       -- uncall double(csize)
755
                          (Nothing, SUBI r_counter $ Immediate 1),
756
                                                                       -- counter++
                                                                       -- S1_outer end
757
                          (Nothing, XORI rt $ Immediate 1),
                          (Just l_o_assert_t, BRA l_o_assert),
                         (Just l_o_test_f, BRA l_o_test)]
759
760
                         ++ l_e1_inner ++
                                                                       -- Set r_e1_i -> r_block != 0
                              (S2_OUTER)
                        [(Nothing, XOR rt2 r_e1_inner)]
                                                                       -- Set rt2 -> r_e1_i
761
762
                          ++ invertInstructions l_e1_inner ++
                                                                       -- Clear r_e1_i
                         [(Just l_i_test, BEQ rt2 registerZero l_i_test_f),
763
764
                          (Nothing, XORI rt2 $ Immediate 1),
                                                                      -- S1_inner start
                          (Nothing, ADD r_p r_block),
                                                                      -- Set address of p to said
765
                             block
766
                          (Nothing, SUB r_block r_p),
                                                                       -- Clear r_block
767
                          (Nothing, EXCH r_tmp r_p),
                                                                       -- Load address of next block
                          (Nothing, EXCH r_tmp r_fl),
                                                                       -- Set address of next block
768
                              as head of current free list
                          (Nothing, XOR r_tmp r_p),
                                                                       -- Clear address of next block
769
                                                                       -- S1_inner end
                          (Nothing, XORI rt2 $ Immediate 1),
770
                          (Just l_i_assert_t, BRA l_i_assert),
771
                          (Just l_i_test_f, BRA l_i_test),
772
773
                          (Nothing, ADDI r_counter $ Immediate 1),
                                                                      -- S2_inner start
774
                         (Nothing, RL r_csize $ Immediate 1)]
                                                                      -- call double(csize)
                        ++ concatMap pushRegisterToStack tmpRegisterList ++
775
776
                        [(Nothing, BRA l_m_entry)]
                                                                      -- call malloc1()
                        ++ invertInstructions(concatMap pushRegisterToStack tmpRegisterList) ++
777
```

```
[(Nothing, RR r_csize $ Immediate 1),
                                                                      -- uncall double(csize)
779
                         (Nothing, SUBI r_counter $ Immediate 1),
                                                                      -- counter -= 1
780
                          (Nothing, XOR r_tmp r_p),
                                                                      -- Copy current address of p
781
                         (Nothing, EXCH r_tmp r_fl),
                                                                      -- Store address in current
                             free list
                         (Nothing, ADD r_p r_csize),
                                                                      -- Set p to other half of the
782
                             block we're splitting
                         (Just l_i_assert, BNE rt2 registerZero l_i_assert_t),
783
                         (Nothing, EXCH r_tmp r_fl),
                         (Nothing, SUB r_p r_csize)]
785
                                                                      -- set r_e2_i1 <- p - csize !=
786
                         ++ l_e2_i1
                              free_list[counter]
                         ++ l_e2_i2
                                                                      -- set r_e2_i2 <- free_list[
787
                             counter] = 0
788
                         ++ l_e2_i3
                                                                      -- set r_e2_i3 <- r_e2_i1 ||
                              r_e2_i2
                         ++ [(Nothing, XOR rt2 r_e2_i3)]
                                                                      -- Set rt2 -> r_i_2
                                                                      -- Clear r_i_2
                         ++ invertInstructions l_e2_i3
790
791
                         ++ invertInstructions l_e2_i2
792
                         ++ invertInstructions l_e2_i1 ++
                         [ (Nothing, ADD r_p r_csize),
793
794
                          (Nothing, EXCH r_tmp r_fl),
                                                                      -- S2_outer end
795
                          (Just l_o_assert, BNE rt registerZero l_o_assert_t)]
                         ++ 1_e2_outer
                                                                       -- Set r_e2 -> c_size <
796
                             obj_size
                         ++ [(Nothing, XOR rt r_e2_outer)]
797
                                                                      -- r_t = r_e1_o
                                                                      -- Clear r_e1_o
798
                         ++ invertInstructions l_e2_outer
                                                                      -- Go to top
                         ++ [(Just l_m_bot, BRA l_m_top)]
          return malloc
800
801
       where pushRegisterToStack r = [(Nothing, EXCH r registerSP), (Nothing, SUBI registerSP $
           Immediate 1)]
802
803 cgMalloc :: CodeGenerator [(Maybe Label, MInstruction)]
804 cqMalloc =
805
        do rp <- tempRegister -- Pointer to new obj</pre>
806
          ros <- tempRegister -- Object size
          rc <- tempRegister -- Free list index
807
808
          rs <- tempRegister -- Current cell size
809
          popTempRegister >> popTempRegister >> popTempRegister >> popTempRegister
          let malloc = [(Just "l_malloc_top", BRA "l_malloc_bot")]
810
811
                        ++
                        [(Just "l_malloc", SWAPBR registerRO),
812
813
                         (Nothing, NEG registerRO),
                         (Nothing, ADDI rs $ Immediate 2),
814
                         (Nothing, XOR rc registerZero)]
815
816
                        ++ concatMap pop [rp, ros]
817
                        ++ push registerRO ++
                        [(Nothing, BRA "l_malloc1")]
818
819
                        ++ pop registerRO
                        ++ concatMap push [ros, rp] ++
820
821
                        [(Nothing, XOR rc registerZero),
                         (Nothing, SUBI rs $ Immediate 2),
822
                         (Just "l_malloc_bot", BRA "l_malloc_top")]
823
          return malloc
824
       where pop r = [(Nothing, ADDI registerSP $ Immediate 1), (Nothing, EXCH r registerSP)]
825
             push r = invertInstructions (pop r)
826
828 -- | Code generation for virtual tables
829 cgVirtualTables :: CodeGenerator [(Maybe Label, MInstruction)]
830 cgVirtualTables = concat <$> (gets (virtualTables . saState) >>= mapM vtInstructions)
       where vtInstructions (n, ms) = zip (vtLabel n) <$> mapM vtData ms
831
832
              vtData m = DATA . AddressMacro <$> getMethodLabel m
              vtLabel n = (Just $ "l_" ++ n ++ "_vt") : repeat Nothing
834
835 -- | Returns the main class label
836 getMainLabel :: CodeGenerator Label
```

```
837 getMainLabel = gets (mainMethod . saState) >>= getMethodLabel
838
839 -- | Fetches the main class from the class analysis state
840 getMainClass :: CodeGenerator TypeName
841 getMainClass = gets (mainClass . caState . saState) >>= \mbox{mc} ->
842
       case mc of
843
           (Just tp) -> return tp
           Nothing -> throwError "ICE: No main method defined"
844
845
846 -- | Fetches the field of a given type name
847 getFields :: TypeName -> CodeGenerator [VariableDeclaration]
848 getFields tp =
       do cs <- gets (classes . caState . saState)</pre>
849
850
          case lookup tp cs of
851
               (Just (GCDecl _ _ fs _)) -> return fs
              Nothing -> throwError $ "ICE: Unknown class " ++ tp
852
853
854 -- | Code generation for output
855 cgOutput :: TypeName -> CodeGenerator ([(Maybe Label, MInstruction)], [(Maybe Label,
       MInstruction) |)
856 cgOutput tp =
857
       do mfs <- getFields tp</pre>
858
          co <- concat <$> mapM cgCopyOutput (zip [1..] mfs)
          return (map cgStatic mfs, co)
859
       860
             cgCopyOutput(o, GDecl _ n) =
861
862
                 do rt <- tempRegister</pre>
                    ra <- tempRegister
                     popTempRegister >> popTempRegister
864
865
                     let copy = [ADDI registerThis ReferenceCounterIndex,
866
                                 ADDI registerThis $ Immediate o,
867
                                 EXCH rt registerThis,
                                 XORI ra $ AddressMacro $ "l_r_" ++ n,
868
869
                                 EXCH rt ra,
                                 XORI ra $ AddressMacro $ "l_r_" ++ n,
870
871
                                 SUBI registerThis $ Immediate o,
                                 SUBI registerThis ReferenceCounterIndex1
872
873
                    return $ zip (repeat Nothing) copy
875 -- | Generates code for the program entry point
876 cgProgram :: SProgram -> CodeGenerator PISA.MProgram
877 cgProgram p =
       do vt <- cgVirtualTables</pre>
878
          malloc <- cgMalloc
879
          malloc1 <- cgMalloc1
880
881
          rv <- tempRegister -- V table register
          rb <- tempRegister -- Memory block register
882
          popTempRegister >> popTempRegister
883
884
          ms <- concat <$> mapM cgMethod p
          l_main <- getMainLabel</pre>
885
886
          mtp <- getMainClass
887
          (out, co) <- cgOutput mtp
          let mvt = "1_" ++ mtp ++ "_vt"
888
              mn = [(Just "start", BRA "top"),
889
890
                     (Nothing, START),
                     (Nothing, ADDI registerFLPs ProgramSize),
                                                                   -- Init free list pointer list
891
                     (Nothing, XOR registerHP registerFLPs),
                                                                   -- Init heap pointer
892
                     (Nothing, ADDI registerHP FreeListsSize),
                                                                   -- Init space for FLPs list
893
                                                                   -- Store address of initial
894
                     (Nothing, XOR rb registerHP),
                         memory block in rb
                     (Nothing, ADDI registerFLPs FreeListsSize), -- Index to end of free lists
895
                     (Nothing, SUBI registerFLPs $ Immediate 1), -- Index to last element of free
896
                          lists
                     (Nothing, EXCH rb registerFLPs),
                                                                   -- Store address of first block
897
                         in last element of free lists
                     (Nothing, ADDI registerFLPs $ Immediate 1), -- Index to end of free lists
898
```

```
(Nothing, SUBI registerFLPs FreeListsSize), -- Index to beginning of free
899
                         lists
900
                     (Nothing, XOR registerSP registerHP),
                                                                   -- Init stack pointer 1/2
901
                     (Nothing, ADDI registerSP StackOffset),
                                                                    -- Init stack pointer 2/2
                     (Nothing, SUBI registerSP $ SizeMacro mtp), -- Allocate space for main on
902
                         stack
903
                     (Nothing, XOR registerThis registerSP),
                                                                    -- Store address of main object
                                                                    -- Store address of vtable in rv
                     (Nothing, XORI rv $ AddressMacro mvt),
904
                     (Nothing, EXCH rv registerThis),
                                                                    -- Add address of vtable to
                         stack
                                                                    -- Add address of vtable to
906
                     (Nothing, SUBI registerSP $ Immediate 1),
                         stack
                     (Nothing, EXCH registerThis registerSP),
                                                                    -- Push 'this' to stack
907
                                                                    -- Push 'this' to stack
908
                     (Nothing, SUBI registerSP $ Immediate 1),
                                                                    -- Execute main
909
                     (Nothing, BRA l_main),
                     (Nothing, ADDI registerSP $ Immediate 1),
                                                                    -- Pop 'this'
910
                                                                    -- Pop 'this'
911
                     (Nothing, EXCH registerThis registerSP)]
                      ++ co ++
912
913
                    [(Nothing, ADDI registerSP $ Immediate 1),
                                                                    -- Pop vtable address
914
                     (Nothing, EXCH rv registerThis),
                                                                    -- Pop vtable address
                                                                    -- Clear rv
                     (Nothing, XORI rv $ AddressMacro mvt),
915
916
                     (Nothing, XOR registerThis registerSP),
                                                                    -- Clear 'this'
917
                     (Nothing, ADDI registerSP $ SizeMacro mtp),
                                                                   -- Deallocate space for main
                     (Nothing, SUBI registerSP StackOffset),
                                                                    -- Clear stack pointer
918
                     (Nothing, XOR registerSP registerHP),
                                                                    -- Clear stack pointer
919
                     (Nothing, SUBI registerHP FreeListsSize),
                                                                    -- Reset Heap pointer (For
920
                         pretty output)
                     (Nothing, XOR registerHP registerFLPs),
                                                                    -- Reset Heap pointer (For
921
                        pretty output)
922
                     (Nothing, SUBI registerFLPs ProgramSize),
                                                                    -- Reset Free lists pointer (For
                          pretty output)
                     (Just "finish", FINISH)]
923
          return $ PISA.GProg $ [(Just "top", BRA "start")] ++ out ++ vt ++ malloc ++ malloc1 ++
924
              ms ++ mn
925
927 -- | Generates code for a program
928 generatePISA :: (SProgram, SAState) -> Except String (PISA.MProgram, SAState)
929 generatePISA (p, s) = second saState <$> runStateT (runCG $ cgProgram p) (initialState s)
931 showPISAProgram :: (Show a, MonadIO m) \Rightarrow (t, a) \rightarrow m ()
932 showPISAProgram (\_, s) = pPrint s
```

#### MacroExpander.hs

```
1 {-# LANGUAGE GeneralizedNewtypeDeriving #-}
3 module MacroExpander (expandMacros) where
5 import Data.Maybe
6 import Data.List
8 import Control.Monad.Reader
9 import Control.Monad.Except
10 import Control.Arrow
12 import AST hiding (Program, GProg, Offset)
13 import PISA
14
15 import Debug.Trace (trace, traceShow)
17 import ScopeAnalyzer
18 import ClassAnalyzer
20 type Size = Integer
21 type Address = Integer
22 type Offset = Integer
23
24 data MEState = MEState {
      addressTable :: [(Label, Address)],
25
26
       sizeTable :: [(TypeName, Size)],
27
      offsetTable :: [(TypeName, [(MethodName, Offset)])],
      programSize :: Size,
28
      freeListsSize :: Size,
29
30
      stackOffset :: Offset,
31
      initialMemoryBlockSize :: Size,
      referenceCounterIndex :: Offset,
      arrayElementOffset :: Offset
33
34 } deriving (Show, Eq)
35
36 newtype MacroExpander a = MacroExpander { runME :: ReaderT MEState (Except String) a }
37
       deriving (Functor, Applicative, Monad, MonadReader MEState, MonadError String)
38
39 -- | Returns the offset table generated from the an indexed virtual table
40 getOffsetTable :: SAState -> [(TypeName, [(MethodName, Offset)])]
41 getOffsetTable s = map (second (map toOffset)) indexedVT
42
       where indexedVT = map (second \$ zip [0..]) \$ virtualTables s
             toOffset (i, m) = (getName $ lookup m $ symbolTable s, i)
43
             getName (Just (Method _ n)) = n
44
45
             getName _ = error "ICE: Invalid method index"
46
47 -- \mid Initializes the macro state containing the address, size, offset tables and the program
48 initialState :: MProgram -> SAState -> MEState
49 initialState (GProg p) s = MEState {
      addressTable = mapMaybe toPair $ zip [0..] p,
       sizeTable = (classSize . caState) s,
51
       offsetTable = getOffsetTable s,
52
      programSize = genericLength p,
53
54
      freeListsSize = 10,
      stackOffset = 2048,
      initialMemoryBlockSize = 1024,
56
57
      referenceCounterIndex = 1,
58
      arrayElementOffset = 2
59 }
60
       where toPair (a, (Just l, \underline{\ })) = Just (l, a)
61
             toPair _ = Nothing
62
```

```
63 -- | Returns the address of a given label
64 getAddress :: Label -> MacroExpander Address
65 getAddress l = asks addressTable >>= \at ->
       case lookup 1 at of
           (Just i) -> return i
67
           Nothing -> throwError $ "ICE: Unknown label " ++ 1
68
70 -- | Returns the size of a given class name
71 getSize :: TypeName -> MacroExpander Size
72 getSize tn = asks sizeTable >>= \st ->
73
       case lookup tn st of
           (Just s) -> return s
75
           Nothing -> throwError $ "ICE: Unknown type " ++ tn
76
77 -- | Returns the off set of a method for a given class name and method
78 getOffset :: TypeName -> MethodName -> MacroExpander Offset
79 getOffset tn mn = asks offsetTable >>= \ot ->
       case lookup tn ot of
80
81
           Nothing -> throwError $ "ICE: Unknown type " ++ tn
           (Just mo) \rightarrow case lookup mn mo of
                            Nothing -> throwError $ "ICE: Unknown method " ++ mn
83
84
                             (Just o) -> return o
85
86 -- | Macro definitions
87 meMacro :: Macro -> MacroExpander Integer
88 meMacro (Immediate i) = return i
89 meMacro (AddressMacro 1) = getAddress 1
90 meMacro (SizeMacro tn) = getSize tn
91 meMacro (OffsetMacro tn mn) = getOffset tn mn
92 meMacro ProgramSize = asks programSize
93 meMacro FreeListsSize = asks freeListsSize
94 meMacro StackOffset = asks stackOffset
95 meMacro InitialMemoryBlockSize = asks initialMemoryBlockSize
96 meMacro ReferenceCounterIndex = asks referenceCounterIndex
97 meMacro ArrayElementOffset = asks arrayElementOffset
99 -- | Macro instructions
100 meInstruction :: MInstruction -> MacroExpander Instruction
101 meInstruction (ADD r1 r2) = return $ ADD r1 r2
102 meInstruction (ADDI r m) = ADDI r <$> meMacro m
103 meInstruction (ANDX r1 r2 r3) = return $ ANDX r1 r2 r3
104 meInstruction (ANDIX r1 r2 m) = ANDIX r1 r2 <$> meMacro m
105 meInstruction (NORX r1 r2 r3) = return $ NORX r1 r2 r3
106 meInstruction (NEG r) = return $ NEG r
107 meInstruction (ORX r1 r2 r3) = return $ ORX r1 r2 r3
108 meInstruction (ORIX r1 r2 m) = ORIX r1 r2 <$> meMacro m
109 meInstruction (RL r m) = RL r <> meMacro m
110 meInstruction (RLV r1 r2 ) = return $ RLV r1 r2
111 meInstruction (RR r m) = RR r <$> meMacro m
112 meInstruction (RRV r1 r2 ) = return $ RRV r1 r2
113 meInstruction (SLLX r1 r2 m) = SLLX r1 r2 <$> meMacro m
114 meInstruction (SLLVX r1 r2 r3) = return $ SLLVX r1 r2 r3
115 meInstruction (SRAX r1 r2 m) = SRAX r1 r2 <$> meMacro m
116 meInstruction (SRAVX r1 r2 r3) = return $ SRAVX r1 r2 r3
117 meInstruction (SRLX r1 r2 m) = SRLX r1 r2 <$> meMacro m
118 meInstruction (SRLVX r1 r2 r3) = return $ SRLVX r1 r2 r3
119 meInstruction (SUB r1 r2) = return $ SUB r1 r2
120 meInstruction (XOR r1 r2) = return $ XOR r1 r2
121 meInstruction (XORI r m) = XORI r <$> meMacro m
122 meInstruction (BEQ r1 r2 l) = return $ BEQ r1 r2 l
123 meInstruction (BGEZ r l) = return $ BGEZ r l
124 meInstruction (BGTZ r l) = return $ BGTZ r l
125 meInstruction (BLEZ r l) = return $ BLEZ r l
126 meInstruction (BLTZ r l) = return $ BLTZ r l
127 meInstruction (BNE r1 r2 l) = return $ BNE r1 r2 l
128 meInstruction (BRA 1) = return $ BRA 1
```

### ROOPLPPC.hs

```
1 import Control.Monad.Except
2 import System.IO
3 import System. Environment
5 import PISA
6 import Parser
7 import ClassAnalyzer
8 import ScopeAnalyzer
9 import TypeChecker
10 import CodeGenerator
11 import MacroExpander
12
13 import Data.List.Split
14
15 type Error = String
17 main :: IO ()
18 \text{ main} =
      do args <- getArgs</pre>
19
          when (null args) (error "Supply input filename.\nUsage: ROOPLPPC input.rplpp output.pal
20
          when (length args > 2) (error "Too many arguments.\nUsage: ROOPLPPC input.rplpp output.
21
              pal\n")
          handle <- openFile (head args) ReadMode
          input <- hGetContents handle
23
          let output = if length args == 2 then last args else head (splitOn "." (head args)) ++
24
               ".pal"
          either (hPutStrLn stderr) (writeProgram output) $ compileProgram input
25
26
27 compileProgram :: String -> Either Error PISA.Program
28 compileProgram s =
      runExcept $
      parseString s
30
31
      >>= classAnalysis
      >>= scopeAnalysis
32
      >>= typeCheck
33
34
      >>= generatePISA
     >>= expandMacros
35
```

# **Example Ouput**

### LinkedList.rplpp

```
class Cell
       Cell next
3
       int data
 5
      method constructor(int value)
6
          data ^= value
7
      method append(Cell cell)
9
          if next = nil & cell != nil then
               next <=> cell
                                                  // Store as next cell if current cell is end of
10
                   list
           else skip
11
           fi next != nil & cell = nil
12
^{13}
           \textbf{if} \ \texttt{next} \ != \ \textbf{nil} \ \textbf{then}
14
                                                // Recusively search until we reach end of list
15
               call next::append(cell)
16
           else skip
           fi next != nil
17
  class LinkedList
19
20
      Cell head
21
      int listLength
22
23
      method insertHead(Cell cell)
           if head = nil & cell != nil then
24
               head <=> cell
                                             // Set cell as head of list if list is empty
25
26
           else skip
27
           fi head != nil & cell = nil
28
       method appendCell(Cell cell)
           call insertHead(cell)
                                             // Insert as head if empty list
30
31
           if head != nil then
32
               call head::append(cell)
                                            // Iterate list until we reach the end, then insert
33
                   the node
34
           else skip
35
           fi head != nil
36
37
           listLength += 1
                                             // Increment lenght
38
      method prependCell(Cell cell)
39
           call insertHead(cell)
                                             // Insert as head if empty list
40
41
42
           if cell != nil & head != nil then
                                           // Set cell.next = head. head = nil after execution
               call cell::append(head)
```

```
else skip
         fi cell != nil & head = nil
45
46
47
         if cell != nil & head = nil then
            cell <=> head
                                     // Set head = cell. Cell is nil after execution
48
         else skip
49
         fi cell = nil & head != nil
50
51
52
         listLength += 1
                                     // Increment length
53
     method length(int result)
54
55
        result ^= listLength
56
57
  class Program
     LinkedList linkedList
58
     \quad \textbf{int} \ \text{sumResult}
59
60
     int listLength
61
62
     method main()
         63
         listLength += 10
64
65
66
         local int x = 0
         from x = 0 do
67
68
            skip
69
         loop
            local Cell cell = nil
70
71
            new Cell cell
                                             // Instantiate new cell
            72
73
74
            delocal Cell cell = nil
75
            x += 1
         until x = listLength
76
         delocal int x = listLength
77
```

# ${\bf LinkedList.pal}$

1	;; pendulum pal file				60		XORI	\$10 1
2	top:	BRA	sta	rt	61		ADDI	\$8 1
3	l r linkedList:	DATA	0		62		EXCH	\$19 \$17
4	l_r_sumResult:	DATA	0		63		XOR	\$18 \$19
			0					
5	l_r_listLength:	DATA			64		EXCH	\$19 \$17
6	l_Program_vt:	DATA	972		65		RL	\$9 1
7	l_LinkedList_vt:	DATA	459		66		EXCH	\$10 \$1
8		DATA	562		67		ADDI	\$1 -1
9		DATA	704		68		EXCH	\$11 \$1
10		DATA	945		69		ADDI	\$1 -1
	1 Coll w+.	DATA	223		70		EXCH	\$12 \$1
11	l_Cell_vt:							
12		DATA	252		71		ADDI	\$1 -1
13	l_malloc_top:	BRA	1_m	alloc_	_bot 72		EXCH	\$14 \$1
14	l_malloc:	SWAPBR	\$2		73		ADDI	\$1 -1
15		NEG	\$2		74		EXCH	\$16 \$1
16		ADDI	\$9 2	2	75		ADDI	\$1 -1
17		XOR	\$8		76		EXCH	\$17 \$1
18		ADDI	\$1	L	77		ADDI	\$1 -1
19		EXCH	\$6	\$1	78		EXCH	\$18 \$1
20		ADDI	\$1	l	79		ADDI	\$1 -1
21		EXCH	\$7		80		EXCH	\$20 \$1
22		EXCH	\$2		81		ADDI	\$1 -1
23		ADDI	\$1 -	-1	82		EXCH	\$21 \$1
$^{24}$		BRA	1_m	alloc1	. 83		ADDI	\$1 -1
25		ADDI	\$1	1	84		EXCH	\$22 \$1
26		EXCH	\$2	\$1	85		ADDI	\$1 -1
27		EXCH	\$7		86		EXCH	\$23 \$1
28		ADDI	\$1 .		87		ADDI	\$1 -1
29		EXCH	\$6	\$1	88		BRA	l_malloc1
30		ADDI	\$1 -	-1	89		ADDI	\$1 1
31		XOR	\$8	\$0	90		EXCH	\$23 \$1
32		ADDI	\$9 -		91		ADDI	\$1 1
	l_malloc_bot:	BRA		alloc_			EXCH	\$22 \$1
33					-			
34	l_malloc1_top:	BRA	_		_bot 93		ADDI	\$1 1
35		ADDI	\$1	L	94		EXCH	\$21 \$1
36		EXCH	\$2	\$1	95		ADDI	\$1 1
37		SUB	\$17	\$8	96		EXCH	\$20 \$1
38		XOR	\$17		97		ADDI	\$1 1
39	l_malloc1:	SWAPBR		т -	98		EXCH	\$18 \$1
	I_MailOCi.							
40		NEG	\$2		99		ADDI	\$1 1
41		EXCH	\$2	\$1	100		EXCH	\$17 \$1
42		ADDI	\$1 .	-1	101		ADDI	\$1 1
43		XOR	\$17	\$4	102		EXCH	\$16 \$1
44		ADD	\$17		103		ADDI	\$1 1
		EXCH		\$17			EXCH	
45					104			\$14 \$1
46		XOR		\$19	105		ADDI	\$1 1
47		EXCH		\$17	106		EXCH	\$12 \$1
48		XOR	\$13	\$9	107		ADDI	\$1 1
49		SUB	\$13	\$7	108		EXCH	\$11 \$1
50	cmp_top_7:	BGEZ			ot_8109		ADDI	\$1 1
51	1	XORI	\$14		110		EXCH	\$10 \$1
	amp bot 0.							
52	cmp_bot_8:	BGEZ		-	op_7111		RR	\$9 1
53		XOR		\$14	112		ADDI	\$8 -1
54	cmp_bot_8_i:	BGEZ	\$13		113		XORI	\$10 1
	cmp_top_7_i				114	l_o_assert_true:	BRA	l_o_assert
55	<del>-</del>	XORI	\$14	1		l_o_test_false:	BRA	l_o_test
56	cmp_top_7_i:	BGEZ	\$13	_	116	cmp_top_11:	BEQ	\$18 \$0
50		DGBA	ΛTϽ		110		DEQ	7±0 40
	cmp_bot_8_i		A	A 77		cmp_bot_12		400 4
57		ADD	\$13		117		XORI	\$20 1
58		XOR	\$13	\$9	118	cmp_bot_12:	BEQ	\$18 \$0
59	l_o_test:	BEQ	\$10	\$0		cmp_top_11		
	l_o_test_false				119		XOR	\$11 \$20

120	cmp_bot_12_i:	BEQ	\$18 \$0	183		EXCH	\$12 \$17
	cmp_top_11_i			184		ADD	\$6 \$9
121		XORI	\$20 1	185	l_i_assert:	BNE	\$11 \$0
122	cmp_top_11_i:	BEQ	\$18 \$0		l_i_assert_true		
	cmp_bot_12_i			186		EXCH	\$12 \$17
123		BEQ	\$11 \$0	187		SUB	\$6 \$9
	l_i_test_false			188	cmp_top_13:	BEQ	\$6 \$12
124		XORI	\$11 1		cmp_bot_14		
125		ADD	\$6 \$18	189		XORI	\$21 1
126		SUB	\$18 \$6	190	cmp_bot_14:	BEQ	\$6 \$12
127		EXCH	\$12 \$6	İ	cmp_top_13		
128		EXCH	\$12 \$17	191	cmp_top_15:	BNE	\$12 \$0
129		XOR	\$12 \$6		cmp_bot_16		
130		XORI	\$11 1	192	_	XORI	\$22 1
131	l_i_assert_true:	BRA	l_i_assert	193	cmp_bot_16:	BNE	\$12 \$0
132		BRA	l_i_test		cmp_top_15		
133		ADDI	\$8 1	194	1 - 1 -	ORX	\$23 \$21 \$22
134		RL	\$9 1	195		XOR	\$11 \$23
135		EXCH	\$10 \$1	196		ORX	\$23 \$21 \$22
136		ADDI	\$1 -1	197	cmp_bot_16_i:	BNE	\$12 \$0
137		EXCH	\$11 \$1	131	cmp_top_15_i	2112	Y12 Y0
138		ADDI	\$1 -1	198	Cmp_cop_13_1	XORI	\$22 1
139		EXCH	\$12 \$1	199	cmp_top_15_i:	BNE	\$12 \$0
				199		DINE	712 70
140		ADDI	\$1 -1		cmp_bot_16_i	DE0	0.6 010
141		EXCH	\$14 \$1	200	cmp_bot_14_i:	BEQ	\$6 \$12
142		ADDI	\$1 -1		cmp_top_13_i		+04 4
143		EXCH	\$16 \$1	201		XORI	\$21 1
144		ADDI	\$1 -1	202	cmp_top_13_i:	BEQ	\$6 \$12
145		EXCH	\$17 \$1		cmp_bot_14_i		
146		ADDI	\$1 -1	203		ADD	\$6 \$9
147		EXCH	\$18 \$1	204		EXCH	\$12 \$17
148		ADDI	\$1 -1	205	l_o_assert:	BNE	\$10 \$0
149		EXCH	\$20 \$1		l_o_assert_true		
150		ADDI	\$1 -1	206		XOR	\$15 \$9
151		EXCH	\$21 \$1	207		SUB	\$15 \$7
152		ADDI	\$1 -1	208	cmp_top_9:	BGEZ	\$15 cmp_bot_10
153		EXCH	\$22 \$1	209		XORI	\$16 1
154		ADDI	\$1 -1	210	cmp_bot_10:	BGEZ	\$15 cmp_top_9
155		EXCH	\$23 \$1	211		XOR	\$10 \$16
156		ADDI	\$1 -1	212	cmp_bot_10_i:	BGEZ	\$15
157		BRA	l_malloc1		cmp_top_9_i		
158		ADDI	\$1 1	213	1 - 1 -	XORI	\$16 1
159		EXCH	\$23 \$1	214	cmp_top_9_i:	BGEZ	\$15
160		ADDI	\$1 1		cmp bot 10 i		
161		EXCH	\$22 \$1	215		ADD	\$15 \$7
162		ADDI	\$1 1	216		XOR	\$15 \$9
163		EXCH	\$21 \$1		l_malloc1_bot:	BRA	l_malloc1_top
164		ADDI	\$1 1		l_constructor_5_top:	BRA	
165		EXCH	\$20 \$1		l_constructor_5_bot		
166		ADDI	\$1 1	219		ADDI	\$1 1
167		EXCH	\$18 \$1	220		EXCH	\$2 \$1
168		ADDI	\$1 1	221		EXCH	\$6 \$1
169		EXCH	\$17 \$1	222		ADDI	\$1 -1
170		ADDI	\$1 1	223		EXCH	\$3 \$1
171		EXCH	\$16 \$1	224		ADDI	\$1 -1
171		ADDI	\$1 1		l_constructor_5:	SWAPBR	
172		EXCH	\$14 \$1	226	1_001130140001_0.	NEG	\$2
173		ADDI	\$14 \$1	226		ADDI	\$2 \$1 1
		EXCH				EXCH	\$3 \$1
175			\$12 \$1	228			
176		ADDI	\$1 1	229		ADDI	\$1 1
177		EXCH	\$11 \$1	230		EXCH	\$6 \$1
178		ADDI	\$1 1	231		EXCH	\$2 \$1
179		EXCH	\$10 \$1	232		ADDI	\$1 -1
180		RR	\$9 1	233		ADD	\$7 \$3
181		ADDI	\$8 -1	234		ADDI	\$7 3
182		XOR	\$12 \$6	235		EXCH	\$8 \$7

	I		÷= 0	1			
236		ADDI	\$7 -3		cmp_bot_22_i		
237		SUB	\$7 \$3	290		ADD	\$8 \$3
238		EXCH	\$9 \$6	291		ADDI	\$8 2
239		XOR	\$8 \$9	292		EXCH	\$9 \$8
240		EXCH	\$9 \$6	293		ADDI	\$8 -2
241		ADD	\$7 \$3	294		SUB	\$8 \$3
242		ADDI	\$7 3	295	test_17:	BEQ	\$7 \$0
243		EXCH	\$8 \$7		test_false_19	_	
244		ADDI	\$7 -3	296		XORI	\$7 1
245		SUB	\$7 \$3	297		ADD	\$8 \$3
	l gangt mugt an E bot.		7/ 73			ADDI	
246	l_constructor_5_bot:	BRA		298			\$8 2
	l_constructor_5_top			299		EXCH	\$9 \$8
247	l_append_6_top:	BRA	l_append_			ADDI	\$8 -2
248		ADDI	\$1 1	301		SUB	\$8 \$3
249		EXCH	\$2 \$1	302		EXCH	\$10 \$6
250		EXCH	\$6 \$1	303	swap_27:	XOR	\$9 \$10
251		ADDI	\$1 -1	304		XOR	\$10 \$9
252		EXCH	\$3 \$1	305		XOR	\$9 \$10
253		ADDI	\$1 -1	306		EXCH	\$10 \$6
254	l_append_6:	SWAPBR	\$2	307		ADD	\$8 \$3
255		NEG	\$2	308		ADDI	\$8 2
256		ADDI	\$1 1	309		EXCH	\$9 \$8
257		EXCH	\$3 \$1	310		ADDI	\$8 -2
		ADDI	\$1 1			SUB	
258				311			\$8 \$3
259		EXCH	\$6 \$1	312		XORI	\$7 1
260		EXCH	\$2 \$1	313	assert_true_18:	BRA	assert_20
261		ADDI	\$1 -1	314	test_false_19:	BRA	test_17
262		ADD	\$8 \$3	315	assert_20:	BNE	\$7 \$0
263		ADDI	\$8 2		assert_true_18		
264		EXCH	\$9 \$8	316		ADD	\$8 \$3
265		ADDI	\$8 -2	317		ADDI	\$8 2
266		SUB	\$8 \$3	318		EXCH	\$9 \$8
267	cmp_top_21:	BNE	\$9 \$0	319		ADDI	\$8 -2
	cmp_bot_22		1 - 1 -	320		SUB	\$8 \$3
268		XORI	\$10 1	321	cmp_top_28:	BEQ	\$9 \$0
269	amp hot 22.	BNE	\$9 \$0	321	cmp_bot_29	DDQ	Ψ5 Ψ0
209	cmp_bot_22:	DNE	79 70		CIUP_DOC_29	WORT	610 1
	cmp_top_21		411 46	322	1	XORI	\$10 1
270		EXCH	\$11 \$6	323	cmp_bot_29:	BEQ	\$9 \$0
271	cmp_top_23:	BEQ	\$11 \$0		cmp_top_28		
	cmp_bot_24			324		EXCH	\$11 \$6
272		XORI	\$12 1	325	cmp_top_30:	BNE	\$11 \$0
273	cmp_bot_24:	BEQ	\$11 \$0		cmp_bot_31		
	cmp_top_23			326		XORI	\$12 1
274		ANDX	\$13 \$10 \$	12 327	cmp_bot_31:	BNE	\$11 \$0
275	f_top_25:	BEQ	\$13 \$0		cmp_top_30		
	f_bot_26	~		328	1 = 1 = 1	ANDX	\$13 \$10 \$12
276		XORI	\$14 1	329	f_top_32:	BEQ	\$13 \$0
	f_bot_26:	BEQ	\$13 \$0	020	f_bot_33		710 70
211	f_top_25	<b></b> 2	+10 90	330	1_200_00	XORI	\$14 1
0.70	1_cop_23	VOD	Ċ7 Ċ1 <i>A</i>		£ 1-+ 22.		
278	F 1-+ 26 :.	XOR	\$7 \$14	331	f_bot_33:	BEQ	\$13 \$0
279		BEQ	\$13 \$0		f_top_32		
	f_top_25_i			332		XOR	\$7 \$14
280		XORI	\$14 1	333	f_bot_33_i:	BEQ	\$13 \$0
281	f_top_25_i:	BEQ	\$13 \$0		f_top_32_i		
	f_bot_26_i			334		XORI	\$14 1
282		ANDX	\$13 \$10 \$	12 335	f_top_32_i:	BEQ	\$13 \$0
283	cmp_bot_24_i:	BEQ	\$11 \$0		f_bot_33_i		
	cmp_top_23_i	_	• •	336		ANDX	\$13 \$10 \$12
284		XORI	\$12 1	337	cmp_bot_31_i:	BNE	\$11 \$0
285	cmp_top_23_i:	BEQ	\$11 \$0	551	cmp_top_30_i		122 70
200		בייע	~ 1	990	Cmb_cob_20_1	YORT	¢12 1
000	cmp_bot_24_i	EVCE	¢11 ¢7	338	amp + op 20 :	XORI	\$12 1
286		EXCH	\$11 \$6	339	cmp_top_30_i:	BNE	\$11 \$0
287	*	BNE	\$9 \$0		cmp_bot_31_i		411 45
	cmp_top_21_i			340		EXCH	\$11 \$6
288		XORI	\$10 1	341		BEQ	\$9 \$0
289	cmp_top_21_i:	BNE	\$9 \$0		cmp_top_28_i		

i							
342		XORI	\$10 1	398		ADDI	\$13 -397
343	cmp_top_28_i:	BEQ	\$9 \$0	399	1_jmp_43:	SWAPBR	\$13
	cmp_bot_29_i			400		NEG	\$13
344		ADD	\$8 \$3	401		ADDI	\$13 397
345		ADDI	\$8 2	402		ADDI	\$1 1
346		EXCH	\$9 \$8	403		EXCH	\$10 \$1
347		ADDI	\$8 -2	404		ADDI	\$1 1
348		SUB	\$8 \$3	405		EXCH	\$6 \$1
349		ADD	\$8 \$3	406		ADDI	\$1 1
350		ADDI	\$8 2	407		EXCH	\$3 \$1
351		EXCH	\$9 \$8	408		ADD	\$8 \$3
352		ADDI	\$8 -2	- 1		ADDI	\$8 2
				409			
353		SUB	\$8 \$3	410		EXCH	\$9 \$8
354	cmp_top_38:	BEQ	\$9 \$0	411		ADDI	\$8 -2
	cmp_bot_39			412		SUB	\$8 \$3
355		XORI	\$10 1	413		EXCH	\$11 \$10
356	cmp_bot_39:	BEQ	\$9 \$0	414		ADDI	\$11 1
000	cmp_top_38	z	73 70	415		EXCH	\$12 \$11
		DE0	¢10 ¢0	- 1			
357	f_top_40:	BEQ	\$10 \$0	416		XOR	\$13 \$12
	f_bot_41			417		EXCH	\$12 \$11
358		XORI	\$11 1	418		ADDI	\$11 -1
359	f_bot_41:	BEQ	\$10 \$0	419	loadMetAdd_42_i:	EXCH	\$11 \$10
	f_top_40			420		XOR	\$10 \$9
360		XOR	\$7 \$11	421		ADD	\$8 \$3
	£ 1 41 ±.			- 1			
361	f_bot_41_i:	BEQ	\$10 \$0	422		ADDI	\$8 2
	f_top_40_i			423		EXCH	\$9 \$8
362		XORI	\$11 1	424		ADDI	\$8 -2
363	f_top_40_i:	BEQ	\$10 \$0	425		SUB	\$8 \$3
	f_bot_41_i			426		XORI	\$7 1
364	cmp_bot_39_i:	BEQ	\$9 \$0		assert_true_35:	BRA	assert_37
004		222	77 70		test_false_36:	BRA	
	cmp_top_38_i	WODT	Ċ10 1				test_34
365		XORI	\$10 1	429	assert_37:	BNE	\$7 \$0
366	cmp_top_38_i:	BEQ	\$9 \$0		assert_true_35		
	cmp_bot_39_i			430		ADD	\$8 \$3
367		ADD	\$8 \$3	431		ADDI	\$8 2
368		ADDI	\$8 2	432		EXCH	\$9 \$8
369		EXCH	\$9 \$8	433		ADDI	\$8 -2
				- 1			
370		ADDI	\$8 -2	434		SUB	\$8 \$3
371		SUB	\$8 \$3	435	cmp_top_44:	BEQ	\$9 \$0
372	test_34:	BEQ	\$7 \$0		cmp_bot_45		
	test_false_36			436		XORI	\$10 1
373		XORI	\$7 1	437	cmp_bot_45:	BEQ	\$9 \$0
374		ADD	\$8 \$3	i	cmp_top_44		
375		ADDI	\$8 2	438	f_top_46:	BEQ	\$10 \$0
		EXCH	\$9 \$8	400	=	DEQ	V10 V0
376					f_bot_47		A11 1
377		ADDI	\$8 -2	439	6.1	XORI	\$11 1
378		SUB	\$8 \$3	440	f_bot_47:	BEQ	\$10 \$0
379		XOR	\$10 \$9		f_top_46		
380	loadMetAdd_42:	EXCH	\$11 \$10	441		XOR	\$7 \$11
381		ADDI	\$11 1	442	f_bot_47_i:	BEQ	\$10 \$0
382		EXCH	\$12 \$11		f_top_46_i	-	
383		XOR	\$13 \$12	443		XORI	\$11 1
				- 1	6 1 46 .		
384		EXCH	\$12 \$11	444	f_top_46_i:	BEQ	\$10 \$0
385		ADDI	\$11 -1		f_bot_47_i		
386		EXCH	\$11 \$10	445	cmp_bot_45_i:	BEQ	\$9 \$0
387		ADD	\$8 \$3		cmp_top_44_i		
388		ADDI	\$8 2	446		XORI	\$10 1
389		EXCH	\$9 \$8	447	cmp_top_44_i:	BEQ	\$9 \$0
390		ADDI	\$8 -2		cmp_bot_45_i		1 - 1 -
1				440	C!!.b_DOC_40_1	ADD.	60 63
391		SUB	\$8 \$3	448		ADD	\$8 \$3
392		EXCH	\$3 \$1	449		ADDI	\$8 2
393		ADDI	\$1 -1	450		EXCH	\$9 \$8
394		EXCH	\$6 \$1	451		ADDI	\$8 -2
395		ADDI	\$1 -1	452		SUB	\$8 \$3
396		EXCH	\$10 \$1	- 1	l_append_6_bot:	BRA	l_append_6_top
397		ADDI	\$1 -1		l_insertHead_1_top:	BRA	~rracop
391		TUUL	Υ T	404		DIVE	

ĺ	l_insertHead_1_bot			507		ADDI	\$8 -2
455	1_1113C1C11C44_1_D0C	ADDI	\$1 1	508		SUB	\$8 \$3
456		EXCH	\$2 \$1	509		EXCH	\$10 \$6
457		EXCH	\$6 \$1	510	  swap_58:	XOR	\$9 \$10
458		ADDI	\$1 -1	511		XOR	\$10 \$9
459		EXCH	\$3 \$1	512		XOR	\$9 \$10
460		ADDI	\$1 -1	513		EXCH	\$10 \$6
461	l_insertHead_1:	SWAPBR		514		ADD	\$8 \$3
462		NEG	\$2	515		ADDI	\$8 2
463		ADDI	\$1 1	516		EXCH	\$9 \$8
464		EXCH	\$3 \$1	517		ADDI	\$8 -2
465		ADDI	\$1 1	518		SUB	\$8 \$3
466		EXCH	\$6 \$1	519		XORI	\$7 1
467		EXCH	\$2 \$1	520	assert_true_49:	BRA	assert_51
468		ADDI	\$1 -1	521	test_false_50:	BRA	test_48
469		ADD	\$8 \$3	522	assert_51:	BNE	\$7 \$0
470		ADDI	\$8 2		assert_true_49		
471		EXCH	\$9 \$8	523		ADD	\$8 \$3
472		ADDI	\$8 -2	524		ADDI	\$8 2
473		SUB	\$8 \$3	525		EXCH	\$9 \$8
474	cmp_top_52:	BNE	\$9 \$0	526		ADDI	\$8 -2
	cmp_bot_53			527		SUB	\$8 \$3
475		XORI	\$10 1	528	cmp_top_59:	BEQ	\$9 \$0
476	cmp_bot_53:	BNE	\$9 \$0		cmp_bot_60		
	cmp_top_52			529		XORI	\$10 1
477		EXCH	\$11 \$6	530	cmp_bot_60:	BEQ	\$9 \$0
478	cmp_top_54:	BEQ	\$11 \$0		cmp_top_59		***
	cmp_bot_55		***	531		EXCH	\$11 \$6
479	1 . 55	XORI	\$12 1	532	cmp_top_61:	BNE	\$11 \$0
480	cmp_bot_55:	BEQ	\$11 \$0		cmp_bot_62	VODT	610 1
401	cmp_top_54	7 110 17	¢12 ¢10 ¢10	533		XORI	\$12 1
481	f + on E6.	ANDX	\$13 \$10 \$12	534	cmp_bot_62:	BNE	\$11 \$0
482	f_top_56: f_bot_57	BEQ	\$13 \$0	535	cmp_top_61	ANDX	\$13 \$10 \$12
483	1_000_37	XORI	\$14 1	536	  f_top_63:	BEQ	\$13 \$10 \$12
484	f_bot_57:	BEQ	\$13 \$0	330	f_bot_64	DEQ	713 70
101	f_top_56	DDg	413 40	537	1_300_01	XORI	\$14 1
485	1_00P_00	XOR	\$7 \$14	538	f_bot_64:	BEQ	\$13 \$0
486	f_bot_57_i:	BEQ	\$13 \$0		f_top_63	2	1 1-
	f_top_56_i	~		539		XOR	\$7 \$14
487		XORI	\$14 1	540	f_bot_64_i:	BEQ	\$13 \$0
488	f_top_56_i:	BEQ	\$13 \$0		f_top_63_i		
	f_bot_57_i			541	<u>-</u>	XORI	\$14 1
489		ANDX	\$13 \$10 \$12	542	f_top_63_i:	BEQ	\$13 \$0
490	cmp_bot_55_i:	BEQ	\$11 \$0		f_bot_64_i		
	cmp_top_54_i			543		ANDX	\$13 \$10 \$12
491		XORI	\$12 1	544		BNE	\$11 \$0
492	cmp_top_54_i:	BEQ	\$11 \$0		cmp_top_61_i		
	cmp_bot_55_i			545		XORI	\$12 1
493	1	EXCH	\$11 \$6	546	cmp_top_61_i:	BNE	\$11 \$0
494	cmp_bot_53_i:	BNE	\$9 \$0		cmp_bot_62_i		
	cmp_top_52_i		***	547		EXCH	\$11 \$6
495		XORI	\$10 1	548	cmp_bot_60_i:	BEQ	\$9 \$0
496	cmp_top_52_i:	BNE	\$9 \$0	- 10	cmp_top_59_i	VODT	¢10 1
407	cmp_bot_53_i	מחג	¢0 ¢3	549	amp top 50 i.	XORI	\$10 1
497		ADD	\$8 \$3	550	cmp_top_59_i:	BEQ	\$9 \$0
498		ADDI EXCH	\$8 2 \$9 \$8	gen	cmp_bot_60_i	מתג	\$8 \$3
499 500		ADDI	\$8 -2	551 552		ADD ADDI	\$8 \$3 \$8 2
501		SUB	\$8 \$3	553		EXCH	\$9 \$8
502	test_48:	BEQ	\$7 \$0	554		ADDI	\$8 -2
332	test_false_50	z	, , , ,	555		SUB	\$8 \$3
503		XORI	\$7 1	556	l_insertHead_1_bot:	BRA	
504		ADD	\$8 \$3		l_insertHead_1_top		
505		ADDI	\$8 2	557	l_appendCell_2_top:	BRA	
506		EXCH	\$9 \$8		l_appendCell_2_bot		
					:		

					1			
558		ADDI	\$1		615		XOR	\$13 \$12
559		EXCH	\$2	\$1	616		EXCH	\$12 \$11
560		EXCH	\$6	\$1	617		ADDI	\$11 -1
561		ADDI	\$1	-1	618		EXCH	\$11 \$10
562		EXCH	\$3	\$1	619		ADD	\$8 \$3
							ADDI	
563		ADDI	\$1	-1	620			\$8 2
564	l_appendCell_2:	SWAPBR			621		EXCH	\$9 \$8
565		NEG	\$2		622		ADDI	\$8 -2
566		ADDI	\$1	1	623		SUB	\$8 \$3
567		EXCH	\$3	\$1	624		EXCH	\$3 \$1
568		ADDI	\$1		625		ADDI	\$1 -1
		EXCH	\$6		626		EXCH	\$6 \$1
569								
570		EXCH	\$2		627		ADDI	\$1 -1
571		ADDI	\$1	-1	628		EXCH	\$10 \$1
572		EXCH	\$6	\$1	629		ADDI	\$1 -1
573		ADDI	\$1	-1	630		ADDI	\$13 -629
574		EXCH	\$3		631	l_jmp_74:	SWAPBR	
						1_Jmp_/4.		
575		ADDI	\$1		632		NEG	\$13
576		BRA	l_i	nsert	Head_ <b>6</b> 33		ADDI	\$13 629
577		ADDI	\$1	1	634		ADDI	\$1 1
578		EXCH	\$3	\$1	635		EXCH	\$10 \$1
579		ADDI	\$1	1	636		ADDI	\$1 1
580		EXCH	\$6		637		EXCH	\$6 \$1
581		ADD	\$8		638		ADDI	\$1 1
582		ADDI	\$8	2	639		EXCH	\$3 \$1
583		EXCH	\$9	\$8	640		ADD	\$8 \$3
584		ADDI	\$8	-2	641		ADDI	\$8 2
585		SUB	\$8		642		EXCH	\$9 \$8
586	cmp_top_69:	BEQ	\$9	\$0	643		ADDI	\$8 -2
	cmp_bot_70				644		SUB	\$8 \$3
587		XORI	\$10	) 1	645		EXCH	\$11 \$10
588	cmp_bot_70:	BEQ	\$9	\$0	646		ADDI	\$11 1
	cmp_top_69	~			647		EXCH	\$12 \$11
F00		BEO.	¢10	0 0 0				
589	f_top_71:	BEQ	ŞΙU	\$0	648		XOR	\$13 \$12
	f_bot_72				649		EXCH	\$12 \$11
590		XORI	\$11	. 1	650		ADDI	\$11 -1
591	f_bot_72:	BEQ	\$10	\$0	651	loadMetAdd_73_i:	EXCH	\$11 \$10
	 f_top_71	_			652		XOR	\$10 \$9
592	1_00p_,1	XOR	¢7	\$11			ADD	\$8 \$3
	6.1 . 70 .				653			
593	f_bot_72_i:	BEQ	ŞIU	) \$0	654		ADDI	\$8 2
	f_top_71_i				655		EXCH	\$9 \$8
594		XORI	\$11	. 1	656		ADDI	\$8 -2
595	f_top_71_i:	BEQ	\$10	\$0	657		SUB	\$8 \$3
	f_bot_72_i	_			658		XORI	\$7 1
500		BEQ	\$9	ĊΩ	659	2000+ + 200 66.	BRA	
596	cmp_bot_70_i:	PEÕ	γJ	γU		assert_true_66:		assert_68
	cmp_top_69_i				660	test_false_67:	BRA	test_65
597		XORI	\$10		661	assert_68:	BNE	\$7 \$0
598	cmp_top_69_i:	BEQ	\$9	\$0		assert_true_66		
	cmp_bot_70_i				662		ADD	\$8 \$3
599	<del>-</del>	ADD	\$8	\$3	663		ADDI	\$8 2
600		ADDI	\$8		664		EXCH	\$9 \$8
601		EXCH	\$9		665		ADDI	\$8 -2
602		ADDI	\$8	-2	666		SUB	\$8 \$3
603		SUB	\$8	\$3	667	cmp_top_75:	BEQ	\$9 \$0
604	test_65:	BEQ	\$7	\$0		cmp_bot_76		
	test_false_67	~			668	1 - 1 - 1	XORI	\$10 1
COF	0000_10100_07	VODT	ċ7	1		amp bat 76.		
605		XORI	\$7		669	cmp_bot_76:	BEQ	\$9 \$0
606		ADD	\$8			cmp_top_75		
607		ADDI	\$8	2	670	f_top_77:	BEQ	\$10 \$0
608		EXCH	\$9	\$8		f_bot_78		
609		ADDI	\$8		671		XORI	\$11 1
610		SUB	\$8		672	f_bot_78:	BEQ	\$10 \$0
					012		השם	4 T O
611		XOR		) \$9		f_top_77		
612	loadMetAdd_73:	EXCH	\$11	\$10	673		XOR	\$7 \$11
613		ADDI	\$11	. 1	674	f_bot_78_i:	BEQ	\$10 \$0
614		EXCH		\$11		f_top_77_i		
,								

675		XORI	\$11 1	733		XORI	\$12 1
676	f_top_77_i:	BEQ	\$10 \$0	734	cmp_bot_86:	BEQ	\$11 \$0
	f_bot_78_i	_			cmp_top_85	_	
677	cmp_bot_76_i:	BEQ	\$9 \$0	735	op_cop_co	ANDX	\$13 \$9 \$12
011	=	בחק	<b>Ψ 9 Ψ 0</b>		£ + 07.		
	cmp_top_75_i		410 1	736		BEQ	\$13 \$0
678		XORI	\$10 1		f_bot_88		
679	cmp_top_75_i:	BEQ	\$9 \$0	737		XORI	\$14 1
	cmp_bot_76_i			738	f_bot_88:	BEQ	\$13 \$0
680		ADD	\$8 \$3		f_top_87		
681		ADDI	\$8 2	739		XOR	\$7 \$14
682		EXCH	\$9 \$8	740	f_bot_88_i:	BEQ	\$13 \$0
		ADDI	\$8 -2	140	f_top_87_i	222	713 70
683					1_top_0/_1		6141
684		SUB	\$8 \$3	741		XORI	\$14 1
685		ADD	\$7 \$3	742	f_top_87_i:	BEQ	\$13 \$0
686		ADDI	\$7 3		f_bot_88_i		
687		EXCH	\$8 \$7	743		ANDX	\$13 \$9 \$12
688		ADDI	\$7 -3	744	cmp_bot_86_i:	BEQ	\$11 \$0
689		SUB	\$7 \$3		cmp_top_85_i	_	
690		XORI	\$9 1	745	ob_cob_co_1	XORI	\$12 1
		ADD			amp tap OF :		
691			\$8 \$9	746	cmp_top_85_i:	BEQ	\$11 \$0
692		XORI	\$9 1		cmp_bot_86_i		
693		ADD	\$7 \$3	747		ADD	\$10 \$3
694		ADDI	\$7 3	748		ADDI	\$10 2
695		EXCH	\$8 \$7	749		EXCH	\$11 \$10
696		ADDI	\$7 -3	750		ADDI	\$10 -2
697		SUB	\$7 \$3	751		SUB	\$10 \$3
698	l_appendCell_2_bot:	BRA	Ψ7 Ψ3	752	cmp_bot_84_i:	BEQ	\$8 \$0
698		DKA		152	*	PFŐ	30 3U
	l_appendCell_2_top				cmp_top_83_i		
699	l_prependCell_3_top:	BRA		753		XORI	\$9 1
	l_prependCell_3_bot			754	cmp_top_83_i:	BEQ	\$8 \$0
700		ADDI	\$1 1		cmp_bot_84_i		
701		EXCH	\$2 \$1	755		EXCH	\$8 \$6
702		EXCH	\$6 \$1	756	test_79:	BEQ	\$7 \$0
703		ADDI	\$1 -1	100	test_false_81	LLV	7 7 9 0
					cesc_raise_or	VODT	67 1
704		EXCH	\$3 \$1	757		XORI	\$7 1
705		ADDI	\$1 -1	758		EXCH	\$8 \$6
706	l_prependCell_3:	SWAPBR	\$2	759		XOR	\$9 \$8
707		NEG	\$2	760	loadMetAdd_89:	EXCH	\$10 \$9
708		ADDI	\$1 1	761		ADDI	\$10 1
709		EXCH	\$3 \$1	762		EXCH	\$11 \$10
710		ADDI	\$1 1	763		XOR	\$12 \$11
		EXCH	\$6 \$1			EXCH	
711				764			\$11 \$10
712		EXCH	\$2 \$1	765		ADDI	\$10 -1
713		ADDI	\$1 -1	766		EXCH	\$10 \$9
714		EXCH	\$6 \$1	767		EXCH	\$8 \$6
715		ADDI	\$1 -1	768		ADD	\$13 \$3
716		EXCH	\$3 \$1	769		ADDI	\$13 2
717		ADDI	\$1 -1	770		EXCH	\$3 \$1
718		BRA	l_insertHead_			ADDI	\$1 -1
719		ADDI	\$1 1	772		EXCH	\$6 \$1
720		EXCH	\$3 \$1	773		ADDI	\$1 -1
721		ADDI	\$1 1	774		EXCH	\$13 \$1
722		EXCH	\$6 \$1	775		ADDI	\$1 -1
723		EXCH	\$8 \$6	776		EXCH	\$9 \$1
724	cmp_top_83:	BEQ	\$8 \$0	777		ADDI	\$1 -1
	cmp_bot_84	•		778		ADDI	\$12 -777
725	<u> </u>	XORI	\$9 1	779	1_jmp_90:	SWAPBR	
	cmp bot 84.					NEG	\$12
726		BEQ	\$8 \$0	780			
	cmp_top_83		\$10 \$0	781		ADDI	\$12 777
727		ADD	\$10 \$3	782		ADDI	\$1 1
728		ADDI	\$10 2	783		EXCH	\$9 \$1
729		EXCH	\$11 \$10	784		ADDI	\$1 1
730		ADDI	\$10 -2	785		EXCH	\$13 \$1
731		SUB	\$10 \$3	786		ADDI	\$1 1
732	cmp_top_85:	BEQ	\$11 \$0	787		EXCH	\$6 \$1
.02	cmp_bot_86		1 + +	788		ADDI	\$1 1
	Cmp_50c_60			100		דעעה	7 + +

789		EXCH	\$3 \$1	841		XORI	\$9 1
790		ADDI	\$13 -2	842	cmp_bot_102:	BEQ	\$8 \$0
791		SUB	\$13 \$3		cmp_top_101	_	
792		EXCH	\$8 \$6	843		ADD	\$10 \$3
793		EXCH	\$10 \$9	844		ADDI	\$10 2
794		ADDI	\$10 1	845		EXCH	\$11 \$10
795		EXCH	\$11 \$10	846		ADDI	\$10 -2
796		XOR	\$12 \$11	847		SUB	\$10 \$3
797		EXCH	\$11 \$10	848	cmp_top_103:	BNE	\$11 \$0
798		ADDI	\$10 -1		cmp_bot_104		
799	loadMetAdd_89_i:	EXCH	\$10 \$9	849		XORI	\$12 1
800	TOUGHTE CHACLO J_I.	XOR	\$9 \$8		cmp bot 104:	BNE	\$11 \$0
				850		DINE	SII SO
801		EXCH	\$8 \$6		cmp_top_103		
802		XORI	\$7 1	851		ANDX	\$13 \$9 \$12
803	assert_true_80:	BRA	assert_82	852	f_top_105:	BEQ	\$13 \$0
804	test_false_81:	BRA	test_79		f_bot_106		
805	assert_82:	BNE	\$7 \$0	853		XORI	\$14 1
	assert_true_80				f_bot_106:	BEQ	\$13 \$0
806	ubboro_cruo_c	EXCH	\$8 \$6	001	f_top_105		720 70
	amp + ap 01.		\$8 \$0	055	1_000_100	VOD	¢7 ¢1/
807	cmp_top_91:	BEQ	30 3U	855		XOR	\$7 \$14
	cmp_bot_92			856	f_bot_106_i:	BEQ	\$13 \$0
808		XORI	\$9 1		f_top_105_i		
809	cmp_bot_92:	BEQ	\$8 \$0	857		XORI	\$14 1
	cmp_top_91			858	f_top_105_i:	BEQ	\$13 \$0
810		ADD	\$10 \$3		f_bot_106_i	_	
811		ADDI	\$10 2	859	1_200_100_1	ANDX	\$13 \$9 \$12
812		EXCH	\$11 \$10		cmp_bot_104_i:	BNE	\$11 \$0
				860	– –	DINE	SII SO
813		ADDI	\$10 -2		cmp_top_103_i		
814		SUB	\$10 \$3	861		XORI	\$12 1
815	cmp_top_93:	BNE	\$11 \$0	862	cmp_top_103_i:	BNE	\$11 \$0
	cmp_bot_94				cmp_bot_104_i		
816		XORI	\$12 1	863		ADD	\$10 \$3
817	cmp_bot_94:	BNE	\$11 \$0	864		ADDI	\$10 2
011	*		411 40	865		EXCH	\$11 \$10
	cmp_top_93	7.110.17	612 60 610				
818		ANDX	\$13 \$9 \$12	866		ADDI	\$10 -2
819	f_top_95:	BEQ	\$13 \$0	867		SUB	\$10 \$3
	f_bot_96			868	cmp_bot_102_i:	BEQ	\$8 \$0
820		XORI	\$14 1		cmp_top_101_i		
821	f_bot_96:	BEQ	\$13 \$0	869		XORI	\$9 1
	f_top_95	_		870	cmp_top_101_i:	BEQ	\$8 \$0
822		XOR	\$7 \$14		cmp_bot_102_i	2	1 - 1 -
	f_bot_96_i:	BEQ	\$13 \$0	071	Cmp_b0c_102_1	EXCH	\$8 \$6
823		PFÕ	3T2 30	871	07		
	f_top_95_i		A14 1	872	test_97:	BEQ	\$7 \$0
824		XORI	\$14 1		test_false_99		
825	f_top_95_i:	BEQ	\$13 \$0	873		XORI	\$7 1
ļ	f_bot_96_i			874		EXCH	\$8 \$6
826		ANDX	\$13 \$9 \$12	875		ADD	\$9 \$3
827	cmp_bot_94_i:	BNE	\$11 \$0	876		ADDI	\$9 2
	cmp_top_93_i		• •	877		EXCH	\$10 \$9
828	CWA_CON_30_T	XORI	\$12 1	878		ADDI	\$9 -2
	amp + ap 02 ; ;						
829	cmp_top_93_i:	BNE	\$11 \$0	879	107	SUB	\$9 \$3
	cmp_bot_94_i			880	swap_107:	XOR	\$8 \$10
830		ADD	\$10 \$3	881		XOR	\$10 \$8
831		ADDI	\$10 2	882		XOR	\$8 \$10
832		EXCH	\$11 \$10	883		ADD	\$9 \$3
833		ADDI	\$10 -2	884		ADDI	\$9 2
834		SUB	\$10 \$3	885		EXCH	\$10 \$9
835	cmp_bot_92_i:	BEQ	\$8 \$0	886		ADDI	\$9 -2
030		המת	YU YU				
	cmp_top_91_i	we==	ĆO 1	887		SUB	\$9 \$3
836		XORI	\$9 1	888		EXCH	\$8 \$6
837	cmp_top_91_i:	BEQ	\$8 \$0	889		XORI	\$7 1
	cmp_bot_92_i			890	assert_true_98:	BRA	assert_100
838		EXCH	\$8 \$6	891	test_false_99:	BRA	test_97
839		EXCH	\$8 \$6	892	assert_100:	BNE	\$7 \$0
840	cmp_top_101:	BEQ	\$8 \$0		assert_true_98		•
- 10	cmp_bot_102	2		893		EXCH	\$8 \$6
ļ	CP_200C_102			093	I		+0 40

894	cmp_top_108:	BNE	\$8 \$0	947	l_length_4:	SWAPBR	\$2
	cmp_bot_109			948		NEG	\$2
895		XORI	\$9 1	949		ADDI	\$1 1
896	cmp_bot_109:	BNE	\$8 \$0	950		EXCH	\$3 \$1
	cmp_top_108		1 - 1 -	951		ADDI	\$1 1
207	Cmp_cop_100	ADD	\$10 \$3			EXCH	\$6 \$1
897				952			
898		ADDI	\$10 2	953		EXCH	\$2 \$1
899		EXCH	\$11 \$10	954		ADDI	\$1 -1
900		ADDI	\$10 -2	955		EXCH	\$7 \$6
901		SUB	\$10 \$3	956		ADD	\$8 \$3
902	cmp_top_110:	BEQ	\$11 \$0	957		ADDI	\$8 3
	cmp_bot_111			958		EXCH	\$9 \$8
903		XORI	\$12 1	959		ADDI	\$8 -3
904	cmp_bot_111:	BEQ	\$11 \$0	960		SUB	\$8 \$3
304	cmp_top_110	222	711 70	961		XOR	\$7 \$9
005	Cmp_cop_110	7 10 17	¢12 ¢0 ¢10				
905		ANDX	\$13 \$9 \$12	962		ADD	\$8 \$3
906	f_top_112:	BEQ	\$13 \$0	963		ADDI	\$8 3
	f_bot_113			964		EXCH	\$9 \$8
907		XORI	\$14 1	965		ADDI	\$8 -3
908	f_bot_113:	BEQ	\$13 \$0	966		SUB	\$8 \$3
	f_top_112			967		EXCH	\$7 \$6
909		XOR	\$7 \$14	968	l_length_4_bot:	BRA	l_length_4_top
	f_bot_113_i:	BEQ	\$13 \$0	969	l_main_0_top:	BRA	l_main_0_bot
310	f_top_112_i	222	410 40	970	<u></u>	ADDI	\$1 1
011	1_00P_112_1	VODT	Ċ1 / 1				
911		XORI	\$14 1	971		EXCH	\$2 \$1
912	f_top_112_i:	BEQ	\$13 \$0	972		EXCH	\$3 \$1
	f_bot_113_i			973		ADDI	\$1 -1
913		ANDX	\$13 \$9 \$12	974	l_main_0:	SWAPBR	\$2
914	cmp_bot_111_i:	BEQ	\$11 \$0	975		NEG	\$2
	cmp_top_110_i			976		ADDI	\$1 1
915		XORI	\$12 1	977		EXCH	\$3 \$1
916	cmp_top_110_i:	BEQ	\$11 \$0	978		EXCH	\$2 \$1
	cmp_bot_111_i		1 1-	979		ADDI	\$1 -1
917	Cmp_000_111_1	ADD	\$10 \$3	980		EXCH	\$3 \$1
918		ADDI	\$10 2	981	114	ADDI	\$1 -1
919		EXCH	\$11 \$10	982	obj_con_114:	ADDI	\$8 4
920		ADDI	\$10 -2	983		EXCH	\$8 \$1
921		SUB	\$10 \$3	984		ADDI	\$1 -1
922	cmp_bot_109_i:	BNE	\$8 \$0	985		EXCH	\$7 \$1
	cmp_top_108_i			986		ADDI	\$1 -1
923		XORI	\$9 1	987		BRA	l_malloc
924	cmp_top_108_i:	BNE	\$8 \$0	988		ADDI	\$1 1
021	cmp_bot_109_i		70 70	989		EXCH	\$7 \$1
025	Cmp_boc_105_1	EXCH	\$8 \$6	990		ADDI	\$1 1
925		ADD					
926			\$7 \$3	991	114	EXCH	\$8 \$1
927		ADDI	\$7 3		obj_con_114_i:	ADDI	\$8 -4
928		EXCH	\$8 \$7	993		ADDI	\$1 1
929		ADDI	\$7 -3	994		EXCH	\$3 \$1
930		SUB	\$7 \$3	995		ADD	\$6 \$3
931		XORI	\$9 1	996		ADDI	\$6 2
932		ADD	\$8 \$9	997		XORI	\$8 5
933		XORI	\$9 1	998		EXCH	\$8 \$7
934		ADD	\$7 \$3	999		ADDI	\$7 1
935		ADDI	\$7 3	1000		XORI	\$8 1
936		EXCH	\$8 \$7	1001		EXCH	\$8 \$7
937		ADDI	\$7 -3	1001	obj_con_114_bot:	ADDI	\$7 -1
					00.1_0011_114_000:		
938	1 10 . 12 . 2 . 1	SUB	\$7 \$3	1003		EXCH	\$7 \$6
939	l_prependCell_3_bot:	BRA		1004		ADDI	\$6 -2
	l_prependCell_3_top			1005		SUB	\$6 \$3
940	l_length_4_top:	BRA	l_length_4_	.b <b>ott</b> 06		ADD	\$6 \$3
941		ADDI	\$1 1	1007		ADDI	\$6 4
942		EXCH	\$2 \$1	1008		EXCH	\$7 \$6
943		EXCH	\$6 \$1	1009		ADDI	\$6 -4
944		ADDI	\$1 -1	1010		SUB	\$6 \$3
945		EXCH	\$3 \$1	1011		XORI	\$8 10
946		ADDI	\$1 -1	1011		ADD	\$7 \$8
940		-1001	~ ± ±	1012			7, 40

1013		XORI	\$8 10	1062		ADDI	\$9 -4
1014		ADD	\$6 \$3	1063		SUB	\$9 \$3
1015		ADDI	\$6 4	1064		EXCH	\$8 \$6
1016		EXCH	\$7 \$6		test_116:	BNE	\$7 \$0 exit_118
1017		ADDI	\$6 -4		localBlock_128:	XOR	\$8 \$1
1018		SUB	\$6 \$3	3 1067		XOR	\$9 \$0
1019	localBlock_133:	XOR	\$6 \$1	L 1068		EXCH	\$9 \$1
1020		XOR	\$7 \$0	1069		ADDI	\$1 -1
1021		EXCH	\$7 \$1	1070		EXCH	\$3 \$1
1022		ADDI	\$1 -1			ADDI	\$1 -1
1023		XORI	\$7 1			EXCH	\$8 \$1
1	115			1072			
1024	entry_115:	BEQ	\$7 \$0	1073		ADDI	\$1 -1
	assert_117			1074		EXCH	\$6 \$1
1025		EXCH	\$8 \$6	5 1075		ADDI	\$1 -1
1026	cmp_top_119:	BNE	\$8 \$0	1076	obj_con_123:	ADDI	\$10 4
	cmp_bot_120			1077		EXCH	\$10 \$1
1027		XORI	\$9 1	1078		ADDI	\$1 -1
	l+ 120.						
1028	cmp_bot_120:	BNE	\$8 \$0			EXCH	\$9 \$1
	cmp_top_119			1080		ADDI	\$1 -1
1029	f_top_121:	BEQ	\$9 \$0	1081		BRA	l_malloc
	f_bot_122			1082		ADDI	\$1 1
1030		XORI	\$10 1	1083		EXCH	\$9 \$1
1031	f_bot_122:	BEQ	\$9 \$0			ADDI	\$1 1
1001	f_top_121		42 40	1085		EXCH	\$10 \$1
	1_cop_121		00 01		100		
1032		XOR	\$7 \$1		obj_con_123_i:	ADDI	\$10 -4
1033	f_bot_122_i:	BEQ	\$9 \$0	1087		ADDI	\$1 1
	f_top_121_i			1088		EXCH	\$6 \$1
1034		XORI	\$10 1	L 1089		ADDI	\$1 1
1035	f_top_121_i:	BEQ	\$9 \$0	1090		EXCH	\$8 \$1
	f_bot_122_i			1091		ADDI	\$1 1
1000		DME	60 60			EXCH	\$3 \$1
1036	cmp_bot_120_i:	BNE	\$8 \$0				
	cmp_top_119_i			1093		XORI	\$10 9
1037		XORI	\$9 1	1094		EXCH	\$10 \$9
1038	cmp_top_119_i:	BNE	\$8 \$0	1095		ADDI	\$9 1
	cmp_bot_120_i			1096		XORI	\$10 1
1039	-	EXCH	\$8 \$6	5 1097		EXCH	\$10 \$9
1040		EXCH	\$8 \$6		obj_con_123_bot:	ADDI	\$9 -1
		ADD	\$9 \$3		05]_051_125_500:	EXCH	\$9 \$8
1041							
1042		ADDI	\$9 4	1100		EXCH	\$9 \$8
1043		EXCH	\$10 \$			XOR	\$10 \$9
1044		ADDI	\$9 -4	1102	loadMetAdd_124:	EXCH	\$11 \$10
1045		SUB	\$9 \$3	1103		ADDI	\$11 0
1046	cmp_top_129:	BNE	\$8 \$1	1104		EXCH	\$12 \$11
	cmp_bot_130			1105		XOR	\$13 \$12
1047		XORI	\$11 1			EXCH	\$12 \$11
	1 1.20						
1048	cmp_bot_130:	BNE	\$8 \$1			ADDI	\$11 0
	cmp_top_129			1108		EXCH	\$11 \$10
1049	f_top_131:	BEQ	\$11 \$	1109		EXCH	\$9 \$8
	f_bot_132			1110		EXCH	\$3 \$1
1050		XORI	\$12 1	L 1111		ADDI	\$1 -1
1051	f_bot_132:	BEQ	\$11 \$			EXCH	\$8 \$1
	f_top_131	~	+	1113		ADDI	\$1 -1
1050		YOP	¢7 ¢1			EXCH	\$6 \$1
1052	£ 120 :.	XOR	\$7 \$1				
1053	f_bot_132_i:	BEQ	\$11 \$			ADDI	\$1 -1
	f_top_131_i			1116		EXCH	\$10 \$1
1054		XORI	\$12 1	L 1117		ADDI	\$1 -1
1055	f_top_131_i:	BEQ	\$11 \$	50 1118		ADDI	\$13 -1117
	f_bot_132_i	•		1119	l_jmp_125:	SWAPBR	
1056	cmp_bot_130_i:	BNE	\$8 \$1			NEG	\$13
1000	cmp_top_129_i		+ O Y 1	1120		ADDI	\$13 1117
1655	Cmp_cop_123_1	VORT	611 1				
1057	100	XORI	\$11 1			ADDI	\$1 1
1058	cmp_top_129_i:	BNE	\$8 \$1			EXCH	\$10 \$1
	cmp_bot_130_i			1124		ADDI	\$1 1
1059		ADD	\$9 \$3	1125		EXCH	\$6 \$1
1060		ADDI	\$9 4	1126		ADDI	\$1 1
1061		EXCH	\$10 \$			EXCH	\$8 \$1
1001			7±0 Y		ı		1 - T <del>-</del>

i							+0.0
1128		ADDI	\$1 1	1194		ADDI	\$9 -2
1129		EXCH	\$3 \$1	1195		SUB	\$9 \$3
1130		EXCH	\$9 \$8	1196		ADDI	\$1 1
1131		EXCH	\$11 \$10	1197		EXCH	\$9 \$1
1132		ADDI	\$11 0	1198		XOR	\$9 \$0
1133		EXCH	\$12 \$11	1199	localBlock_128_i:	XOR	\$8 \$1
1134		XOR	\$13 \$12	1200		EXCH	\$8 \$6
1135		EXCH	\$12 \$11	1201		XORI	\$9 1
1136		ADDI	\$11 0	1202		ADD	\$8 \$9
	1						
1137	loadMetAdd_124_i:	EXCH	\$11 \$10	1203		XORI	\$9 1
1138		XOR	\$10 \$9	1204		EXCH	\$8 \$6
1139		EXCH	\$9 \$8	1205	assert_117:	BRA	entry_115
1140		ADD	\$9 \$3	1206	exit_118:	BRA	test_116
1141		ADDI	\$9 2	1207		XORI	\$7 1
1142		EXCH	\$10 \$9	1208		ADD	\$7 \$3
1143		ADDI	\$9 -2	1209		ADDI	\$7 4
1144		SUB	\$9 \$3	1210		EXCH	\$8 \$7
1145		XOR	\$11 \$10	1211		ADDI	\$7 -4
1146	loadMetAdd_126:	EXCH	\$12 \$11	1212		SUB	\$7 \$3
1147	10441001144_120.	ADDI	\$12 1	1213		ADDI	\$1 1
		EXCH				EXCH	
1148			\$13 \$12	1214			\$9 \$1
1149		XOR	\$14 \$13	1215	1 151 1 100 '	XOR	\$9 \$8
1150		EXCH	\$13 \$12	1216	localBlock_133_i:	XOR	\$6 \$1
1151		ADDI	\$12 -1	1217		ADD	\$7 \$3
1152		EXCH	\$12 \$11	1218		ADDI	\$7 4
1153		ADD	\$9 \$3	1219		EXCH	\$8 \$7
1154		ADDI	\$9 2	1220		ADDI	\$7 -4
1155		EXCH	\$10 \$9	1221		SUB	\$7 \$3
1156		ADDI	\$9 -2	1222	l_main_0_bot:	BRA	l_main_0_top
1157		SUB	\$9 \$3	1223	start:	BRA	top
1158		EXCH	\$3 \$1	1224		START	-
1159		ADDI	\$1 -1	1225		ADDI	\$4 1279
1160		EXCH	\$6 \$1	1226		XOR	\$5 \$4
1161		ADDI	\$1 -1	1227		ADDI	\$5 10
1162		EXCH	\$8 \$1	1228		XOR	\$7 \$5
		ADDI				ADDI	
1163			\$1 -1	1229			\$4 10
1164		EXCH	\$11 \$1	1230		ADDI	\$4 -1
1165		ADDI	\$1 -1	1231		EXCH	\$7 \$4
1166		ADDI	\$14 -1165	1232		ADDI	\$4 1
1167	1_jmp_127:	SWAPBR		1233		ADDI	\$4 -10
1168		NEG	\$14	1234		XOR	\$1 \$5
1169		ADDI	\$14 1165	1235		ADDI	\$1 2048
1170		ADDI	\$1 1	1236		ADDI	\$1 -8
1171		EXCH	\$11 \$1	1237		XOR	\$3 \$1
1172		ADDI	\$1 1	1238		XORI	\$6 4
1173		EXCH	\$8 \$1	1239		EXCH	\$6 \$3
1174		ADDI	\$1 1	1240		ADDI	\$1 -1
1175		EXCH	\$6 \$1	1241		EXCH	\$3 \$1
1176		ADDI	\$1 1	1242		ADDI	\$1 -1
1177		EXCH	\$3 \$1	1243		BRA	l_main_0
1177		ADD	\$9 \$3	1243		ADDI	\$1 1
1179		ADDI	\$9 2	1245		EXCH	\$3 \$1
1180		EXCH	\$10 \$9	1246		ADDI	\$3 1
1181		ADDI	\$9 -2	1247		ADDI	\$3 1
1182		SUB	\$9 \$3	1248		EXCH	\$6 \$3
1183		EXCH	\$12 \$11	1249		XORI	\$7 1
1184		ADDI	\$12 1	1250		EXCH	\$6 \$7
1185		EXCH	\$13 \$12	1251		XORI	\$7 1
1186		XOR	\$14 \$13	1252		ADDI	\$3 -1
1187		EXCH	\$13 \$12	1253		ADDI	\$3 -1
1188		ADDI	\$12 -1	1254		ADDI	\$3 1
1189	loadMetAdd_126_i:	EXCH	\$12 \$11	1255		ADDI	\$3 2
1190		XOR	\$11 \$10	1256		EXCH	\$6 \$3
1191		ADD	\$9 \$3	1257		XORI	\$7 2
1192		ADDI	\$9 2	1258		EXCH	\$6 \$7
1193		EXCH	\$10 \$9	1259		XORI	\$7 2
				_55			•

1260	ADDI	\$3	-2	1271		EXCH	\$6	\$3
1261	ADDI	\$3	-1	1272		XORI	\$6	4
1262	ADDI	\$3	1	1273		XOR	\$3	\$1
1263	ADDI	\$3	3	1274		ADDI	\$1	8
1264	EXCH	\$6	\$3	1275		ADDI	\$1	-2048
1265	XORI	\$7	3	1276		XOR	\$1	\$5
1266	EXCH	\$6	\$7	1277		ADDI	\$5	-10
1267	XORI	\$7	3	1278		XOR	\$5	\$4
1268	ADDI	\$3	-3	1279		ADDI	\$4	-1279
1269	ADDI	\$3	-1	1280 fi	inish:	FINISH		
1270	ADDI	\$1	1	·				

### BinaryTree.rplpp

```
class Node
       Node left
 3
       Node right
       int value
5
 6
       method setValue(int newValue)
           value ^= newValue
 8
9
       method insertNode(Node node, int nodeValue)
                                                                // Determine if we insert left or
           if nodeValue < value then</pre>
10
               if left = nil & node != nil then
11
                    left <=> node
                                                                // If open left node, store here
12
                else skip
13
               fi left != nil & node = nil
14
15
               if left != nil then
16
17
                    call left::insertNode(node, nodeValue) // If current node has left, continue
18
               else skip
               fi left != nil
19
20
           else
               if right = nil & node != nil then
21
22
                   right <=> node
                                                                // If open right node spot, store here
23
                else skip
               fi right != nil & node = nil
24
25
26
                if right != nil then
                    call right::insertNode(node, nodeValue) // If current node has, continue
27
               else skip
28
                \begin{tabular}{ll} \bf fi & right & != nil \\ \end{tabular} 
29
30
           fi nodeValue < value</pre>
31
32
       method getSum(int result)
33
           result += value
                                               // Add the value of this node to the sum
34
35
           if left != nil then
               call left::getSum(result)
                                             // If we have a left child, follow that path
36
           else skip
                                              // Else, skip
37
38
           fi left != nil
39
40
           if right != nil then
               call right::getSum(result) // If we have a right child, follow that path
41
                                              // Else, skip
           else skip
42
43
            fi right != nil
44
       method mirror()
45
           left <=> right
                                              // Swap left and right children
46
47
           if left = nil then skip
48
           else call left::mirror()
                                              // Recursively swap children if left != nil
49
           fi left = nil
50
51
52
           if right = nil then skip
                                              // Recursively swap children if right != nil
53
           else call right::mirror()
           fi right = nil
54
55
56
  class Tree
57
      Node root
58
59
       method insertNode(Node node, int value)
60
           if root = nil & node != nil then
```

```
root <=> node
62
           else skip
           fi root != nil & node = nil
63
64
           if root != nil then
65
66
               call root::insertNode(node, value)
           else skip
67
           fi root != nil
68
69
70
       method sum(int result)
           if root != nil then
71
72
               call root::getSum(result)
           else skip
73
           fi root != nil
74
75
       method mirror()
76
           if root != nil then
77
             call root::mirror()
78
           else skip
79
80
           fi root != nil
81
82
   class Program
83
       int sumResult
       Tree tree
84
85
       int nodeCount
86
       method main()
87
           new Tree tree
           nodeCount += 3
89
90
91
           local int x = 0
           from x = 0 do
92
93
               skip
94
           loop
                local Node node = nil
95
96
                new Node node
                                                  // Init new node
                call node::setValue(x)
                                                 // Set node value
97
98
                call tree::insertNode(node, x) // Insert node in tree
99
                delocal Node node = nil
               x += 1
100
101
           until x = nodeCount
           delocal int x = nodeCount
102
103
           call tree::sum(sumResult)
105
           call tree::mirror()
```

# ${\bf Binary Tree.pal}$

1	;; pendulum pal file					l_o_test_false		
2	top:	BRA	stai	rt	61		XORI	\$10 1
3	l_r_sumResult:	DATA	0		62		ADDI	\$8 1
4	l_r_tree:	DATA	0		63		EXCH	\$19 \$17
5	l_r_nodeCount:	DATA	0		64		XOR	\$18 \$19
6	l_Program_vt:	DATA	1644		65		EXCH	\$19 \$17
7	l_Tree_vt:	DATA	1201		66		RL	\$9 1
8		DATA	1414		67		EXCH	\$10 \$1
9	3. Nr. 3.	DATA	1532	2	68		ADDI	\$1 -1
10	l_Node_vt:	DATA DATA	224 255		69		EXCH ADDI	\$11 \$1
11 12		DATA	727		70 71		EXCH	\$1 -1 \$12 \$1
13		DATA	962		72		ADDI	\$1 -1
14	l_malloc_top:	BRA		alloc_bot			EXCH	\$14 \$1
15	l_malloc:	SWAPBR			74		ADDI	\$1 -1
16	_	NEG	\$2		75		EXCH	\$16 \$1
17		ADDI	\$9 2	2	76		ADDI	\$1 -1
18		XOR	\$8 \$	\$0	77		EXCH	\$17 \$1
19		ADDI	\$1 1	1	78		ADDI	\$1 -1
20		EXCH	\$6 5	\$1	79		EXCH	\$18 \$1
21		ADDI	\$1 1		80		ADDI	\$1 -1
22		EXCH	\$7 \$		81		EXCH	\$20 \$1
23		EXCH	\$2.5		82		ADDI	\$1 -1
24		ADDI	\$1 -		83		EXCH	\$21 \$1
25		BRA ADDI	\$1 1	alloc1	84		ADDI EXCH	\$1 -1 \$22 \$1
26 27		EXCH	\$2.5		85 86		ADDI	\$1 -1
28		EXCH	\$7 \$		87		EXCH	\$23 \$1
29		ADDI	\$1 -		88		ADDI	\$1 -1
30		EXCH	\$6 \$		89		BRA	l_malloc1
31		ADDI	\$1 -	-1	90		ADDI	\$1 1
32		XOR	\$8 \$	\$0	91		EXCH	\$23 \$1
33		ADDI	\$9 -	-2	92		ADDI	\$1 1
34	l_malloc_bot:	BRA		alloc_top			EXCH	\$22 \$1
35	l_malloc1_top:	BRA		alloc1_bo			ADDI	\$1 1
36		ADDI	\$1 1		95		EXCH	\$21 \$1
37		EXCH	\$2.5		96		ADDI	\$1 1
38 39		SUB XOR	\$17 \$17		97		EXCH ADDI	\$20 \$1 \$1 1
40	l_malloc1:	SWAPBR		74	98 99		EXCH	\$18 \$1
41	i_maiioci.	NEG	\$2		100		ADDI	\$1 1
42		EXCH	\$2.5	\$1	101		EXCH	\$17 \$1
43		ADDI	\$1 -		102		ADDI	\$1 1
44		XOR	\$17	\$4	103		EXCH	\$16 \$1
45		ADD	\$17	\$8	104		ADDI	\$1 1
46		EXCH	\$19	\$17	105		EXCH	\$14 \$1
47		XOR		\$19	106		ADDI	\$1 1
48		EXCH		\$17	107		EXCH	\$12 \$1
49		XOR	\$13		108		ADDI	\$1 1
50		SUB	\$13		109		EXCH	\$11 \$1
51 52	cmp_top_8:	BGEZ XORI	\$13	cmp_bot_	111		ADDI EXCH	\$1 1 \$10 \$1
52 53	cmp_bot_9:	BGEZ		cmp_top_			RR	\$9 1
54		XOR		\$14	113		ADDI	\$8 -1
55	cmp_bot_9_i:	BGEZ	\$13		114		XORI	\$10 1
	cmp_top_8_i					l_o_assert_true:	BRA	l_o_assert
56		XORI	\$14	1	116	 l_o_test_false:	BRA	l_o_test
57	cmp_top_8_i:	BGEZ	\$13		117	cmp_top_12:	BEQ	\$18 \$0
	cmp_bot_9_i					cmp_bot_13		
58		ADD	\$13		118		XORI	\$20 1
59		XOR	\$13		119	cmp_bot_13:	BEQ	\$18 \$0
60	l_o_test:	BEQ	\$10	\$ O		cmp_top_12		

				1			
120		XOR	\$11 \$20	183		XOR	\$12 \$6
121	cmp_bot_13_i:	BEQ	\$18 \$0	184		EXCH	\$12 \$17
	cmp_top_12_i	WODT	¢00 1	185		ADD	\$6 \$9
122		XORI	\$20 1	186	l_i_assert:	BNE	\$11 \$0
123	cmp_top_12_i:	BEQ	\$18 \$0		l_i_assert_true		410 410
	cmp_bot_13_i		411 40	187		EXCH	\$12 \$17
124	l_i_test:	BEQ	\$11 \$0	188		SUB	\$6 \$9
	l_i_test_false			189	cmp_top_14:	BEQ	\$6 \$12
125		XORI	\$11 1		cmp_bot_15		
126		ADD	\$6 \$18	190		XORI	\$21 1
127		SUB	\$18 \$6	191	cmp_bot_15:	BEQ	\$6 \$12
128		EXCH	\$12 \$6		cmp_top_14		
129		EXCH	\$12 \$17	192	cmp_top_16:	BNE	\$12 \$0
130		XOR	\$12 \$6		cmp_bot_17		
131		XORI	\$11 1	193		XORI	\$22 1
132	l_i_assert_true:	BRA	l_i_assert	194	cmp_bot_17:	BNE	\$12 \$0
133	l_i_test_false:	BRA	l_i_test		cmp_top_16		
134		ADDI	\$8 1	195		ORX	\$23 \$21 \$22
135		RL	\$9 1	196		XOR	\$11 \$23
136		EXCH	\$10 \$1	197		ORX	\$23 \$21 \$22
137		ADDI	\$1 -1	198	cmp_bot_17_i:	BNE	\$12 \$0
138		EXCH	\$11 \$1		cmp_top_16_i		
139		ADDI	\$1 -1	199		XORI	\$22 1
140		EXCH	\$12 \$1	200	cmp_top_16_i:	BNE	\$12 \$0
141		ADDI	\$1 -1		cmp_bot_17_i		
142		EXCH	\$14 \$1	201	cmp_bot_15_i:	BEQ	\$6 \$12
143		ADDI	\$1 -1		cmp_top_14_i	_	
144		EXCH	\$16 \$1	202	- 1 1	XORI	\$21 1
145		ADDI	\$1 -1	203	cmp_top_14_i:	BEQ	\$6 \$12
146		EXCH	\$17 \$1	200	cmp_bot_15_i		70 722
147		ADDI	\$1 -1	204	ob_200_10_1	ADD	\$6 \$9
148		EXCH	\$18 \$1	205		EXCH	\$12 \$17
149		ADDI	\$1 -1	206	l_o_assert:	BNE	\$10 \$0
150		EXCH	\$20 \$1	200	l_o_assert_true	2112	710 70
151		ADDI	\$1 -1	207	1_0_assere_erae	XOR	\$15 \$9
152		EXCH	\$21 \$1	208		SUB	\$15 \$7
153		ADDI	\$1 -1	209	cmp_top_10:	BGEZ	\$15 cmp_bot_11
154		EXCH	\$22 \$1	210	cmp_cop_io.	XORI	\$16 1
155		ADDI	\$1 -1	211	cmp_bot_11:	BGEZ	\$15 cmp_top_10
156		EXCH	\$23 \$1	212	Cmp_b0c_11.	XOR	\$10 \$16
		ADDI	\$1 -1		amp bot 11 ; .		
157		BRA		213	cmp_bot_11_i:	BGEZ	\$15
158			l_malloc1	01.4	cmp_top_10_i	XORI	¢1.C 1
159		ADDI	\$1 1	214	+ 10 ÷-		\$16 1
160		EXCH	\$23 \$1	215	cmp_top_10_i:	BGEZ	\$15
161		ADDI	\$1 1		cmp_bot_11_i		615 67
162		EXCH	\$22 \$1	216		ADD	\$15 \$7
163		ADDI	\$1 1	217	l mallogi ba+.	XOR	\$15 \$9
164		EXCH	\$21 \$1		l_malloc1_bot:	BRA	l_malloc1_top
165		ADDI	\$1 1	219	l_setValue_4_top:	BRA	
166		EXCH	\$20 \$1		l_setValue_4_bot	300-	¢1 1
167		ADDI	\$1 1	220		ADDI	\$1 1
168		EXCH	\$18 \$1	221		EXCH	\$2 \$1
169		ADDI	\$1 1	222		EXCH	\$6 \$1
170		EXCH	\$17 \$1	223		ADDI	\$1 -1
171		ADDI	\$1 1	224		EXCH	\$3 \$1
172		EXCH	\$16 \$1	225		ADDI	\$1 -1
173		ADDI	\$1 1	226	l_setValue_4:	SWAPBR	
174		EXCH	\$14 \$1	227		NEG	\$2
175		ADDI	\$1 1	228		ADDI	\$1 1
176		EXCH	\$12 \$1	229		EXCH	\$3 \$1
177		ADDI	\$1 1	230		ADDI	\$1 1
178		EXCH	\$11 \$1	231		EXCH	\$6 \$1
179		ADDI	\$1 1	232		EXCH	\$2 \$1
180		EXCH	\$10 \$1	233		ADDI	\$1 -1
181		RR	\$9 1	234		ADD	\$7 \$3
182		ADDI	\$8 -1	235		ADDI	\$7 4
	•			'			

236		EXCH	\$8 \$7	294		SUB	\$10 \$3
237		ADDI	\$7 -4	295		EXCH	\$9 \$7
238		SUB	\$7 \$3	296	test_18:	BEQ	\$8 \$0
				290	_	DEQ	70 70
239		EXCH	\$9 \$6		test_false_20		
$^{240}$		XOR	\$8 \$9	297		XORI	\$8 1
$^{241}$		EXCH	\$9 \$6	298		ADD	\$10 \$3
242		ADD	\$7 \$3	299		ADDI	\$10 2
243		ADDI	\$7 4	300		EXCH	\$11 \$10
244		EXCH	\$8 \$7	301		ADDI	\$10 -2
245		ADDI	\$7 -4	302		SUB	\$10 \$3
246		SUB	\$7 \$3	303	cmp_top_30:	BNE	\$11 \$0
247	l setValue 4 bot:	BRA			cmp_bot_31		
	l_setValue_4_top			304	1 - 1 - 1	XORI	\$12 1
	_	DD 3			1 21		
248	l_insertNode_5_top:	BRA		305	cmp_bot_31:	BNE	\$11 \$0
	l_insertNode_5_bot				cmp_top_30		
$^{249}$		ADDI	\$1 1	306		EXCH	\$13 \$6
250		EXCH	\$2 \$1	307	cmp_top_32:	BEQ	\$13 \$0
251		EXCH	\$6 \$1		cmp_bot_33	_	
		ADDI	\$1 -1	308	Cmp_b0e_55	XORI	¢1/1 1
252					1 . 22		\$14 1
253		EXCH	\$7 \$1	309	cmp_bot_33:	BEQ	\$13 \$0
254		ADDI	\$1 -1		cmp_top_32		
255		EXCH	\$3 \$1	310		ANDX	\$15 \$12 \$14
256		ADDI	\$1 -1	311	f_top_34:	BEQ	\$15 \$0
257	l_insertNode_5:	SWAPBR			f bot 35	z	. =
				010	1_500_55	VODT	¢1 6 1
258		NEG	\$2	312		XORI	\$16 1
259		ADDI	\$1 1	313	f_bot_35:	BEQ	\$15 \$0
260		EXCH	\$3 \$1		f_top_34		
261		ADDI	\$1 1	314		XOR	\$9 \$16
262		EXCH	\$7 \$1	315	f_bot_35_i:	BEQ	\$15 \$0
				313		בבע	V13 V0
263		ADDI	\$1 1		f_top_34_i		***
264		EXCH	\$6 \$1	316		XORI	\$16 1
$^{265}$		EXCH	\$2 \$1	317	f_top_34_i:	BEQ	\$15 \$0
266		ADDI	\$1 -1		f_bot_35_i		
267		EXCH	\$9 \$7	318		ANDX	\$15 \$12 \$14
268		ADD	\$10 \$3	319	cmp_bot_33_i:	BEQ	\$13 \$0
		ADDI		313		בבע	V13 V0
269			\$10 4		cmp_top_32_i		
270		EXCH	\$11 \$10	320		XORI	\$14 1
271		ADDI	\$10 -4	321	cmp_top_32_i:	BEQ	\$13 \$0
272		SUB	\$10 \$3		cmp_bot_33_i		
273		XOR	\$12 \$9	322		EXCH	\$13 \$6
274		SUB	\$12 \$11	323	cmp_bot_31_i:	BNE	\$11 \$0
	+ 22.					DIVE	YII YU
275	cmp_top_22:	BGEZ	\$12 cmp_bot_		cmp_top_30_i		***
276		XORI	\$13 1	324		XORI	\$12 1
277	cmp_bot_23:	BGEZ	\$12 cmp_top_	23225	cmp_top_30_i:	BNE	\$11 \$0
278	f_top_24:	BEQ	\$13 \$0		cmp_bot_31_i		
	f_bot_25			326		ADD	\$10 \$3
279		XORI	\$14 1	327		ADDI	\$10 2
	f bot 25.						
280	f_bot_25:	BEQ	\$13 \$0	328		EXCH	\$11 \$10
	f_top_24			329		ADDI	\$10 -2
281		XOR	\$8 \$14	330		SUB	\$10 \$3
282	f_bot_25_i:	BEQ	\$13 \$0	331	test_26:	BEQ	\$9 \$0
	f_top_24_i				test_false_28	_	
283		XORI	\$14 1	332		XORI	\$9 1
	f + on 24 :						
284	f_top_24_i:	BEQ	\$13 \$0	333		ADD	\$10 \$3
	f_bot_25_i			334		ADDI	\$10 2
285	cmp_bot_23_i:	BGEZ	\$12	335		EXCH	\$11 \$10
	cmp_top_22_i			336		ADDI	\$10 -2
286		XORI	\$13 1	337		SUB	\$10 \$3
287	cmp_top_22_i:	BGEZ	\$12	338		EXCH	\$12 \$6
201			T ± 4		gwan 36.		
	cmp_bot_23_i		410 411	339	swap_36:	XOR	\$11 \$12
288		ADD	\$12 \$11	340		XOR	\$12 \$11
289		XOR	\$12 \$9	341		XOR	\$11 \$12
290		ADD	\$10 \$3	342		EXCH	\$12 \$6
291		ADDI	\$10 4	343		ADD	\$10 \$3
292		EXCH	\$11 \$10	344		ADDI	\$10 2
293		ADDI	\$10 -4	345		EXCH	\$11 \$10
293	I	.1001	710 1	949			Y11 Y10

i							
346		ADDI	\$10 -2		f_top_49		
347		SUB	\$10 \$3	396		XOR	\$9 \$13
348		XORI	\$9 1	397	f_bot_50_i:	BEQ	\$12 \$0
349	assert_true_27:	BRA	assert_29		f_top_49_i		
350	test_false_28:	BRA	test_26	398		XORI	\$13 1
351	assert_29:	BNE	\$9 \$0	399	f_top_49_i:	BEQ	\$12 \$0
	assert_true_27				f_bot_50_i		
352		ADD	\$10 \$3	400	cmp_bot_48_i:	BEQ	\$11 \$0
353		ADDI	\$10 2		cmp_top_47_i		
354		EXCH	\$11 \$10	401		XORI	\$12 1
355		ADDI	\$10 -2	402	cmp_top_47_i:	BEQ	\$11 \$0
356		SUB	\$10 \$3		cmp_bot_48_i		
357	cmp_top_37:	BEQ	\$11 \$0	403		ADD	\$10 \$3
	cmp_bot_38			404		ADDI	\$10 2
358		XORI	\$12 1	405		EXCH	\$11 \$10
359	cmp_bot_38:	BEQ	\$11 \$0	406		ADDI	\$10 -2
	cmp_top_37			407		SUB	\$10 \$3
360		EXCH	\$13 \$6	408	test_43:	BEQ	\$9 \$0
361	cmp_top_39:	BNE	\$13 \$0		test_false_45		
	cmp_bot_40			409		XORI	\$9 1
362		XORI	\$14 1	410		ADD	\$10 \$3
363	cmp_bot_40:	BNE	\$13 \$0	411		ADDI	\$10 2
	cmp_top_39			412		EXCH	\$11 \$10
364		ANDX	\$15 \$12 \$14	413		ADDI	\$10 -2
365	f_top_41:	BEQ	\$15 \$0	414		SUB	\$10 \$3
	f_bot_42			415		XOR	\$12 \$11
366		XORI	\$16 1	416	loadMetAdd_51:	EXCH	\$13 \$12
367	f_bot_42:	BEQ	\$15 \$0	417		ADDI	\$13 1
	f_top_41			418		EXCH	\$14 \$13
368		XOR	\$9 \$16	419		XOR	\$15 \$14
369	f_bot_42_i:	BEQ	\$15 \$0	420		EXCH	\$14 \$13
	f_top_41_i			421		ADDI	\$13 -1
370		XORI	\$16 1	422		EXCH	\$13 \$12
371	f_top_41_i:	BEQ	\$15 \$0	423		ADD	\$10 \$3
	f_bot_42_i			424		ADDI	\$10 2
372		ANDX	\$15 \$12 \$14	425		EXCH	\$11 \$10
373	cmp_bot_40_i:	BNE	\$13 \$0	426		ADDI	\$10 -2
	cmp_top_39_i			427		SUB	\$10 \$3
374		XORI	\$14 1	428		EXCH	\$3 \$1
375	cmp_top_39_i:	BNE	\$13 \$0	429		ADDI	\$1 -1
	cmp_bot_40_i			430		EXCH	\$6 \$1
376		EXCH	\$13 \$6	431		ADDI	\$1 -1
377	cmp_bot_38_i:	BEQ	\$11 \$0	432		EXCH	\$7 \$1
	cmp_top_37_i			433		ADDI	\$1 -1
378		XORI	\$12 1	434		EXCH	\$12 \$1
379	cmp_top_37_i:	BEQ	\$11 \$0	435		ADDI	\$1 -1
	cmp_bot_38_i		***	436		ADDI	\$15 -435
380		ADD	\$10 \$3	437	l_jmp_52:	SWAPBR	
381		ADDI	\$10 2	438		NEG	\$15
382		EXCH	\$11 \$10	439		ADDI	\$15 435
383		ADDI	\$10 -2	440		ADDI	\$1 1
384		SUB	\$10 \$3	441		EXCH	\$12 \$1
385		ADD	\$10 \$3	442		ADDI	\$1 1
386		ADDI	\$10 2	443		EXCH	\$7 \$1
387		EXCH	\$11 \$10	444		ADDI	\$1 1
388		ADDI	\$10 -2	445		EXCH	\$6 \$1
389	4.5	SUB	\$10 \$3	446		ADDI	\$1 1
390	cmp_top_47:	BEQ	\$11 \$0	447		EXCH	\$3 \$1
	cmp_bot_48	wo==	<b>610</b> 1	448		ADD	\$10 \$3
391	1 4 40	XORI	\$12 1	449		ADDI	\$10 2
392	cmp_bot_48:	BEQ	\$11 \$0	450		EXCH	\$11 \$10
	cmp_top_47		410 40	451		ADDI	\$10 -2
393		BEQ	\$12 \$0	452		SUB	\$10 \$3
	f_bot_50		***	453		EXCH	\$13 \$12
394	C 1	XORI	\$13 1	454		ADDI	\$13 1
395	f_bot_50:	BEQ	\$12 \$0	455		EXCH	\$14 \$13

456		XOR	\$15 \$14	509	f_top_65:	BEQ	\$15 \$0
457		EXCH	\$14 \$13		f_bot_66	2	1 1-
					1_000_00		0161
458		ADDI	\$13 -1	510		XORI	\$16 1
459	loadMetAdd_51_i:	EXCH	\$13 \$12	511	f_bot_66:	BEQ	\$15 \$0
460		XOR	\$12 \$11		f_top_65		
461		ADD	\$10 \$3	512		XOR	\$9 \$16
462		ADDI	\$10 2	513	f_bot_66_i:	BEQ	\$15 \$0
463		EXCH	\$11 \$10		f_top_65_i		
464		ADDI	\$10 -2	514		XORI	\$16 1
465		SUB	\$10 \$3	515	f_top_65_i:	BEQ	\$15 \$0
				313	_	PFŐ	512 50
466		XORI	\$9 1		f_bot_66_i		
467	assert_true_44:	BRA	assert_46	516		ANDX	\$15 \$12 \$14
468	test_false_45:	BRA	test_43	517	cmp_bot_64_i:	BEQ	\$13 \$0
469	assert 46:	BNE	\$9 \$0		cmp_top_63_i	2	1 1-
409	_	DNE	79 70		Cmp_cop_63_1		6141
	assert_true_44			518		XORI	\$14 1
470		ADD	\$10 \$3	519	cmp_top_63_i:	BEQ	\$13 \$0
471		ADDI	\$10 2		cmp_bot_64_i		
472		EXCH	\$11 \$10	520		EXCH	\$13 \$6
					1		
473		ADDI	\$10 -2	521	cmp_bot_62_i:	BNE	\$11 \$0
474		SUB	\$10 \$3		cmp_top_61_i		
475	cmp_top_53:	BEQ	\$11 \$0	522		XORI	\$12 1
		~			cmp_top_61_i:	BNE	\$11 \$0
	cmp_bot_54	V05-	ć10 1	523	1	DIE	ATT A0
476		XORI	\$12 1		cmp_bot_62_i		
477	cmp_bot_54:	BEQ	\$11 \$0	524		ADD	\$10 \$3
	cmp_top_53			525		ADDI	\$10 3
478	f_top_55:	BEQ	\$12 \$0	526		EXCH	\$11 \$10
410	_	PEQ	717 70				
	f_bot_56			527		ADDI	\$10 -3
479		XORI	\$13 1	528		SUB	\$10 \$3
480	f_bot_56:	BEQ	\$12 \$0	529	test_57:	BEQ	\$9 \$0
100		z	712 70	020	_	2	73 70
	f_top_55				test_false_59		
481		XOR	\$9 \$13	530		XORI	\$9 1
482	f_bot_56_i:	BEQ	\$12 \$0	531		ADD	\$10 \$3
	f_top_55_i			532		ADDI	\$10 3
400	1_00P_00_1	XORI	¢12 1			EXCH	
483			\$13 1	533			\$11 \$10
484	f_top_55_i:	BEQ	\$12 \$0	534		ADDI	\$10 -3
	f_bot_56_i			535		SUB	\$10 \$3
485	cmp_bot_54_i:	BEQ	\$11 \$0	536		EXCH	\$12 \$6
100	_	z	7-1- 70		a 67.	XOR	
	cmp_top_53_i			537	swap_67:		\$11 \$12
486		XORI	\$12 1	538		XOR	\$12 \$11
487	cmp_top_53_i:	BEQ	\$11 \$0	539		XOR	\$11 \$12
	cmp_bot_54_i			540		EXCH	\$12 \$6
488	1 - 1 - 1 - 1	ADD	\$10 \$3	541		ADD	\$10 \$3
489		ADDI	\$10 2	542		ADDI	\$10 3
490		EXCH	\$11 \$10	543		EXCH	\$11 \$10
491		ADDI	\$10 -2	544		ADDI	\$10 -3
492		SUB	\$10 \$3	545		SUB	\$10 \$3
				546			\$9 1
493		XORI	\$8 1			XORI	
494		BRA	assert_21	547	assert_true_58:	BRA	assert_60
495	test_false_20:	BRA	test_18	548	test_false_59:	BRA	test_57
496	_	ADD	\$10 \$3	549	assert 60:	BNE	\$9 \$0
				J-±3	_		T - T -
497		ADDI	\$10 3		assert_true_58		
498		EXCH	\$11 \$10	550		ADD	\$10 \$3
499		ADDI	\$10 -3	551		ADDI	\$10 3
500		SUB	\$10 \$3	552		EXCH	\$11 \$10
501	cmp top 61.	BNE	\$11 \$0	553		ADDI	\$10 -3
501	cmp_top_61:	DNE	SIT SO				
	cmp_bot_62			554		SUB	\$10 \$3
502		XORI	\$12 1	555	cmp_top_68:	BEQ	\$11 \$0
503	cmp_bot_62:	BNE	\$11 \$0		cmp_bot_69		
-00	_		. ==	550		XORI	\$12 1
	cmp_top_61	<b></b>	612 66	556	1 50		
504		EXCH	\$13 \$6	557	cmp_bot_69:	BEQ	\$11 \$0
505	cmp_top_63:	BEQ	\$13 \$0		cmp_top_68		
	cmp_bot_64			558		EXCH	\$13 \$6
506	1 - 1 - 1 - 1	XORI	\$14 1	559	cmp_top_70:	BNE	\$13 \$0
500		VOLT		559	1	DRE	4T2 40
	l + C / -	DEC	Ċ12 ĊC				
507	cmp_bot_64:	BEQ	\$13 \$0		cmp_bot_71		
507	<pre>cmp_bot_64:    cmp_top_63</pre>	BEQ	\$13 \$0	560	Cmp_bot_/1	XORI	\$14 1
507 508		<b>BEQ</b> ANDX	\$13 \$0 \$15 \$12 \$14		cmp_bot_71:	XORI BNE	\$14 1 \$13 \$0

ĺ	amp + ap 70			caol		EVCU	¢11 ¢10
500	cmp_top_70	ANDX	¢1E ¢10 ¢14	610 611		EXCH ADDI	\$11 \$10 \$10 -3
562 563	f_top_72:	BEQ	\$15 \$12 \$14 \$15 \$0	612		SUB	\$10 \$3
303	f_bot_73	PEQ	310 AO	613		XOR	\$12 \$11
564	1_000_73	XORI	\$16 1	614	loadMetAdd_82:	EXCH	\$13 \$12
	f hot 73.	BEQ	\$15 \$0		TOAUMETAUG_02.	ADDI	\$13 1
565	f_bot_73:	PEQ	717 70	615 616		EXCH	\$14 \$13
566	f_top_72	XOR	\$9 \$16			XOR	
	f bot 72 :.			617			\$15 \$14
567	f_bot_73_i:	BEQ	\$15 \$0	618		EXCH	\$14 \$13
	f_top_72_i	VODT	Ċ1 C 1	619		ADDI	\$13 -1
568	£ 70 i.	XORI	\$16 1	620		EXCH	\$13 \$12
569	f_top_72_i:	BEQ	\$15 \$0	621		ADD	\$10 \$3
	f_bot_73_i	7.110.17	615 610 614	622		ADDI	\$10 3
570	1 71	ANDX	\$15 \$12 \$14	623		EXCH	\$11 \$10
571	cmp_bot_71_i:	BNE	\$13 \$0	624		ADDI	\$10 -3
	cmp_top_70_i			625		SUB	\$10 \$3
572		XORI	\$14 1	626		EXCH	\$3 \$1
573	cmp_top_70_i:	BNE	\$13 \$0	627		ADDI	\$1 -1
	cmp_bot_71_i			628		EXCH	\$6 \$1
574		EXCH	\$13 \$6	629		ADDI	\$1 -1
575	cmp_bot_69_i:	BEQ	\$11 \$0	630		EXCH	\$7 \$1
	cmp_top_68_i			631		ADDI	\$1 -1
576		XORI	\$12 1	632		EXCH	\$12 \$1
577	cmp_top_68_i:	BEQ	\$11 \$0	633		ADDI	\$1 -1
	cmp_bot_69_i			634		ADDI	\$15 -633
578		ADD	\$10 \$3	635	1_jmp_83:	SWAPBR	
579		ADDI	\$10 3	636		NEG	\$15
580		EXCH	\$11 \$10	637		ADDI	\$15 633
581		ADDI	\$10 -3	638		ADDI	\$1 1
582		SUB	\$10 \$3	639		EXCH	\$12 \$1
583		ADD	\$10 \$3	640		ADDI	\$1 1
584		ADDI	\$10 3	641		EXCH	\$7 \$1
585		EXCH	\$11 \$10	642		ADDI	\$1 1
586		ADDI	\$10 -3	643		EXCH	\$6 \$1
587		SUB	\$10 \$3	644		ADDI	\$1 1
588	cmp_top_78:	BEQ	\$11 \$0	645		EXCH	\$3 \$1
	cmp_bot_79			646		ADD	\$10 \$3
589		XORI	\$12 1	647		ADDI	\$10 3
590	cmp_bot_79:	BEQ	\$11 \$0	648		EXCH	\$11 \$10
	cmp_top_78			649		ADDI	\$10 -3
591	f_top_80:	BEQ	\$12 \$0	650		SUB	\$10 \$3
	f_bot_81			651		EXCH	\$13 \$12
592		XORI	\$13 1	652		ADDI	\$13 1
593	f_bot_81:	BEQ	\$12 \$0	653		EXCH	\$14 \$13
	f_top_80			654		XOR	\$15 \$14
594		XOR	\$9 \$13	655		EXCH	\$14 \$13
595	f_bot_81_i:	BEQ	\$12 \$0	656		ADDI	\$13 -1
	f_top_80_i			657	loadMetAdd_82_i:	EXCH	\$13 \$12
596		XORI	\$13 1	658		XOR	\$12 \$11
597	f_top_80_i:	BEQ	\$12 \$0	659		ADD	\$10 \$3
	f_bot_81_i			660		ADDI	\$10 3
598	cmp_bot_79_i:	BEQ	\$11 \$0	661		EXCH	\$11 \$10
	cmp_top_78_i			662		ADDI	\$10 -3
599		XORI	\$12 1	663		SUB	\$10 \$3
600	cmp_top_78_i:	BEQ	\$11 \$0	664		XORI	\$9 1
	cmp_bot_79_i			665	assert_true_75:	BRA	assert_77
601		ADD	\$10 \$3	666	test_false_76:	BRA	test_74
602		ADDI	\$10 3	667	assert_77:	BNE	\$9 \$0
603		EXCH	\$11 \$10		assert_true_75		
604		ADDI	\$10 -3	668		ADD	\$10 \$3
605		SUB	\$10 \$3	669		ADDI	\$10 3
606	test_74:	BEQ	\$9 \$0	670		EXCH	\$11 \$10
	test_false_76			671		ADDI	\$10 -3
607		XORI	\$9 1	672		SUB	\$10 \$3
608		ADD	\$10 \$3	673	cmp_top_84:	BEQ	\$11 \$0
609		ADDI	\$10 3		cmp_bot_85	•	
'				'	= =		

674		XORI	\$12	1	725		EXCH	\$6 \$1
675	cmp_bot_85:	BEQ	\$11	\$0	726		ADDI	\$1 -1
	cmp_top_84				727		EXCH	\$3 \$1
676	f_top_86:	BEQ	\$12	¢ O	728		ADDI	\$1 -1
076	=	PFÕ	7 I Z	Ş U				
	f_bot_87				729	l_getSum_6:	SWAPBR	\$2
677		XORI	\$13	1	730		NEG	\$2
678	f_bot_87:	BEO	\$12	\$0	731		ADDI	\$1 1
	f_top_86			1 -	732		EXCH	\$3 \$1
	1_cop_00		<b>A</b> O	44.0				
679		XOR	\$9	\$13	733		ADDI	\$1 1
680	f_bot_87_i:	BEQ	\$12	\$0	734		EXCH	\$6 \$1
	f_top_86_i				735		EXCH	\$2 \$1
681	_ :	XORI	\$13	1	736		ADDI	\$1 -1
	6 1 06 .							
682	f_top_86_i:	BEQ	\$12	\$0	737		EXCH	\$7 \$6
	f_bot_87_i				738		ADD	\$8 \$3
683	cmp_bot_85_i:	BEQ	\$11	\$0	739		ADDI	\$8 4
	cmp_top_84_i				740		EXCH	\$9 \$8
60.4	05_005_01_1	VODT	¢10	1			ADDI	
684		XORI	\$12		741			\$8 -4
685	cmp_top_84_i:	BEQ	\$11	\$0	742		SUB	\$8 \$3
	cmp_bot_85_i				743		ADD	\$7 \$9
686		ADD	\$10	\$3	744		ADD	\$8 \$3
687		ADDI	\$10		745		ADDI	\$8 4
688		EXCH		\$10	746		EXCH	\$9 \$8
689		ADDI	\$10	-3	747		ADDI	\$8 -4
690		SUB	\$10	\$3	748		SUB	\$8 \$3
691	assert_21:	BNE	\$8	\$0	749		EXCH	\$7 \$6
	assert_true_19				750		ADD	\$8 \$3
	assert_true_r9		<b>A</b> O					
692		EXCH	\$9		751		ADDI	\$8 2
693		ADD	\$10	\$3	752		EXCH	\$9 \$8
694		ADDI	\$10	4	753		ADDI	\$8 -2
695		EXCH	\$11	\$10	754		SUB	\$8 \$3
		ADDI				amp top 06.		
696			\$10		755	cmp_top_96:	BEQ	\$9 \$0
697		SUB	\$10	\$3		cmp_bot_97		
698		XOR	\$12	\$9	756		XORI	\$10 1
699		SUB	\$12	\$11	757	cmp_bot_97:	BEQ	\$9 \$0
700	cmp_top_88:	BGEZ		cmp_bot_		cmp_top_96	~	
	cmp_cop_oo.			_	- 1		DE0	610 60
701		XORI	\$13		758	f_top_98:	BEQ	\$10 \$0
702	cmp_bot_89:	BGEZ	\$12	cmp_top_	88	f_bot_99		
703	f_top_90:	BEQ	\$13	\$0	759		XORI	\$11 1
	f_bot_91				760	f_bot_99:	BEQ	\$10 \$0
704		XORI	\$14	1		f_top_98	~	
	6.101					1_000_50		00 011
705	f_bot_91:	BEQ	\$13	\$0	761		XOR	\$7 \$11
	f_top_90				762	f_bot_99_i:	BEQ	\$10 \$0
706		XOR	\$8	\$14		f_top_98_i		
707	f_bot_91_i:	BEQ	\$13	\$0	763		XORI	\$11 1
	f_top_90_i	~			764	f_top_98_i:	BEO	\$10 \$0
700	cob_>o_+	VODT	ć 1 A	1	104	_	ההה	710 70
708	5	XORI	\$14			f_bot_99_i		40 +-
709	f_top_90_i:	BEQ	\$13	ŞU	765		BEQ	\$9 \$0
	f_bot_91_i					cmp_top_96_i		
710	cmp_bot_89_i:	BGEZ	\$12		766		XORI	\$10 1
-	cmp_top_88_i				767	cmp_top_96_i:	BEQ	\$9 \$0
711	op_cop_oo_+	YOPT	¢12	1			z	, , , , ,
711		XORI	\$13	Τ		cmp_bot_97_i		40 40
712	cmp_top_88_i:	BGEZ	\$12		768		ADD	\$8 \$3
	cmp_bot_89_i				769		ADDI	\$8 2
713		ADD	\$12	\$11	770		EXCH	\$9 \$8
714		XOR	\$12		771		ADDI	\$8 -2
715		ADD	\$10		772		SUB	\$8 \$3
716		ADDI	\$10		773	_	BEQ	\$7 \$0
717		EXCH	\$11	\$10		test_false_94		
718		ADDI	\$10	-4	774		XORI	\$7 1
719		SUB	\$10		- 1		ADD	\$8 \$3
					775			
720		EXCH	\$9	<b>⇒</b> /	776		ADDI	\$8 2
721	<pre>l_insertNode_5_bot:</pre>	BRA			777		EXCH	\$9 \$8
	l_insertNode_5_top				778		ADDI	\$8 -2
722	l_getSum_6_top:	BRA	1 0	etSum_6_b			SUB	\$8 \$3
723		ADDI	\$1		780		XOR	\$10 \$9
						1		
724		EXCH	\$2	ÞΤ	781	loadMetAdd_100:	EXCH	\$11 \$10

782	2	ADDI	\$11 2	843	f_bot_105_i:	BEQ	\$10 \$0
783		EXCH	\$12 \$11	0.00	f_top_104_i		1 1-
1					1_cop_104_1		
784	2	KOR	\$13 \$12	844		XORI	\$11 1
785	F	EXCH	\$12 \$11	845	f_top_104_i:	BEQ	\$10 \$0
786	I	ADDI	\$11 -2		f_bot_105_i		
787		EXCH	\$11 \$10	846	cmp_bot_103_i:	BEQ	\$9 \$0
1				840	_	PFŐ	79 70
788	I	ADD	\$8 \$3		cmp_top_102_i		
789	I	ADDI	\$8 2	847		XORI	\$10 1
790	F	EXCH	\$9 \$8	848	cmp_top_102_i:	BEQ	\$9 \$0
		ADDI		0.10		z	73 70
791			\$8 -2		cmp_bot_103_i		
792	\$	SUB	\$8 \$3	849		ADD	\$8 \$3
793	E	EXCH	\$3 \$1	850		ADDI	\$8 2
794	z	ADDI	\$1 -1	851		EXCH	\$9 \$8
		EXCH					
795			\$6 \$1	852		ADDI	\$8 -2
796	I	ADDI	\$1 -1	853		SUB	\$8 \$3
797	E	EXCH	\$10 \$1	854		ADD	\$8 \$3
798	z	ADDI	\$1 -1	855		ADDI	\$8 3
799		ADDI	\$13 -798	856		EXCH	\$9 \$8
800	l_jmp_101:	SWAPBR	\$13	857		ADDI	\$8 -3
801	ı	NEG	\$13	858		SUB	\$8 \$3
802	7	ADDI	\$13 798	859	cmp_top_110:	BEQ	\$9 \$0
				000	_ =	DEQ	<b>Ψ 9 Ψ 0</b>
803		ADDI	\$1 1		cmp_bot_111		
804	E	EXCH	\$10 \$1	860		XORI	\$10 1
805	I	ADDI	\$1 1	861	cmp_bot_111:	BEQ	\$9 \$0
		EXCH	\$6 \$1			~	
806					cmp_top_110		
807	I	ADDI	\$1 1	862	f_top_112:	BEQ	\$10 \$0
808	E	EXCH	\$3 \$1		f_bot_113		
809	2	ADD	\$8 \$3	863		XORI	\$11 1
		ADDI	\$8 2		f bo+ 112.		
810				864	f_bot_113:	BEQ	\$10 \$0
811	E	EXCH	\$9 \$8		f_top_112		
812	I	ADDI	\$8 -2	865		XOR	\$7 \$11
813	g	SUB	\$8 \$3	866	f_bot_113_i:	BEQ	\$10 \$0
				000		222	410 40
814		EXCH	\$11 \$10		f_top_112_i		
815	I	ADDI	\$11 2	867		XORI	\$11 1
816	E	EXCH	\$12 \$11	868	f_top_112_i:	BEQ	\$10 \$0
817	3	KOR	\$13 \$12		f_bot_113_i	_	
							40 40
818	<u> </u>	EXCH	\$12 \$11	869	cmp_bot_111_i:	BEQ	\$9 \$0
819	I	ADDI	\$11 -2		cmp_top_110_i		
820	loadMetAdd_100_i: E	EXCH	\$11 \$10	870		XORI	\$10 1
821		KOR	\$10 \$9	871	cmp_top_110_i:	BEQ	\$9 \$0
				0/1	= =	PFŐ	79 70
822	I	ADD	\$8 \$3		cmp_bot_111_i		
823	I	ADDI	\$8 2	872		ADD	\$8 \$3
824	F	EXCH	\$9 \$8	873		ADDI	\$8 3
1							\$9 \$8
825		ADDI	\$8 -2	874		EXCH	
826	\$	SUB	\$8 \$3	875		ADDI	\$8 -3
827	2	KORI	\$7 1	876		SUB	\$8 \$3
828	assert_true_93:	BRA	assert_95	877	test_106:	BEQ	\$7 \$0
829		BRA	test_92		test_false_108	-	
					cesc_rarse_ruo	we==	67 1
830	_	BNE	\$7 \$0	878		XORI	\$7 1
	assert_true_93			879		ADD	\$8 \$3
831	7	ADD	\$8 \$3	880		ADDI	\$8 3
1							
832		ADDI	\$8 2	881		EXCH	\$9 \$8
833	E	EXCH	\$9 \$8	882		ADDI	\$8 -3
834	I	ADDI	\$8 -2	883		SUB	\$8 \$3
835	۶	SUB	\$8 \$3	884		XOR	\$10 \$9
					loadMo+7 dd 114.		
836		BEQ	\$9 \$0	885	loadMetAdd_114:	EXCH	\$11 \$10
	cmp_bot_103			886		ADDI	\$11 2
837	2	KORI	\$10 1	887		EXCH	\$12 \$11
838		BEQ	\$9 \$0	888		XOR	\$13 \$12
000	<del>-</del>	2	T - Y -				
	cmp_top_102			889		EXCH	\$12 \$11
839	f_top_104:	BEQ	\$10 \$0	890		ADDI	\$11 -2
ļ	f_bot_105			891		EXCH	\$11 \$10
840		KORI	\$11 1	892		ADD	\$8 \$3
1							
841		BEQ	\$10 \$0	893		ADDI	\$8 3
	f_top_104			894		EXCH	\$9 \$8
842	>	KOR	\$7 \$11	895		ADDI	\$8 -3
,				,			

896		SUB	\$8 \$3	953		ADD	\$8 \$3
897		EXCH	\$3 \$1	954		ADDI	\$8 3
1							
898		ADDI	\$1 -1	955		EXCH	\$9 \$8
899		EXCH	\$6 \$1	956		ADDI	\$8 -3
900		ADDI	\$1 -1	957		SUB	\$8 \$3
901		EXCH	\$10 \$1	958	l_getSum_6_bot:	BRA	l_getSum_6_top
1					_		
902		ADDI	\$1 -1	959	l_mirror_7_top:	BRA	l_mirror_7_bot
903		ADDI	\$13 -902	960		ADDI	\$1 1
904	l_jmp_115:	SWAPBR	\$13	961		EXCH	\$2 \$1
905		NEG	\$13	962		EXCH	\$3 \$1
906		ADDI	\$13 902	963		ADDI	\$1 -1
907		ADDI	\$1 1	964	l_mirror_7:	SWAPBR	\$2
908		EXCH	\$10 \$1	965		NEG	\$2
909		ADDI	\$1 1	966		ADDI	\$1 1
910		EXCH	\$6 \$1	967		EXCH	\$3 \$1
911		ADDI	\$1 1	968		EXCH	\$2 \$1
912		EXCH	\$3 \$1	969		ADDI	\$1 -1
		ADD	\$8 \$3			ADD	\$6 \$3
913				970			
914		ADDI	\$8 3	971		ADDI	\$6 2
915		EXCH	\$9 \$8	972		EXCH	\$7 \$6
916		ADDI	\$8 -3	973		ADDI	\$6 -2
917		SUB	\$8 \$3	974		SUB	\$6 \$3
918		EXCH	\$11 \$10	975		ADD	\$8 \$3
919		ADDI	\$11 2	976		ADDI	\$8 3
920		EXCH	\$12 \$11	977		EXCH	\$9 \$8
921		XOR	\$13 \$12	978		ADDI	\$8 -3
922		EXCH	\$12 \$11	979		SUB	\$8 \$3
923		ADDI	\$11 -2	980	swap_120:	XOR	\$7 \$9
924	loadMetAdd 114 i:	EXCH	\$11 \$10	981	<del>-</del> -	XOR	\$9 \$7
	10441001144_11						
925		XOR	\$10 \$9	982		XOR	\$7 \$9
926		ADD	\$8 \$3	983		ADD	\$8 \$3
927		ADDI	\$8 3	984		ADDI	\$8 3
928		EXCH	\$9 \$8	985		EXCH	\$9 \$8
929		ADDI	\$8 -3	986		ADDI	\$8 -3
930		SUB	\$8 \$3	987		SUB	\$8 \$3
931		XORI	\$7 1	988		ADD	\$6 \$3
	2000t tous 107.					ADDI	
932	assert_true_107:	BRA	assert_109	989			\$6 2
933	test_false_108:	BRA	test_106	990		EXCH	\$7 \$6
934	assert_109:	BNE	\$7 \$0	991		ADDI	\$6 -2
	assert_true_107			992		SUB	\$6 \$3
005	abbere_erae_ro,	* D.D.	¢0 ¢2				
935		ADD	\$8 \$3	993		ADD	\$7 \$3
936		ADDI	\$8 3	994		ADDI	\$7 2
937		EXCH	\$9 \$8	995		EXCH	\$8 \$7
938		ADDI	\$8 -3	996		ADDI	\$7 -2
939		SUB	\$8 \$3	997		SUB	\$7 \$3
940	cmp_top_116:	BEQ	\$9 \$0	998	cmp_top_125:	BNE	\$8 \$0
	cmp_bot_117				cmp_bot_126		
941		XORI	\$10 1	999		XORI	\$9 1
	cmp bot 117:	BEQ	\$9 \$0	1000	cmp bot 126:	BNE	\$8 \$0
942	1	הבה	∪ د و ب	1000		DNE	γυ γυ
	cmp_top_116				cmp_top_125		
943	f_top_118:	BEQ	\$10 \$0	1001	f_top_127:	BEQ	\$9 \$0
	f_bot_119			i	f_bot_128		
044		VODT	ė11 1	1000	1_200_120	VORT	\$10 1
944		XORI	\$11 1	1002		XORI	
945	f_bot_119:	BEQ	\$10 \$0	1003	f_bot_128:	BEQ	\$9 \$0
	f_top_118				f_top_127		
946	— <u>*</u> —	XOR	\$7 \$11	1004	— <u>*</u> —	XOR	\$6 \$10
	5.1 . 110 .				5.1 . 100 .		
947	f_bot_119_i:	BEQ	\$10 \$0	1005		BEQ	\$9 \$0
	f_top_118_i				f_top_127_i		
948		XORI	\$11 1	1006		XORI	\$10 1
	f_top_118_i:	BEQ	\$10 \$0	1007	f_top_127_i:	BEQ	\$9 \$0
949		25	7±0 YU	1007	<del>-</del>	222	Y > Y U
	f_bot_119_i				f_bot_128_i		
950	cmp_bot_117_i:	BEQ	\$9 \$0	1008	cmp_bot_126_i:	BNE	\$8 \$0
	cmp_top_116_i			ļ	cmp_top_125_i		
951	- 11	XORI	\$10 1	1009	1 _ 1 _ 1	XORI	\$9 1
	116				105		
952	cmp_top_116_i:	BEQ	\$9 \$0	1010	cmp_top_125_i:	BNE	\$8 \$0
	cmp_bot_117_i				cmp_bot_126_i		
,				'			

1011		ADD	\$7 \$3	1075	cmp_top_131:	BNE	\$8 \$0
1012		ADDI	\$7 2		cmp_bot_132		
1013		EXCH	\$8 \$7	1076		XORI	\$9 1
1014		ADDI	\$7 -2	1077	cmp_bot_132:	BNE	\$8 \$0
1015		SUB	\$7 \$3		cmp_top_131		
1016	test_121:	BEQ	\$6 \$0	1078	f_top_133:	BEQ	\$9 \$0
	test_false_123		1 - 1 -		f_bot_134	2	1 - 1 -
1017	0000_10100_120	XORI	\$6 1	1079	1_2000_101	XORI	\$10 1
					f bot 124.		
1018	100	XORI	\$6 1	1080	f_bot_134:	BEQ	\$9 \$0
1019	assert_true_122:	BRA	assert_124		f_top_133		
1020	test_false_123:	BRA	test_121	1081		XOR	\$6 \$10
1021		ADD	\$7 \$3	1082	f_bot_134_i:	BEQ	\$9 \$0
1022		ADDI	\$7 2		f_top_133_i		
1023		EXCH	\$8 \$7	1083		XORI	\$10 1
1024		ADDI	\$7 -2	1084	f_top_133_i:	BEQ	\$9 \$0
1025		SUB	\$7 \$3		f_bot_134_i		
1026		XOR	\$9 \$8	1085	cmp_bot_132_i:	BNE	\$8 \$0
1027	loadMetAdd_129:	EXCH	\$10 \$9		cmp_top_131_i		
1028	104411001144_12).	ADDI	\$10 3	1086	omb_00b_101_1	XORI	\$9 1
1029		EXCH	\$11 \$10	1087	cmp_top_131_i:	BNE	\$8 \$0
				1007	= =	DNE	70 70
1030		XOR	\$12 \$11		cmp_bot_132_i		47 40
1031		EXCH	\$11 \$10	1088		ADD	\$7 \$3
1032		ADDI	\$10 -3	1089		ADDI	\$7 2
1033		EXCH	\$10 \$9	1090		EXCH	\$8 \$7
1034		ADD	\$7 \$3	1091		ADDI	\$7 -2
1035		ADDI	\$7 2	1092		SUB	\$7 \$3
1036		EXCH	\$8 \$7	1093		ADD	\$7 \$3
1037		ADDI	\$7 -2	1094		ADDI	\$7 3
1038		SUB	\$7 \$3	1095		EXCH	\$8 \$7
1039		EXCH	\$3 \$1	1096		ADDI	\$7 -3
1040		ADDI	\$1 -1	1097		SUB	\$7 \$3
1040		EXCH	\$9 \$1	1098	cmp_top_139:	BNE	\$8 \$0
				1098	_ = =	DIVE	70 70
1042		ADDI	\$1 -1		cmp_bot_140		00 1
1043		ADDI	\$12 -1042	1099		XORI	\$9 1
	1_jmp_130:	ADDI SWAPBR	\$12 -1042 \$12	1099 1100	cmp_bot_140:	XORI BNE	\$9 1 \$8 \$0
1043	1_jmp_130:	ADDI	\$12 -1042				
1043 1044	1_jmp_130:	ADDI SWAPBR	\$12 -1042 \$12		cmp_bot_140:		
1043 1044 1045	1_jmp_130:	ADDI SWAPBR NEG	\$12 -1042 \$12 \$12	1100	cmp_bot_140: cmp_top_139	BNE	\$8 \$0
1043 1044 1045 1046	1_jmp_130:	ADDI SWAPBR NEG ADDI	\$12 -1042 \$12 \$12 \$12 1042	1100	<pre>cmp_bot_140:      cmp_top_139 f_top_141:</pre>	BNE	\$8 \$0
1043 1044 1045 1046 1047	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1	1100 1101	<pre>cmp_bot_140:      cmp_top_139 f_top_141:</pre>	BNE BEQ	\$8 \$0
1043 1044 1045 1046 1047 1048 1049	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1	1100 1101 1102	<pre>cmp_bot_140:     cmp_top_139 f_top_141:     f_bot_142 f_bot_142:</pre>	BNE BEQ XORI	\$8 \$0 \$9 \$0 \$10 1
1043 1044 1045 1046 1047 1048 1049	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1	1100 1101 1102 1103	<pre>cmp_bot_140:     cmp_top_139 f_top_141:     f_bot_142</pre>	BNE BEQ XORI BEQ	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0
1043 1044 1045 1046 1047 1048 1049 1050 1051	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3	1100 1101 1102 1103 1104	<pre>cmp_bot_140:     cmp_top_139 f_top_141:     f_bot_142  f_bot_142:     f_top_141</pre>	BNE BEQ XORI BEQ XOR	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10
1043 1044 1045 1046 1047 1048 1049 1050 1051 1052	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADD ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2	1100 1101 1102 1103	<pre>cmp_bot_140:     cmp_top_139 f_top_141:     f_bot_142  f_bot_142:     f_top_141  f_bot_142_i:</pre>	BNE BEQ XORI BEQ	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0
1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADD ADDI EXCH	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7	1100 1101 1102 1103 1104 1105	<pre>cmp_bot_140:     cmp_top_139 f_top_141:     f_bot_142  f_bot_142:     f_top_141</pre>	BNE BEQ XORI BEQ XOR BEQ	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0
1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADD ADDI EXCH ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2	1100 1101 1102 1103 1104 1105	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XOR	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1
1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055	1_jmp_130:	ADDI SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADD ADDI EXCH ADDI SUB	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3	1100 1101 1102 1103 1104 1105	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0
1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADD ADDI EXCH ADD SUB EXCH	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9	1100 1101 1102 1103 1104 1105 1106 1107	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3	1100 1101 1102 1103 1104 1105	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XOR	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1
1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADD ADDI EXCH ADD SUB EXCH	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9	1100 1101 1102 1103 1104 1105 1106 1107	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3	1100 1101 1102 1103 1104 1105 1106 1107	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI SUB EXCH ADDI EXCH	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10	1100 1101 1102 1103 1104 1105 1106 1107	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ BNE	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059	1_jmp_130:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11	1100 1101 1102 1103 1104 1105 1106 1107 1108	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ XORI BEQ XORI	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$0
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059	<pre>1_jmp_130: loadMetAdd 129 i:</pre>	ADDI SWAPBR NEG ADDI EXCH ADDI EXCH ADD ADDI EXCH ADDI SUB EXCH ADDI SUB EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10	1100 1101 1102 1103 1104 1105 1106 1107 1108	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ XORI BEQ XORI	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$0
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062		ADDI SWAPBR NEG ADDI EXCH ADDI EXCH ADD ADDI EXCH EXCH EXCH	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$10 \$9	1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ XORI BEQ BNE XORI BNE	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$0 \$8 \$0 \$9 \$0
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063		ADDI SWAPBR NEG ADDI EXCH XOR	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$9 \$9 \$9 \$9	1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1111	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ XORI BEQ BNE ADD ADDI	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 3
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064		ADDI SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 \$9 \$10 \$1 \$11 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$10 \$9 \$9 \$8 \$7 \$3	1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ BNE XORI BNE ADD ADDI EXCH	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 3 \$8 \$7
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065		ADDI SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI ADDI EXCH ADDI ADDI EXCH ADDI ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 \$9 \$10 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$10 \$9 \$9 \$8 \$7 \$3 \$7 \$3	1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ BNE XORI BNE ADD ADDI EXCH ADDI	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 3 \$8 \$7 \$7 -3
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1060 1061 1062 1063 1064 1065 1066		ADDI SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH EXCH EXCH EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH EXCH EXCH	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$10 \$-3 \$10 \$9 \$9 \$8 \$7 \$3 \$7 \$3 \$10 \$9 \$10 \$9 \$10 \$3 \$11 \$10 \$10 \$10 \$12 \$11 \$11 \$10 \$10 \$10	1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XORI BEQ XORI BEQ BNE ADD ADDI EXCH ADDI SUB	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 3 \$8 \$7 \$7 -3 \$7 \$3
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1060 1061 1062 1063 1064 1065 1066 1067		ADDI SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$12 \$11 \$11 \$10 \$10 \$9 \$9 \$8 \$7 \$3 \$7 \$2 \$8 \$7	1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ BNE XORI BNE ADD ADDI EXCH ADDI	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 3 \$8 \$7 \$7 -3
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1060 1061 1062 1063 1064 1065 1066 1067 1068	loadMetAdd_129_i:	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH SUB	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$10 \$9 \$9 \$8 \$7 \$3 \$10 \$9	1100 1101 1102 1103 1104 1105 1106 1107 1108 1110 1111 1112 1113 1114 1115 1116	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ ADRI BNE ADRI ADRI ADDI EXCH ADDI SUB BEQ	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 3 \$8 \$7 \$7 -3 \$7 \$3 \$6 \$0
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1060 1061 1062 1063 1064 1065 1066 1067	<pre>loadMetAdd_129_i: assert_124:</pre>	ADDI SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$12 \$11 \$11 \$10 \$10 \$9 \$9 \$8 \$7 \$3 \$7 \$2 \$8 \$7	1100 1101 1102 1103 1104 1105 1106 1107 1108 1110 1111 1112 1113 1114 1115 1116	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ BNE XORI BNE ADD ADDI EXCH ADDI SUB BEQ XORI	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 3 \$8 \$7 \$7 -3 \$7 \$3 \$6 \$0 \$6 \$1
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1060 1061 1062 1063 1064 1065 1066 1067 1068	loadMetAdd_129_i:	ADDI SWAPBR NEG ADDI ADDI EXCH SUB EXCH ADDI EXCH SUB BNE	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$10 \$9 \$9 \$8 \$7 \$2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9	1100 1101 1102 1103 1104 1105 1106 1107 1108 1110 1111 1112 1113 1114 1115 1116	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ BNE XORI BNE XORI BNE XORI BNE XORI BNE XORI BNE XORI XORI XORI	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 \$3 \$8 \$7 \$7 7-3 \$7 \$3 \$6 \$0 \$6 \$1 \$6 \$1 \$7 \$3
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1060 1061 1062 1063 1064 1065 1066 1067 1068	<pre>loadMetAdd_129_i: assert_124:</pre>	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH SUB	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$10 \$9 \$9 \$8 \$7 \$3 \$10 \$9	1100 1101 1102 1103 1104 1105 1106 1107 1108 1110 1111 1112 1113 1114 1115 1116	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ BNE XORI BNE ADD ADDI EXCH ADDI SUB BEQ XORI	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 3 \$8 \$7 \$7 -3 \$7 \$3 \$6 \$0 \$6 \$1
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1060 1061 1062 1063 1064 1065 1066 1067 1068	<pre>loadMetAdd_129_i: assert_124:</pre>	ADDI SWAPBR NEG ADDI ADDI EXCH SUB EXCH ADDI EXCH SUB BNE	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$10 \$9 \$9 \$8 \$7 \$2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9	1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116	<pre>cmp_bot_140:</pre>	BNE BEQ XORI BEQ XOR BEQ XORI BEQ BNE XORI BNE XORI BNE XORI BNE XORI BNE XORI BNE XORI XORI XORI	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 \$3 \$8 \$7 \$7 7-3 \$7 \$3 \$6 \$0 \$6 \$1 \$6 \$1 \$7 \$3
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069	<pre>loadMetAdd_129_i: assert_124:</pre>	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$10 \$9 \$9 \$8 \$7 \$3 \$10 \$9 \$10 \$9 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10	1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119	<pre>cmp_bot_140:</pre>	BNE BEQ  XORI BEQ  XORI BEQ  XORI BEQ  BNE  XORI BNE  ADD  ADDI EXCH ADDI SUB BEQ  XORI XORI SUB BEQ  XORI XORI BRA	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 \$3 \$
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069	<pre>loadMetAdd_129_i: assert_124:</pre>	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI ADDI SUB BNE	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$10 \$9 \$9 \$8 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10	1100 1101 1102 1103 1104 1105 1106 1107 1108 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120	<pre>cmp_bot_140:</pre>	BNE BEQ  XORI BEQ  XORI BEQ  XORI BEQ  BNE  XORI BNE  ADD  ADDI EXCH ADDI SUB BEQ  XORI XORI SUB BEQ  XORI XORI BRA BRA	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 \$3 \$7 -3 \$7 \$3 \$6 \$0 \$6 \$1 \$8 \$1 \$9 \$0
1043 1044 1045 1046 1047 1048 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069	<pre>loadMetAdd_129_i: assert_124:</pre>	ADDI SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH EXCH ADDI EXCH EXCH EXCH EXCH EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH EXCH EXCH	\$12 -1042 \$12 \$12 \$12 1042 \$1 1 \$9 \$1 \$1 1 \$3 \$1 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 3 \$11 \$10 \$12 \$11 \$11 \$10 \$10 -3 \$10 \$9 \$9 \$8 \$7 \$3 \$7 2 \$8 \$7 \$7 -2 \$7 \$3 \$10 \$9 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10	1100 1101 1102 1103 1104 1105 1106 1107 1108 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121	<pre>cmp_bot_140:</pre>	BNE BEQ  XORI BEQ  XORI BEQ  XORI BEQ  BNE  XORI BNE  ADD  ADDI EXCH ADDI SUB BEQ  XORI XORI SUB BEQ  XORI XORI BRA BRA ADD	\$8 \$0 \$9 \$0 \$10 1 \$9 \$0 \$6 \$10 \$9 \$0 \$10 1 \$9 \$0 \$8 \$0 \$9 \$1 \$8 \$0 \$7 \$3 \$7 3 \$8 \$7 \$7 -3 \$7 \$3 \$6 \$0 \$6 \$1 \$8 \$0

1124		ADDI	\$7 -3	1184		BEQ	\$9 \$0
1125 1126		SUB XOR	\$7 \$3 \$9 \$8	1185	f_bot_148_i cmp_bot_146_i:	BNE	\$8 \$0
1127	loadMetAdd_143:	EXCH	\$10 \$9	1100	cmp_top_145_i	DNE	<b>40 40</b>
1128	_	ADDI	\$10 3	1186	1- 1	XORI	\$9 1
1129		EXCH	\$11 \$10	1187	cmp_top_145_i:	BNE	\$8 \$0
1130		XOR	\$12 \$11	1100	cmp_bot_146_i	3.DD	67 63
1131 1132		EXCH ADDI	\$11 \$10 \$10 -3	1188 1189		ADD ADDI	\$7 \$3 \$7 3
1133		EXCH	\$10 \$9	1190		EXCH	\$8 \$7
1134		ADD	\$7 \$3	1191		ADDI	\$7 -3
1135		ADDI	\$7 3	1192		SUB	\$7 \$3
1136		EXCH	\$8 \$7	1193		BRA	l_mirror_7_top
1137 1138		ADDI SUB	\$7 <b>-</b> 3 \$7 \$3	1194	l_insertNode_1_top: l_insertNode_1_bot	BRA	
1139		EXCH	\$3 \$1	1195	1_1113C1	ADDI	\$1 1
1140		ADDI	\$1 -1	1196		EXCH	\$2 \$1
1141		EXCH	\$9 \$1	1197		EXCH	\$6 \$1
1142		ADDI	\$1 -1	1198		ADDI	\$1 -1
1143 1144	l_jmp_144:	ADDI SWAPBR	\$12 -1142	1199 1200		EXCH ADDI	\$7 \$1 \$1 -1
1144		NEG	\$12	1200		EXCH	\$3 \$1
1146		ADDI	\$12 1142	1202		ADDI	\$1 -1
1147		ADDI	\$1 1	1203	l_insertNode_1:	SWAPBR	\$2
1148		EXCH	\$9 \$1	1204		NEG	\$2
1149 1150		ADDI EXCH	\$1 1 \$3 \$1	1205 1206		ADDI EXCH	\$1 1 \$3 \$1
1151		ADD	\$7 \$3	1200		ADDI	\$1 1
1152		ADDI	\$7 3	1208		EXCH	\$7 \$1
1153		EXCH	\$8 \$7	1209		ADDI	\$1 1
1154		ADDI	\$7 -3	1210		EXCH	\$6 \$1
1155		SUB	\$7 \$3	1211		EXCH ADDI	\$2 \$1
1156 1157		EXCH ADDI	\$10 \$9 \$10 3	1212 $1213$		ADDI	\$1 -1 \$9 \$3
1158		EXCH	\$11 \$10	1214		ADDI	\$9 2
1159		XOR	\$12 \$11	1215		EXCH	\$10 \$9
1160		EXCH	\$11 \$10	1216		ADDI	\$9 -2
1161	1	ADDI	\$10 -3	1217	153.	SUB	\$9 \$3
1162 1163	loadMetAdd_143_i:	EXCH XOR	\$10 \$9 \$9 \$8	1218	cmp_top_153:   cmp_bot_154	BNE	\$10 \$0
1164		ADD	\$7 \$3	1219	6mp_200_101	XORI	\$11 1
1165		ADDI	\$7 3	1220	cmp_bot_154:	BNE	\$10 \$0
1166		EXCH	\$8 \$7		cmp_top_153		
1167		ADDI	\$7 -3	1221	155	EXCH	\$12 \$6
1168 1169	assert_138:	SUB BNE	\$7 \$3 \$6 \$0	1222	cmp_top_155: cmp_bot_156	BEQ	\$12 \$0
1100	assert_true_136	2	70 70	1223	6mp_200_100	XORI	\$13 1
1170		ADD	\$7 \$3	1224	cmp_bot_156:	BEQ	\$12 \$0
1171		ADDI	\$7 3		cmp_top_155		
1172 1173		EXCH ADDI	\$8 \$7 \$7 -3	1225	f_top_157:	ANDX BEQ	\$14 \$11 \$13 \$14 \$0
1173		SUB	\$7 \$3	1226	f_bot_158	PFQ	\$14 \$U
1175	cmp_top_145:	BNE	\$8 \$0	1227	1_200_100	XORI	\$15 1
	cmp_bot_146			1228	f_bot_158:	BEQ	\$14 \$0
1176		XORI	\$9 1		f_top_157		
1177	cmp_bot_146:	BNE	\$8 \$0	1229	f bot 150 ;	XOR	\$8 \$15
1178	cmp_top_145 f_top_147:	BEQ	\$9 \$0	1230	f_bot_158_i: f_top_157_i	BEQ	\$14 \$0
11.0	f_bot_148		12 12	1231		XORI	\$15 1
1179		XORI	\$10 1		f_top_157_i:	BEQ	\$14 \$0
1180	f_bot_148:	BEQ	\$9 \$0		f_bot_158_i		
1101	f_top_147	VOD	¢ 6   ¢ 1 0	1233	amp bet 150 de	ANDX	\$14 \$11 \$13
1181 1182	f_bot_148_i:	XOR BEQ	\$6 \$10 \$9 \$0	1234	cmp_bot_156_i: cmp_top_155_i	BEQ	\$12 \$0
1102	f_top_147_i		12 12	1235		XORI	\$13 1
1183		XORI	\$10 1		cmp_top_155_i:	BEQ	\$12 \$0

	amp bot 156 i			1289	l	XORI	\$13 1
1237	cmp_bot_156_i	EXCH	\$12 \$6	1299	cmp_top_162_i:	BNE	\$12 \$0
1238	cmp_bot_154_i:	BNE	\$10 \$0	1230	cmp_bot_163_i	2112	712 70
	cmp_top_153_i			1291	1 1 - 1 - 1 - 1 - 1	EXCH	\$12 \$6
1239		XORI	\$11 1	1292	cmp_bot_161_i:	BEQ	\$10 \$0
1240	cmp_top_153_i:	BNE	\$10 \$0		cmp_top_160_i		
	cmp_bot_154_i			1293		XORI	\$11 1
1241		ADD	\$9 \$3	1294	cmp_top_160_i:	BEQ	\$10 \$0
1242		ADDI	\$9 2		cmp_bot_161_i		
1243		EXCH	\$10 \$9	1295		ADD	\$9 \$3
1244		ADDI	\$9 -2	1296		ADDI	\$9 2
1245	test 149:	SUB BEQ	\$9 \$3 \$8 \$0	1297		EXCH ADDI	\$10 \$9 \$9 -2
1246	test false 151	PFŐ	30 3U	1298 1299		SUB	\$9 =2
1247		XORI	\$8 1	1300		ADD	\$9 \$3
1248		ADD	\$9 \$3	1301		ADDI	\$9 2
1249		ADDI	\$9 2	1302		EXCH	\$10 \$9
1250		EXCH	\$10 \$9	1303		ADDI	\$9 -2
1251		ADDI	\$9 -2	1304		SUB	\$9 \$3
1252		SUB	\$9 \$3	1305	cmp_top_170:	BEQ	\$10 \$0
1253		EXCH	\$11 \$6		cmp_bot_171		
1254	swap_159:	XOR	\$10 \$11	1306		XORI	\$11 1
1255		XOR	\$11 \$10	1307	cmp_bot_171:	BEQ	\$10 \$0
1256		XOR EXCH	\$10 \$11 \$11 \$6	1200	cmp_top_170	DEO.	\$11 \$0
1257 $1258$		ADD	\$9 \$3	1308	f_top_172: f_bot_173	BEQ	SII SO
1259		ADDI	\$9 2	1309	1_500_175	XORI	\$12 1
1260		EXCH	\$10 \$9		f bot 173:	BEQ	\$11 \$0
1261		ADDI	\$9 -2		f_top_172	~	
1262		SUB	\$9 \$3	1311		XOR	\$8 \$12
1263		XORI	\$8 1	1312	f_bot_173_i:	BEQ	\$11 \$0
1264	assert_true_150:	BRA	assert_152		f_top_172_i		
1265	test_false_151:	BRA	test_149	1313		XORI	\$12 1
1266	assert_152:	BNE	\$8 \$0	1314	f_top_172_i:	BEQ	\$11 \$0
1005	assert_true_150	300	\$9 \$3	1015	f_bot_173_i	DEC	Ċ10 Ċ0
1267 1268		ADD ADDI	\$9 2	1315	cmp_bot_171_i: cmp_top_170_i	BEQ	\$10 \$0
1269		EXCH	\$10 \$9	1316	Cmp_cop_1/0_1	XORI	\$11 1
1270		ADDI	\$9 -2	1317	cmp_top_170_i:	BEQ	\$10 \$0
1271		SUB	\$9 \$3		cmp_bot_171_i	~	
1272	cmp_top_160:	BEQ	\$10 \$0	1318	<del>-</del>	ADD	\$9 \$3
	cmp_bot_161			1319		ADDI	\$9 2
1273		XORI	\$11 1	1320		EXCH	\$10 \$9
1274	cmp_bot_161:	BEQ	\$10 \$0	1321		ADDI	\$9 -2
1075	cmp_top_160	EVOII	¢10 ¢6	1322	L 100.	SUB	\$9 \$3
1275 1276	cmp_top_162:	EXCH BNE	\$12 \$6 \$12 \$0	1323	test_166: test false 168	BEQ	\$8 \$0
1210	cmp_bot_163		714 70	1324	CC3C_1013C_100	XORI	\$8 1
1277		XORI	\$13 1	1325		ADD	\$9 \$3
1278	cmp_bot_163:	BNE	\$12 \$0	1326		ADDI	\$9 2
	cmp_top_162			1327		EXCH	\$10 \$9
1279		ANDX	\$14 \$11 \$13			ADDI	\$9 -2
1280	f_top_164:	BEQ	\$14 \$0	1329		SUB	\$9 \$3
	f_bot_165		A1E 1	1330	1 24 2 11 174	XOR	\$11 \$10
1281	f bot 165.	XORI	\$15 1	1331	loadMetAdd_174:	EXCH	\$12 \$11
1282	f_bot_165: f_top_164	BEQ	\$14 \$0	1332 1333		ADDI EXCH	\$12 1 \$13 \$12
1283		XOR	\$8 \$15	1334		XOR	\$14 \$13
1284	f_bot_165_i:	BEQ	\$14 \$0	1335		EXCH	\$13 \$12
	f_top_164_i	~		1336		ADDI	\$12 -1
1285		XORI	\$15 1	1337		EXCH	\$12 \$11
1286	f_top_164_i:	BEQ	\$14 \$0	1338		ADD	\$9 \$3
	f_bot_165_i			1339		ADDI	\$9 2
1287		ANDX	\$14 \$11 \$13			EXCH	\$10 \$9
1288	*	BNE	\$12 \$0	1341		ADDI	\$9 -2
	cmp_top_162_i			1342		SUB	\$9 \$3

1343		EXCH	\$3 \$1	1401		XORI	\$11 1
1344		ADDI	\$1 -1	1402	cmp_top_176_i:	BEQ	\$10 \$0
1345		EXCH	\$6 \$1		cmp_bot_177_i	2	1 1-
					CIIIP_DOC_I / /_I		40.40
1346		ADDI	\$1 -1	1403		ADD	\$9 \$3
1347		EXCH	\$7 \$1	1404		ADDI	\$9 2
1348		ADDI	\$1 -1	1405		EXCH	\$10 \$9
1349		EXCH	\$11 \$1	1406		ADDI	\$9 -2
1350		ADDI	\$1 -1	1407		SUB	\$9 \$3
1351		ADDI	\$14 -1350	1408	l_insertNode_1_bot:	BRA	
1352	l_jmp_175:	SWAPBR	\$14		l_insertNode_1_top		
1353	_3 :_	NEG	\$14	1409	1_sum_2_top:	BRA	l sum 2 bot
		ADDI			1 - 5 am _ 2 _ c o p •		\$1 1
1354			\$14 1350	1410		ADDI	
1355		ADDI	\$1 1	1411		EXCH	\$2 \$1
1356		EXCH	\$11 \$1	1412		EXCH	\$6 \$1
1357		ADDI	\$1 1	1413		ADDI	\$1 -1
		EXCH	\$7 \$1			EXCH	\$3 \$1
1358				1414			
1359		ADDI	\$1 1	1415		ADDI	\$1 -1
1360		EXCH	\$6 \$1	1416	l_sum_2:	SWAPBR	\$2
1361		ADDI	\$1 1	1417		NEG	\$2
1362		EXCH	\$3 \$1	1418		ADDI	\$1 1
1363		ADD	\$9 \$3	1419		EXCH	\$3 \$1
1364		ADDI	\$9 2	1420		ADDI	\$1 1
1365		EXCH	\$10 \$9	1421		EXCH	\$6 \$1
1366		ADDI	\$9 -2	1422		EXCH	\$2 \$1
		SUB	\$9 \$3			ADDI	\$1 -1
1367				1423			
1368		EXCH	\$12 \$11	1424		ADD	\$8 \$3
1369		ADDI	\$12 1	1425		ADDI	\$8 2
1370		EXCH	\$13 \$12	1426		EXCH	\$9 \$8
1371		XOR	\$14 \$13	1427		ADDI	\$8 -2
1372		EXCH	\$13 \$12	1428		SUB	\$8 \$3
1373		ADDI	\$12 -1	1429	cmp_top_184:	BEQ	\$9 \$0
1374	loadMetAdd_174_i:	EXCH	\$12 \$11		cmp_bot_185		
1375		XOR	\$11 \$10	1430	_	XORI	\$10 1
		ADD			amp ba+ 105.		
1376			\$9 \$3	1431	cmp_bot_185:	BEQ	\$9 \$0
1377		ADDI	\$9 2		cmp_top_184		
1378		EXCH	\$10 \$9	1432	f_top_186:	BEQ	\$10 \$0
1379		ADDI	\$9 -2		f_bot_187		
1380		SUB	\$9 \$3	1433		XORI	\$11 1
1					C 1 107		
1381		XORI	\$8 1	1434	f_bot_187:	BEQ	\$10 \$0
1382	assert_true_167:	BRA	assert_169		f_top_186		
1383	test_false_168:	BRA	test_166	1435		XOR	\$7 \$11
1384	assert_169:	BNE	\$8 \$0	1436	f_bot_187_i:	BEQ	\$10 \$0
1004		2112	40 40	1400		222	710 40
	assert_true_167				f_top_186_i		
1385		ADD	\$9 \$3	1437		XORI	\$11 1
1386		ADDI	\$9 2	1438	f_top_186_i:	BEQ	\$10 \$0
1387		EXCH	\$10 \$9		f_bot_187_i		
1388		ADDI	\$9 -2	1439		BEQ	\$9 \$0
						2	, , , ,
1389		SUB	\$9 \$3		cmp_top_184_i		***
1390	cmp_top_176:	BEQ	\$10 \$0	1440		XORI	\$10 1
	cmp_bot_177			1441	cmp_top_184_i:	BEQ	\$9 \$0
1391		XORI	\$11 1		cmp_bot_185_i		
1392	cmp_bot_177:	BEQ	\$10 \$0	1442	<u> </u>	ADD	\$8 \$3
1002	=	2	7 ± 0 Y 0				
	cmp_top_176			1443		ADDI	\$8 2
1393	f_top_178:	BEQ	\$11 \$0	1444		EXCH	\$9 \$8
	f_bot_179			1445		ADDI	\$8 -2
1394		XORI	\$12 1	1446		SUB	\$8 \$3
	f_bot_179:		\$11 \$0		test_180:	BEQ	\$7 \$0
1395		BEQ	ATT AA	1447	_	DΕŽ	ų / ų U
	f_top_178				test_false_182		
1396		XOR	\$8 \$12	1448		XORI	\$7 1
1397	f_bot_179_i:	BEQ	\$11 \$0	1449		ADD	\$8 \$3
	f_top_178_i	_		1450		ADDI	\$8 2
1200		VODT	¢12 1				
1398	170 1	XORI	\$12 1	1451		EXCH	\$9 \$8
1399		BEQ	\$11 \$0	1452		ADDI	\$8 -2
	f_bot_179_i			1453		SUB	\$8 \$3
1400	cmp_bot_177_i:	BEQ	\$10 \$0	1454		XOR	\$10 \$9
	cmp_top_176_i	_	• •		loadMetAdd_188:	EXCH	\$11 \$10
ı	5			1 200			1 7-0

1456		ADDI	\$11 2	1517	f_bot_193_i:	BEQ	\$10 \$0
1457		EXCH	\$12 \$11		f_top_192_i	2	1-1 11
					1_COP_192_1		A11 1
1458		XOR	\$13 \$12	1518		XORI	\$11 1
1459		EXCH	\$12 \$11	1519	f_top_192_i:	BEQ	\$10 \$0
1460		ADDI	\$11 -2		f_bot_193_i		
1461		EXCH	\$11 \$10	1520	cmp_bot_191_i:	BEQ	\$9 \$0
			\$8 \$3	1020	_	222	43 40
1462		ADD			cmp_top_190_i		
1463		ADDI	\$8 2	1521		XORI	\$10 1
1464		EXCH	\$9 \$8	1522	cmp_top_190_i:	BEQ	\$9 \$0
1465		ADDI	\$8 -2		cmp_bot_191_i		
		SUB	\$8 \$3	1500	0	ADD	\$8 \$3
1466				1523			
1467		EXCH	\$3 \$1	1524		ADDI	\$8 2
1468		ADDI	\$1 -1	1525		EXCH	\$9 \$8
1469		EXCH	\$6 \$1	1526		ADDI	\$8 -2
1470		ADDI	\$1 -1	1527		SUB	\$8 \$3
					1 0 1		
1471		EXCH	\$10 \$1	1528		BRA	l_sum_2_top
1472		ADDI	\$1 -1	1529	l_mirror_3_top:	BRA	l_mirror_3_bot
1473		ADDI	\$13 -1472	1530		ADDI	\$1 1
1474	l_jmp_189:	SWAPBR	\$13	1531		EXCH	\$2 \$1
1475		NEG	\$13	1532		EXCH	\$3 \$1
1476		ADDI	\$13 1472	1533		ADDI	\$1 -1
1477		ADDI	\$1 1	1534	l_mirror_3:	SWAPBR	\$2
1478		EXCH	\$10 \$1	1535		NEG	\$2
1479		ADDI	\$1 1	1536		ADDI	\$1 1
1480		EXCH	\$6 \$1	1537		EXCH	\$3 \$1
1481		ADDI	\$1 1	1538		EXCH	\$2 \$1
1482		EXCH	\$3 \$1	1539		ADDI	\$1 -1
1483		ADD	\$8 \$3	1540		ADD	\$7 \$3
		ADDI					\$7 2
1484			\$8 2	1541		ADDI	
1485		EXCH	\$9 \$8	1542		EXCH	\$8 \$7
1486		ADDI	\$8 -2	1543		ADDI	\$7 -2
1487		SUB	\$8 \$3	1544		SUB	\$7 \$3
1488		EXCH	\$11 \$10	1545	cmp_top_198:	BEQ	\$8 \$0
				1343		DEQ	70 70
1489		ADDI	\$11 2		cmp_bot_199		
1490		EXCH	\$12 \$11	1546		XORI	\$9 1
1491		XOR	\$13 \$12	1547	cmp_bot_199:	BEQ	\$8 \$0
1492		EXCH	\$12 \$11		cmp_top_198	_	
				1540		DEC	20 20
1493		ADDI	\$11 -2	1548		BEQ	\$9 \$0
1494	loadMetAdd_188_i:	EXCH	\$11 \$10		f_bot_201		
1495		XOR	\$10 \$9	1549		XORI	\$10 1
1496		ADD	\$8 \$3	1550	f_bot_201:	BEQ	\$9 \$0
1497		ADDI	\$8 2	1000	f_top_200		73 70
					1_cop_200		46 410
1498		EXCH	\$9 \$8	1551		XOR	\$6 \$10
1499		ADDI	\$8 -2	1552	f_bot_201_i:	BEQ	\$9 \$0
1500		SUB	\$8 \$3		f_top_200_i		
1501		XORI	\$7 1	1553		XORI	\$10 1
1502	assert_true_181:	BRA			f_top_200_i:	BEQ	\$9 \$0
				1004		2	T 2 Y 0
	test_false_182:	BRA	test_180		f_bot_201_i		
1504	assert_183:	BNE	\$7 \$0	1555	cmp_bot_199_i:	BEQ	\$8 \$0
	assert_true_181				cmp_top_198_i		
1505	_	ADD	\$8 \$3	1556		XORI	\$9 1
1506		ADDI	\$8 2	1557	cmp_top_198_i:	BEQ	\$8 \$0
				1997		מיים	40 40
1507		EXCH	\$9 \$8		cmp_bot_199_i		
1508		ADDI	\$8 -2	1558		ADD	\$7 \$3
1509		SUB	\$8 \$3	1559		ADDI	\$7 2
1510	cmp_top_190:	BEQ	\$9 \$0	1560		EXCH	\$8 \$7
1310			, , , ,				
	cmp_bot_191		410 1	1561		ADDI	\$7 -2
1511		XORI	\$10 1	1562		SUB	\$7 \$3
1512	cmp_bot_191:	BEQ	\$9 \$0	1563	test_194:	BEQ	\$6 \$0
	cmp_top_190				test_false_196	-	
1519	f_top_192:	BEQ	\$10 \$0	1564		XORI	\$6 1
1013	=	ההר	4 T O 4 O				
	f_bot_193			1565		ADD	\$7 \$3
1514		XORI	\$11 1	1566		ADDI	\$7 2
1515	f_bot_193:	BEQ	\$10 \$0	1567		EXCH	\$8 \$7
	f_top_192	~		1568		ADDI	\$7 -2
1510	cop122	VOR	67 611				
1516		XOR	\$7 \$11	1569		SUB	\$7 \$3

1570		XOR	\$9 \$8	1630		XORI	\$10 1
1571	loadMetAdd_202:	EXCH	\$10 \$9	1631	f_top_206_i:	BEQ	\$9 \$0
1572		ADDI	\$10 3		f_bot_207_i		
1573		EXCH	\$11 \$10	1632		BEQ	\$8 \$0
1574		XOR	\$12 \$11		cmp_top_204_i		1 - 1 -
				1699	Cmp_cop_204_1	VODT	¢0 1
1575		EXCH	\$11 \$10	1633		XORI	\$9 1
1576		ADDI	\$10 -3	1634	cmp_top_204_i:	BEQ	\$8 \$0
1577		EXCH	\$10 \$9		cmp_bot_205_i		
1578		ADD	\$7 \$3	1635		ADD	\$7 \$3
1579		ADDI	\$7 2	1636		ADDI	\$7 2
1580		EXCH	\$8 \$7	1637		EXCH	\$8 \$7
1581		ADDI	\$7 -2	1638		ADDI	\$7 -2
		SUB					
1582			\$7 \$3	1639		SUB	\$7 \$3
1583		EXCH	\$3 \$1	1640		BRA	l_mirror_3_top
1584		ADDI	\$1 -1	1641	l_main_0_top:	BRA	l_main_0_bot
1585		EXCH	\$9 \$1	1642		ADDI	\$1 1
1586		ADDI	\$1 -1	1643		EXCH	\$2 \$1
1587		ADDI	\$12 -1586	1644		EXCH	\$3 \$1
1588	1_jmp_203:	SWAPBR		1645		ADDI	\$1 -1
1589		NEG	\$12	1646	l_main_0:	SWAPBR	
1590		ADDI	\$12 1586	1647		NEG	\$2
1591		ADDI	\$1 1	1648		ADDI	\$1 1
1592		EXCH	\$9 \$1	1649		EXCH	\$3 \$1
1593		ADDI	\$1 1	1650		EXCH	\$2 \$1
1594		EXCH	\$3 \$1	1651		ADDI	\$1 -1
1595		ADD	\$7 \$3	1652		EXCH	\$3 \$1
1596		ADDI	\$7 2	1653		ADDI	\$1 -1
		EXCH	\$8 \$7		obi con 200.	ADDI	\$8 4
1597				1654	obj_con_208:		
1598		ADDI	\$7 -2	1655		EXCH	\$8 \$1
1599		SUB	\$7 \$3	1656		ADDI	\$1 -1
1600		EXCH	\$10 \$9	1657		EXCH	\$7 \$1
1601		ADDI	\$10 3	1658		ADDI	\$1 -1
1602		EXCH	\$11 \$10	1659		BRA	l_malloc
1603		XOR	\$12 \$11	1660		ADDI	\$1 1
1604		EXCH	\$11 \$10	1661		EXCH	\$7 \$1
1605		ADDI	\$10 -3	1662		ADDI	\$1 1
1606	loadMetAdd_202_i:	EXCH	\$10 \$9	1663		EXCH	\$8 \$1
1607		XOR	\$9 \$8	1664	obj_con_208_i:	ADDI	\$8 -4
1608		ADD	\$7 \$3	1665		ADDI	\$1 1
1609		ADDI	\$7 2	1666		EXCH	\$3 \$1
1610		EXCH	\$8 \$7	1667		ADD	\$6 \$3
1611		ADDI	\$7 -2	1668		ADDI	\$6 3
		SUB				XORI	\$8 5
1612			\$7 \$3	1669			
1613		XORI	\$6 1	1670		EXCH	\$8 \$7
1614	assert_true_195:	BRA	assert_197	1671		ADDI	\$7 1
	test_false_196:	BRA	test_194	1672		XORI	\$8 1
1616	assert_197:	BNE	\$6 \$0	1673		EXCH	\$8 \$7
j	assert_true_195			1674	obj_con_208_bot:	ADDI	\$7 -1
1617		ADD	\$7 \$3	1675		EXCH	\$7 \$6
1618		ADDI	\$7 2	1676		ADDI	\$6 -3
1619		EXCH	\$8 \$7	1677		SUB	\$6 \$3
1620		ADDI	\$7 -2	1678		ADD	\$6 \$3
1621		SUB	\$7 \$3	1679		ADDI	\$6 4
1622	cmp_top_204:	BEQ	\$8 \$0	1680		EXCH	\$7 \$6
	cmp_bot_205			1681		ADDI	\$6 -4
1623		XORI	\$9 1	1682		SUB	\$6 \$3
1624	cmp_bot_205:	BEQ	\$8 \$0	1683		XORI	\$8 3
	cmp_top_204	~		1684		ADD	\$7 \$8
1625	f_top_206:	BEQ	\$9 \$0	1685		XORI	\$8 3
1020	_	החה	72 70				\$6 \$3
	f_bot_207	we==	610 1	1686		ADD	
1626		XORI	\$10 1	1687		ADDI	\$6 4
1627	f_bot_207:	BEQ	\$9 \$0	1688		EXCH	\$7 \$6
	f_top_206			1689		ADDI	\$6 -4
1628		XOR	\$6 \$10	1690		SUB	\$6 \$3
1629	f_bot_207_i:	BEQ	\$9 \$0	1691	localBlock_227:	XOR	\$6 \$1
	f_top_206_i			1692	_	XOR	\$7 \$0
,							•

1000		EVOU	ć 7	ć 1	1740	EVO		60 61
1693		EXCH	\$7		1742	EXCH		\$3 \$1
1694		ADDI	\$1		1743	ADDI		\$1 -1
1695		XORI	\$7		1744	EXC		\$8 \$1
1696	entry_209:	BEQ	\$7	\$0	1745	ADDI		\$1 -1
	assert_211				1746	EXCH		\$6 \$1
1697		EXCH	\$8	\$6	1747	ADDI		\$1 -1
1698	cmp_top_213:	BNE	\$8	\$0	1748	obj_con_217: ADDI		\$10 8
	cmp_bot_214				1749	EXCH		\$10 \$1
1.000	Cmp_000_211	XORI	\$9	1		ADDI		\$1 -1
1699	214				1750			
1700	cmp_bot_214:	BNE	\$8	\$0	1751	EXCH		\$9 \$1
	cmp_top_213				1752	ADDI		\$1 -1
1701	f_top_215:	BEQ	\$9	\$0	1753	BRA		l_malloc
	f_bot_216				1754	ADDI		\$1 1
1702		XORI	\$10	1	1755	EXCH		\$9 \$1
1703	f_bot_216:	BEQ	\$9	\$0	1756	ADDI		\$1 1
	f_top_215	~			1757	EXCH		\$10 \$1
1704	1_000_213	XOR	¢7	\$10				\$10 -8
	5 1 1 016 1				1758	<u> </u>		
1705	f_bot_216_i:	BEQ	\$9	\$0	1759	ADDI		\$1 1
	f_top_215_i				1760	EXC		\$6 \$1
1706		XORI	\$10	1	1761	ADDI		\$1 1
1707	f_top_215_i:	BEQ	\$9	\$0	1762	EXCH		\$8 \$1
	f_bot_216_i				1763	ADDI		\$1 1
1708	cmp_bot_214_i:	BNE	\$8	\$0	1764	EXCH		\$3 \$1
	cmp_top_213_i				1765	XORI		\$10 8
1700	CMP_COP_Z13_1	VODT	\$9	1		EXCH		\$10 \$9
1709	212 1	XORI			1766			
1710	cmp_top_213_i:	BNE	\$8	\$0	1767	ADDI		\$9 1
	cmp_bot_214_i				1768	XORI		\$10 1
1711		EXCH	\$8	\$6	1769	EXCH		\$10 \$9
1712		EXCH	\$8	\$6	1770	obj_con_217_bot: ADDI		\$9 -1
1713		ADD	\$9	\$3	1771	EXCH		\$9 \$8
1714		ADDI	\$9	4	1772	EXCH		\$9 \$8
1715		EXCH		\$9	1773	XOR		\$10 \$9
1716		ADDI	\$9		1774	loadMetAdd_218: EXCE	r	\$11 \$10
						_		
1717		SUB	\$9		1775	ADDI		\$11 0
1718	cmp_top_223:	BNE	\$8	\$10	1776	EXCH		\$12 \$11
	cmp_bot_224				1777	XOR		\$13 \$12
1719		XORI	\$11	1	1778	EXCH		\$12 \$11
1720	cmp_bot_224:	BNE	\$8	\$10	1779	ADDI		\$11 0
	cmp_top_223				1780	EXCH		\$11 \$10
1721	f_top_225:	BEQ	\$11	\$0	1781	EXCH		\$9 \$8
	f_bot_226	~	·		1782	EXCH		\$3 \$1
1722	1_200_220	XORI	\$12	1	1783	ADDI		\$1 -1
	f_bot_226:	BEQ		\$0		EXCH		\$8 \$1
1723		PFŐ	ΥTI	ΨU	1784			
	f_top_225				1785	ADDI		\$1 -1
1724		XOR		\$12	1786	EXC		\$6 \$1
1725	f_bot_226_i:	BEQ	\$11	\$0	1787	ADDI		\$1 -1
	f_top_225_i				1788	EXCH		\$10 \$1
1726		XORI	\$12	1	1789	ADDI		\$1 -1
1727	f_top_225_i:	BEQ	\$11	\$0	1790	ADDI		\$13 -1789
	f_bot_226_i				1791	1_jmp_219: SWAF		
1728	cmp_bot_224_i:	BNE	\$8	\$10	1792	NEG		\$13
20	cmp_top_223_i		+ 5		1793	ADDI		\$13 1789
1700	Cmp_cop_223_1	VORT	¢11	1				
1729	222	XORI	\$11		1794	ADDI		\$1 1
1730	cmp_top_223_i:	BNE	\$8	\$10	1795	EXC		\$10 \$1
	cmp_bot_224_i				1796	ADDI		\$1 1
1731		ADD	\$9	\$3	1797	EXCH		\$6 \$1
1732		ADDI	\$9	4	1798	ADDI		\$1 1
1733		EXCH	\$10	\$9	1799	EXC		\$8 \$1
1734		ADDI	\$9		1800	ADDI		\$1 1
1735		SUB	\$9		1801	EXCH		\$3 \$1
1736		EXCH	\$8		1802	EXCE		\$9 \$8
	tost 210.			\$0 exit_2		EXCE		
1737	test_210:	BNE						\$11 \$10
1738	localBlock_222:	XOR	\$8		1804	ADDI		\$11 0
1739		XOR	\$9		1805	EXC		\$12 \$11
1740		EXCH	\$9		1806	XOR		\$13 \$12
1741		ADDI	\$1	-1	1807	EXCH		\$12 \$11

1808		ADDI	\$11 0	1874	7	ADD	\$8 \$9
	1						
1809	loadMetAdd_218_i:	EXCH	\$11 \$10	1875			\$9 1
1810		XOR	\$10 \$9	1876		EXCH	\$8 \$6
1811		EXCH	\$9 \$8	1877	assert_211:	BRA	entry_209
1812		ADD	\$9 \$3	1878	exit_212:	BRA	test_210
1813		ADDI	\$9 3	1879	<b>X</b>	KORI	\$7 1
1814		EXCH	\$10 \$9	1880	2	ADD	\$7 \$3
1815		ADDI	\$9 -3	1881		ADDI	\$7 4
1816		SUB	\$9 \$3	1882		EXCH	\$8 \$7
1817		XOR	\$11 \$10	1883		ADDI	\$7 -4
1818	loadMetAdd_220:	EXCH	\$12 \$11	1884	S	SUB	\$7 \$3
1819		ADDI	\$12 0	1885	P	ADDI	\$1 1
1820		EXCH	\$13 \$12	1886	E	EXCH	\$9 \$1
1821		XOR	\$14 \$13	1887	2	KOR	\$9 \$8
1822		EXCH	\$13 \$12	1888		KOR	\$6 \$1
1823		ADDI	\$12 0	1889		ADD	\$7 \$3
1824		EXCH	\$12 \$11	1890		ADDI	\$7 4
1825		ADD	\$9 \$3	1891		EXCH	\$8 \$7
1826		ADDI	\$9 3	1892	Z.	ADDI	\$7 -4
1827		EXCH	\$10 \$9	1893	S	SUB	\$7 \$3
1828		ADDI	\$9 -3	1894	P	ADD	\$6 \$3
1829		SUB	\$9 \$3	1895	P	ADDI	\$6 3
1830		EXCH	\$3 \$1	1896		EXCH	\$7 \$6
		ADDI	\$1 -1			ADDI	\$6 -3
1831				1897			
1832		EXCH	\$8 \$1	1898		SUB	\$6 \$3
1833		ADDI	\$1 -1	1899		KOR	\$8 \$7
1834		EXCH	\$6 \$1	1900	loadMetAdd_228:	EXCH	\$9 \$8
1835		ADDI	\$1 -1	1901	P	ADDI	\$9 1
1836		EXCH	\$11 \$1	1902	E	EXCH	\$10 \$9
1837		ADDI	\$1 -1	1903		KOR	\$11 \$10
1838		ADDI	\$14 -1837	1904		EXCH	\$10 \$9
1	1 -mm 221.						
1839	l_jmp_221:	SWAPBR		1905		ADDI	\$9 -1
1840		NEG	\$14	1906		EXCH	\$9 \$8
1841		ADDI	\$14 1837	1907	P.	ADD	\$6 \$3
1842		ADDI	\$1 1	1908	P	ADDI	\$6 3
1843		EXCH	\$11 \$1	1909	E	EXCH	\$7 \$6
1844		ADDI	\$1 1	1910	P	ADDI	\$6 -3
1845		EXCH	\$6 \$1	1911		SUB	\$6 \$3
1846		ADDI	\$1 1	1912		ADD	\$12 \$3
1847		EXCH		1913		ADDI	\$12 2
1848		ADDI	\$1 1	1914			\$3 \$1
1849		EXCH	\$3 \$1	1915	P.	ADDI	\$1 -1
1850		ADD	\$9 \$3	1916	E	EXCH	\$12 \$1
1851		ADDI	\$9 3	1917	Z.	ADDI	\$1 -1
1852		EXCH	\$10 \$9	1918	E	EXCH	\$8 \$1
1853		ADDI	\$9 -3	1919		ADDI	\$1 -1
1854		SUB	\$9 \$3	1920		ADDI	\$11 -1919
1855		EXCH	\$12 \$11	1921		SWAPBR	
1856		ADDI	\$12 0	1922		NEG	\$11
1857		EXCH	\$13 \$12	1923			\$11 1919
1858		XOR	\$14 \$13	1924		ADDI	\$1 1
1859		EXCH	\$13 \$12	1925	E	EXCH	\$8 \$1
1860		ADDI	\$12 0	1926	P.	ADDI	\$1 1
1861	loadMetAdd_220_i:	EXCH	\$12 \$11	1927	E		\$12 \$1
1862	_	XOR	\$11 \$10	1928			\$1 1
1863		ADD	\$9 \$3	1929		EXCH	\$3 \$1
1864		ADDI	\$9 3	1930		ADDI	\$12 -2
1865		EXCH	\$10 \$9	1931		SUB	\$12 \$3
1866		ADDI	\$9 -3	1932		ADD	\$6 \$3
1867		SUB	\$9 \$3	1933			\$6 3
1868		ADDI	\$1 1	1934	E	EXCH	\$7 \$6
1869		EXCH	\$9 \$1	1935	P.	ADDI	\$6 -3
1870		XOR	\$9 \$0	1936	S	SUB	\$6 \$3
1871	localBlock_222_i:	XOR	\$8 \$1	1937			\$9 \$8
1872		EXCH	\$8 \$6	1938			\$9 1
1873		XORI	\$9 1	1939			\$10 \$9
1019		TORI	Y / 1	1999	F	27.011	Y±0 YJ

1940	Х	(OR	\$11 \$10	1999	start:	BRA	top
1941		EXCH	\$10 \$9	2000	beare.	START	ч
1942		ADDI	\$9 -1	2001		ADDI	\$4 2055
1943		EXCH	\$9 \$8	2002		XOR	\$5 \$4
1944		KOR	\$8 \$7	2003		ADDI	\$5 10
1945		ADD	\$6 \$3	2004		XOR	\$7 \$5
1946		ADDI	\$6 3	2005		ADDI	\$4 10
1947		EXCH	\$7 \$6	2006		ADDI	\$4 -1
1948		ADDI	\$6 -3	2007		EXCH	\$7 \$4
1949		SUB	\$6 \$3	2008		ADDI	\$4 1
1950	P	ADD	\$6 \$3	2009		ADDI	\$4 -10
1951	Z.	ADDI	\$6 3	2010		XOR	\$1 \$5
1952	E	EXCH	\$7 \$6	2011		ADDI	\$1 2048
1953	Z.	ADDI	\$6 -3	2012		ADDI	\$1 -8
1954	S	SUB	\$6 \$3	2013		XOR	\$3 \$1
1955	X	(OR	\$8 \$7	2014		XORI	\$6 4
1956	loadMetAdd_230:	EXCH	\$9 \$8	2015		EXCH	\$6 \$3
1957	P	ADDI	\$9 2	2016		ADDI	\$1 -1
1958	E	EXCH	\$10 \$9	2017		EXCH	\$3 \$1
1959	X	(OR	\$11 \$10	2018		ADDI	\$1 -1
1960	E	EXCH	\$10 \$9	2019		BRA	1_main_0
1961	P	ADDI	\$9 -2	2020		ADDI	\$1 1
1962	E	EXCH	\$9 \$8	2021		EXCH	\$3 \$1
1963	P	ADD	\$6 \$3	2022		ADDI	\$3 1
1964	Z.	ADDI	\$6 3	2023		ADDI	\$3 1
1965	E	EXCH	\$7 \$6	2024		EXCH	\$6 \$3
1966	P	ADDI	\$6 -3	2025		XORI	\$7 1
1967	s	SUB	\$6 \$3	2026		EXCH	\$6 \$7
1968	E	EXCH	\$3 \$1	2027		XORI	\$7 1
1969	P	ADDI	\$1 -1	2028		ADDI	\$3 -1
1970	E	EXCH	\$8 \$1	2029		ADDI	\$3 -1
1971	P	ADDI	\$1 -1	2030		ADDI	\$3 1
1972	P	ADDI	\$11 -1971	2031		ADDI	\$3 2
1973	l_jmp_231:	SWAPBR	\$11	2032		EXCH	\$6 \$3
1974	N	NEG	\$11	2033		XORI	\$7 2
1975	<i>p</i>	ADDI	\$11 1971	2034		EXCH	\$6 \$7
1976	P.	ADDI	\$1 1	2035		XORI	\$7 2
1977	E	EXCH	\$8 \$1	2036		ADDI	\$3 -2
1978	Z.	ADDI	\$1 1	2037		ADDI	\$3 -1
1979	E	EXCH	\$3 \$1	2038		ADDI	\$3 1
1980	Z.	ADD	\$6 \$3	2039		ADDI	\$3 3
1981	P	ADDI	\$6 3	2040		EXCH	\$6 \$3
1982	E	EXCH	\$7 \$6	2041		XORI	\$7 3
1983	<i>P</i>	ADDI	\$6 -3	2042		EXCH	\$6 \$7
1984	S	SUB	\$6 \$3	2043		XORI	\$7 3
1985	E	EXCH	\$9 \$8	2044		ADDI	\$3 -3
1986	P	ADDI	\$9 2	2045		ADDI	\$3 -1
1987	E	EXCH	\$10 \$9	2046		ADDI	\$1 1
1988	λ	(OR	\$11 \$10	2047		EXCH	\$6 \$3
1989		EXCH	\$10 \$9	2048		XORI	\$6 4
1990		ADDI	\$9 -2	2049		XOR	\$3 \$1
1991		EXCH	\$9 \$8	2050		ADDI	\$1 8
1992		KOR	\$8 \$7	2051		ADDI	\$1 -2048
1993		ADD	\$6 \$3	2052		XOR	\$1 \$5
1994		ADDI	\$6 3	2053		ADDI	\$5 -10
1995		EXCH	\$7 \$6	2054		XOR	\$5 \$4
1996		ADDI	\$6 -3	2055		ADDI	\$4 -2055
1997		SUB	\$6 \$3		finish:	FINISH	
1998	l_main_0_bot:	BRA	l_main_0_t	op			

## DoublyLinkedList.rplpp

```
class Cell
      int data
3
      int index
      Cell left
5
      Cell right
6
      Cell self
      method setData(int value)
9
          data ^= value
10
11
      method setIndex(int i)
12
          index ^= i
13
      method setLeft(Cell cell)
          left <=> cell
15
16
      method setRight(Cell cell)
17
          right <=> cell
18
19
      method setSelf(Cell cell)
20
          self <=> cell
21
22
      method append(Cell cell)
23
24
          if right = nil & cell != nil then // If current cell does not have a right neighbour
               right <=> cell
                                                 // Set new cell as right neighbour of current cell
25
26
27
               local Cell selfCopy = nil
               copy Cell self selfCopy
                                                 // Copy reference to current cell
28
                                                 // Set current cell as left neighbour of newly
               call right::setLeft(selfCopy)
29
                   added right neighbour
               delocal Cell selfCopy = nil
30
31
               local int cellIndex = index + 1
               call right::setIndex(cellIndex) // Set cell index in newly added right neightbour
33
                   of current cell
               delocal int cellIndex = index + 1
34
           else skip
35
36
          fi right != nil & cell = nil
37
          if right != nil then
38
39
              call right::append(cell)
                                               // Keep searching for empty right neighbour
          else skip
40
          fi right != nil
41
42
  class DoublyLinkedList
43
      Cell head
45
      int length
46
47
      method appendCell(Cell cell)
          if head = nil & cell != nil then
48
               head <=> cell
49
           else skip
50
          fi head != nil & cell = nil
51
52
53
          if head != nil then
54
               call head::append(cell)
           else skip
55
          fi head != nil
56
57
          length += 1
58
59
  class Program
61
     DoublyLinkedList list
```

```
int listLength
63
64
       method main()
65
            new DoublyLinkedList list
            listLength += 10
66
67
68
            local int x = 0
            from x = 0 do skip
69
70
            loop
71
72
                 local Cell cell = nil
   new Cell cell
73
74
                      local Cell cellCopy = nil
75
                      copy Cell cell cellCopy
76
77
                      call cell::setSelf(cellCopy)
                      delocal Cell cellCopy = nil
78
79
80
                      call cell::setData(x)
call list::appendCell(cell)
                 delocal Cell cell = nil
81
            x += 1
until x = listLength
82
83
84
            delocal int x = listLength
```

## Doubly Linked List.pal

1	;; pendulum pal file				60		XORI	\$10 1
2	top:	BRA	sta	rt	61		ADDI	\$8 1
3	l_r_list:	DATA	0		62		EXCH	\$19 \$17
4	l_r_listLength:	DATA	0		63		XOR	\$18 \$19
	<del>-</del>		977					
5	l_Program_vt:	DATA			64		EXCH	\$19 \$17
6	l_DoublyLinkedList_vt:	DATA	759		65		RL	\$9 1
7	1_Cell_vt:	DATA	223		66		EXCH	\$10 \$1
8		DATA	252		67		ADDI	\$1 -1
9		DATA	281		68		EXCH	\$11 \$1
10		DATA	312		69		ADDI	\$1 -1
1		DATA	343		70		EXCH	\$12 \$1
11								
12		DATA	374		71		ADDI	\$1 -1
13	l_malloc_top:	BRA	l_ma	alloc_	bot 72		EXCH	\$14 \$1
14	l_malloc:	SWAPBR	\$2		73		ADDI	\$1 -1
15		NEG	\$2		74		EXCH	\$16 \$1
16		ADDI	\$9 2	2	75		ADDI	\$1 -1
17		XOR	\$8 5		76		EXCH	\$17 \$1
18		ADDI	\$1 :	L	77		ADDI	\$1 -1
19		EXCH	\$6	\$1	78		EXCH	\$18 \$1
20		ADDI	\$1 :	l	79		ADDI	\$1 -1
21		EXCH	\$7 :		80		EXCH	\$20 \$1
		EXCH	\$2 :				ADDI	\$1 -1
22					81			
23		ADDI	\$1 -		82		EXCH	\$21 \$1
24		BRA	l_ma	alloc1	83		ADDI	\$1 -1
25		ADDI	\$1 :	l	84		EXCH	\$22 \$1
26		EXCH	\$2 5	\$1	85		ADDI	\$1 -1
27		EXCH	\$7 5	\$1	86		EXCH	\$23 \$1
28		ADDI	\$1 -		87		ADDI	\$1 -1
29		EXCH	\$6 5		88		BRA	l_malloc1
30		ADDI	\$1 -	-1	89		ADDI	\$1 1
31		XOR	\$8 :	\$0	90		EXCH	\$23 \$1
32		ADDI	\$9 -	-2	91		ADDI	\$1 1
33	l_malloc_bot:	BRA	1 ma	alloc_	top 92		EXCH	\$22 \$1
34	l_malloc1_top:	BRA			_bot 93		ADDI	\$1 1
ł	<u></u> marrocr_cop.	ADDI	\$1		94		EXCH	
35								\$21 \$1
36		EXCH	\$2 :		95		ADDI	\$1 1
37		SUB	\$17	\$8	96		EXCH	\$20 \$1
38		XOR	\$17	\$4	97		ADDI	\$1 1
39	l_malloc1:	SWAPBR	\$2		98		EXCH	\$18 \$1
40		NEG	\$2		99		ADDI	\$1 1
41		EXCH	\$2 :	<b>†</b> 1	100		EXCH	\$17 \$1
1								
42		ADDI	\$1 -		101		ADDI	\$1 1
43		XOR	\$17		102		EXCH	\$16 \$1
44		ADD	\$17	\$8	103		ADDI	\$1 1
45		EXCH	\$19	\$17	104		EXCH	\$14 \$1
46		XOR		\$19	105		ADDI	\$1 1
47		EXCH		\$17	106		EXCH	\$12 \$1
48		XOR	\$13		107		ADDI	\$1 1
49		SUB	\$13		108		EXCH	\$11 \$1
50	cmp_top_8:	BGEZ			ot_9109		ADDI	\$1 1
51		XORI	\$14	1	110		EXCH	\$10 \$1
52	cmp_bot_9:	BGEZ	\$13	cmp_t	op_8111		RR	\$9 1
53		XOR		\$14	112		ADDI	\$8 -1
54	cmp_bot_9_i:	BGEZ	\$13		113		XORI	\$10 1
04		2022	γ±3					
	cmp_top_8_i	W05-	A1.	1		l_o_assert_true:	BRA	l_o_assert
55		XORI	\$14	Τ		l_o_test_false:	BRA	l_o_test
56	cmp_top_8_i:	BGEZ	\$13		116	cmp_top_12:	BEQ	\$18 \$0
	cmp_bot_9_i					cmp_bot_13		
57		ADD	\$13	\$7	117		XORI	\$20 1
58		XOR	\$13		118	cmp_bot_13:	BEQ	\$18 \$0
59	l_o_test:	BEQ	\$10		110	cmp_top_12	z	+ -
59		25	<b>Υ</b> Ι Ο	70	110		VOR	¢11 ¢20
	l_o_test_false				119	I	XOR	\$11 \$20

120	cmp_bot_13_i:	BEQ	\$18 \$0	183		EXCH	\$12 \$17
	cmp_top_12_i			184		ADD	\$6 \$9
121		XORI	\$20 1	185	l_i_assert:	BNE	\$11 \$0
122	cmp_top_12_i:	BEQ	\$18 \$0		l_i_assert_true		
	cmp_bot_13_i			186		EXCH	\$12 \$17
123		BEQ	\$11 \$0	187		SUB	\$6 \$9
	l_i_test_false			188	cmp_top_14:	BEQ	\$6 \$12
124		XORI	\$11 1		cmp_bot_15		
125		ADD	\$6 \$18	189		XORI	\$21 1
126		SUB	\$18 \$6	190	cmp_bot_15:	BEQ	\$6 \$12
127		EXCH	\$12 \$6	İ	cmp_top_14		
128		EXCH	\$12 \$17	191	cmp_top_16:	BNE	\$12 \$0
129		XOR	\$12 \$6		cmp_bot_17		
130		XORI	\$11 1	192	_	XORI	\$22 1
131	l_i_assert_true:	BRA	l_i_assert	193	cmp_bot_17:	BNE	\$12 \$0
132		BRA	l_i_test		cmp_top_16		
133		ADDI	\$8 1	194	1 — 1 —	ORX	\$23 \$21 \$22
134		RL	\$9 1	195		XOR	\$11 \$23
135		EXCH	\$10 \$1	196		ORX	\$23 \$21 \$22
136		ADDI	\$1 -1	197	cmp_bot_17_i:	BNE	\$12 \$0
137		EXCH	\$11 \$1	10.	cmp_top_16_i		710 70
138		ADDI	\$1 -1	198	Cmp_cop_1 0_1	XORI	\$22 1
139		EXCH	\$12 \$1	199	cmp_top_16_i:	BNE	\$12 \$0
140		ADDI	\$1 -1	199	cmp_bot_17_i	DNE	712 70
			\$14 \$1	200	=	DEO	\$6 \$12
141		EXCH		200	cmp_bot_15_i:	BEQ	\$6 \$12
142		ADDI	\$1 -1		cmp_top_14_i		401 1
143		EXCH	\$16 \$1	201		XORI	\$21 1
144		ADDI	\$1 -1	202	cmp_top_14_i:	BEQ	\$6 \$12
145		EXCH	\$17 \$1		cmp_bot_15_i		
146		ADDI	\$1 -1	203		ADD	\$6 \$9
147		EXCH	\$18 \$1	204		EXCH	\$12 \$17
148		ADDI	\$1 -1	205	l_o_assert:	BNE	\$10 \$0
149		EXCH	\$20 \$1		l_o_assert_true		
150		ADDI	\$1 -1	206		XOR	\$15 \$9
151		EXCH	\$21 \$1	207		SUB	\$15 \$7
152		ADDI	\$1 -1	208	cmp_top_10:	BGEZ	\$15 cmp_bot_11
153		EXCH	\$22 \$1	209		XORI	\$16 1
154		ADDI	\$1 -1	210	cmp_bot_11:	BGEZ	\$15 cmp_top_10
155		EXCH	\$23 \$1	211		XOR	\$10 \$16
156		ADDI	\$1 -1	212	cmp_bot_11_i:	BGEZ	\$15
157		BRA	l_malloc1		cmp_top_10_i		
158		ADDI	\$1 1	213		XORI	\$16 1
159		EXCH	\$23 \$1	214	cmp_top_10_i:	BGEZ	\$15
160		ADDI	\$1 1		cmp_bot_11_i		
161		EXCH	\$22 \$1	215		ADD	\$15 \$7
162		ADDI	\$1 1	216		XOR	\$15 \$9
163		EXCH	\$21 \$1	217	l_malloc1_bot:	BRA	l_malloc1_top
164		ADDI	\$1 1		 l_setData_2_top:	BRA	-
165		EXCH	\$20 \$1		l_setData_2_bot		
166		ADDI	\$1 1	219		ADDI	\$1 1
167		EXCH	\$18 \$1	220		EXCH	\$2 \$1
168		ADDI	\$1 1	221		EXCH	\$6 \$1
169		EXCH	\$17 \$1	222		ADDI	\$1 -1
170		ADDI	\$1 1	223		EXCH	\$3 \$1
171		EXCH	\$16 \$1	224		ADDI	\$1 -1
172		ADDI	\$1 1		l_setData_2:	SWAPBR	
173		EXCH	\$14 \$1	226		NEG	\$2
174		ADDI	\$1 1	227		ADDI	\$1 1
175		EXCH	\$12 \$1	228		EXCH	\$3 \$1
176		ADDI	\$1 1	229		ADDI	\$1 1
177		EXCH	\$11 \$1	230		EXCH	\$6 \$1
177		ADDI	\$1 1	230		EXCH	\$2 \$1
		EXCH	\$10 \$1			ADDI	
179				232			\$1 -1
180		RR	\$9 1	233		ADD	\$7 \$3 \$7 3
181		ADDI	\$8 -1	234		ADDI	\$7 2
182	I	XOR	\$12 \$6	235		EXCH	\$8 \$7

236		ADDI	\$7	-2	298	xo	OR	\$9	\$8
237		SUB	\$7	\$3	299	XC	OR	\$8	\$9
238		EXCH	\$9	\$6	300	EX	KCH	\$9	\$6
239		XOR	\$8	\$9	301	AD	DD	\$7	\$3
240		EXCH	\$9	\$6	302	AD	DDI	\$7	4
241		ADD		\$3	303				\$7
242		ADDI	\$7		304			\$7	
		EXCH		\$7		SU		\$7	
243					305			۱ ډ	ŞS
244		ADDI		-2	306	l_setLeft_4_bot: BR	RA.		
245		SUB	\$7	\$3		l_setLeft_4_top			
246	l_setData_2_bot:	BRA			307	l_setRight_5_top: BR	RA		
	l_setData_2_top					l_setRight_5_bot			
247	<pre>l_setIndex_3_top:</pre>	BRA			308	AD	DDI	\$1	1
	l_setIndex_3_bot				309	EX	KCH	\$2	\$1
248		ADDI	\$1	1	310	EX	KCH	\$6	\$1
249		EXCH	\$2	\$1	311	AD		\$1	
250		EXCH		\$1	312			\$3	
251		ADDI		-1	313			\$1	
		EXCH		\$1			WAPBR		-1
252					314				
253		ADDI		-1	315	NE		\$2	_
254	l_setIndex_3:	SWAPBR			316			\$1	
255		NEG	\$2		317			\$3	
256		ADDI	\$1		318			\$1	
257		EXCH	\$3	\$1	319	EX		\$6	
258		ADDI	\$1		320			\$2	\$1
259		EXCH	\$6	\$1	321	AD	DDI	\$1	-1
260		EXCH	\$2	\$1	322	AD	DD	\$7	\$3
261		ADDI	\$1	-1	323	AD	DDI	\$7	5
262		ADD	\$7	\$3	324	EX	KCH	\$8	\$7
263		ADDI	\$7	3	325	AD	DDI	\$7	-5
264		EXCH	\$8	\$7	326	su	JB	\$7	\$3
265		ADDI		-3	327	EX	KCH	\$9	\$6
266		SUB		\$3	328	swap_19:			\$9
267		EXCH		\$6	329	xo			\$8
268		XOR		\$9	330	xo		\$8	
1		EXCH		\$6				\$9	
269					331				
270		ADD		\$3	332	AD		\$7	
271		ADDI	\$7		333			\$7	
272		EXCH		\$7	334				\$7
273		ADDI		-3	335			\$7	
274		SUB	\$7	\$3	336	SU		\$7	\$3
275	<pre>1_setIndex_3_bot:</pre>	BRA			337	l_setRight_5_bot: BR	RA		
	l_setIndex_3_top					l_setRight_5_top			
276	<pre>l_setLeft_4_top:     l_setLeft_4_bot</pre>	BRA			338	<pre>l_setSelf_6_top: BR</pre>	RA		
277		ADDI	\$1	1	339	AD	DDI	\$1	1
278		EXCH	\$2	\$1	340	EX	KCH	\$2	\$1
279		EXCH	\$6	\$1	341	EX	KCH	\$6	\$1
280		ADDI	\$1	-1	342	AD	DDI	\$1	-1
281		EXCH	\$3	\$1	343	EX	KCH	\$3	\$1
282		ADDI	\$1	-1	344			\$1	
283	l_setLeft_4:	SWAPBR			345		WAPBR		
284	_	NEG	\$2		346	NE		\$2	
285		ADDI	\$1		347			\$1	1
286		EXCH		\$1	348			\$3	
287		ADDI	\$1		349			\$1	
288		EXCH		\$1	350			\$6	
289		EXCH		\$1	351			\$2	
		ADDI		-1				\$2 \$1	
290				-ı \$3	352				
291		ADD			353			\$7	
292		ADDI	\$7		354			\$7	
293		EXCH		\$7	355			\$8	
294		ADDI		-4	356			\$7	
295		SUB		\$3	357	SU		\$7	
296		EXCH		\$6	358			\$9	
297	swap_18:	XOR	\$8	\$9	359	swap_20:	OR	\$8	\$9

360		XOR	\$9	\$8	413		ADDI	\$8 5
361	,	XOR	\$8	\$9	414		EXCH	\$9 \$8
		EXCH			1		ADDI	
362				\$6	415			\$8 -5
363	2	ADD	\$7	\$3	416		SUB	\$8 \$3
364	;	ADDI	\$7	6	417	test_21:	BEQ	\$7 \$0
365		EXCH		\$7		test_false_23	~	
						test_raise_23		
366	2	ADDI	\$7	-6	418		XORI	\$7 1
367	:	SUB	\$7	\$3	419		ADD	\$8 \$3
368		BRA			420		ADDI	\$8 5
308		DKA						
	l_setSelf_6_top				421		EXCH	\$9 \$8
369	l_append_7_top:	BRA	1_a	append_7_	_bo#22		ADDI	\$8 -5
370		ADDI	\$1		423		SUB	\$8 \$3
371	1	EXCH	\$2	ŞI	424		EXCH	\$10 \$6
372	1	EXCH	\$6	\$1	425	swap_31:	XOR	\$9 \$10
373	;	ADDI	\$1	-1	426		XOR	\$10 \$9
1								
374		EXCH	\$3	\$1	427		XOR	\$9 \$10
375	i	ADDI	\$1	-1	428		EXCH	\$10 \$6
376	l_append_7:	SWAPBR	\$2		429		ADD	\$8 \$3
1					1			
377		NEG	\$2		430		ADDI	\$8 5
378	·	ADDI	\$1	1	431		EXCH	\$9 \$8
379	1	EXCH	\$3	\$1	432		ADDI	\$8 -5
380		ADDI	\$1		433		SUB	\$8 \$3
1						11511 25		
381		EXCH	\$6		434	localBlock_35:	XOR	\$8 \$1
382	1	EXCH	\$2	\$1	435		XOR	\$9 \$0
383	:	ADDI	\$1		436		EXCH	\$9 \$1
384		ADD		\$3	437		ADDI	\$1 -1
385	į	ADDI	\$8	5	438		EXCH	\$9 \$8
386	1	EXCH	\$9	\$8	439		ADD	\$10 \$3
		ADDI		-5			ADDI	
387					440			\$10 6
388	:	SUB	\$8	\$3	441		EXCH	\$11 \$10
389	cmp_top_25:	BNE	\$9	\$0	442		ADDI	\$10 -6
	cmp_bot_26				443		SUB	\$10 \$3
390	-	XORI	\$10	) 1	444	copy_32:	XOR	\$9 \$11
391	cmp_bot_26:	BNE	\$9	\$0	445		ADDI	\$11 1
	cmp_top_25				446		EXCH	\$12 \$11
			A1-	1 66				
392	·	EXCH		1 \$6	447		ADDI	\$12 1
393	cmp_top_27:	BEQ	\$11	1 \$0	448		EXCH	\$12 \$11
	cmp_bot_28				449		ADDI	\$11 -1
20.4	_	XORI	ċ11	0 1			ADD	
394				2 1	450			\$10 \$3
395	cmp_bot_28:	BEQ	\$1:	1 \$0	451		ADDI	\$10 6
	cmp_top_27				452		EXCH	\$11 \$10
396		ANDX	¢11	3 \$10 \$12			ADDI	\$10 -6
					1			
397		BEQ	ŞΙ	3 \$0	454		SUB	\$10 \$3
	f_bot_30				455		EXCH	\$9 \$8
398	·	XORI	\$14	4 1	456		ADD	\$9 \$3
399		BEQ		3 \$0	457		ADDI	\$9 5
599		ההה	<b>Υ</b> Τ .	J 40				
	f_top_29				458		EXCH	\$10 \$9
400	]	XOR	\$7	\$14	459		ADDI	\$9 -5
401	f_bot_30_i:	BEQ	\$11	3 \$0	460		SUB	\$9 \$3
101		z	T ± 4	- + -				
	f_top_29_i				461		XOR	\$11 \$10
402		XORI	\$14	4 1	462	loadMetAdd_33:	EXCH	\$12 \$11
403	f_top_29_i:	BEQ	\$13	3 \$0	463		ADDI	\$12 2
	_	~					EXCH	
	f_bot_30_i				464			\$13 \$12
404	<u> </u>	ANDX	\$13	3 \$10 \$12	2 465		XOR	\$14 \$13
405	cmp_bot_28_i:	BEQ	\$11	1 \$0	466		EXCH	\$13 \$12
	cmp_top_27_i				467		ADDI	\$12 -2
		VODT	٠	n 1				
406		XORI		2 1	468		EXCH	\$12 \$11
407	cmp_top_27_i:	BEQ	\$11	1 \$0	469		ADD	\$9 \$3
	cmp_bot_28_i				470		ADDI	\$9 5
408		EXCH	\$11	1 \$6	471		EXCH	\$10 \$9
409	cmp_bot_26_i:	BNE	Ş9	\$0	472		ADDI	\$9 -5
	cmp_top_25_i				473		SUB	\$9 \$3
410		XORI	\$10	0 1	474		EXCH	\$3 \$1
1				\$0	-			
411		BNE	マラ	Ų	475		ADDI	\$1 -1
	cmp_bot_26_i				476		EXCH	\$6 \$1
412		ADD	\$8	\$3	477		ADDI	\$1 -1
1					,			

1				1			440 440
478		EXCH	\$8 \$1	544		EXCH	\$13 \$12
479		ADDI	\$1 -1	545		XOR	\$14 \$13
480		EXCH	\$11 \$1	546		EXCH	\$13 \$12
1				- 1			
481		ADDI	\$1 -1	547		ADDI	\$12 -1
482		ADDI	\$14 -481	548		EXCH	\$12 \$11
	1 24.			- 1			
483	l_jmp_34:	SWAPBR		549		ADD	\$9 \$3
484		NEG	\$14	550		ADDI	\$9 5
485		ADDI	\$14 481	551		EXCH	\$10 \$9
				- 1			
486		ADDI	\$1 1	552		ADDI	\$9 -5
487		EXCH	\$11 \$1	553		SUB	\$9 \$3
1				- 1			
488		ADDI	\$1 1	554		EXCH	\$3 \$1
489		EXCH	\$8 \$1	555		ADDI	\$1 -1
400		ADDI				EXCH	\$6 \$1
490			\$1 1	556			
491		EXCH	\$6 \$1	557		ADDI	\$1 -1
492		ADDI	\$1 1	558		EXCH	\$8 \$1
1				- 1			
493		EXCH	\$3 \$1	559		ADDI	\$1 -1
494		ADD	\$9 \$3	560		EXCH	\$11 \$1
495		ADDI	\$9 5	561		ADDI	\$1 -1
1							
496		EXCH	\$10 \$9	562		ADDI	\$14 -561
497		ADDI	\$9 -5	563	1_jmp_37:	SWAPBR	\$14
498		SUB	\$9 \$3	564		NEG	\$14
499		EXCH	\$12 \$11	565		ADDI	\$14 561
1		ADDI	\$12 2	- 1		ADDI	
500				566			\$1 1
501		EXCH	\$13 \$12	567		EXCH	\$11 \$1
502		XOR	\$14 \$13	568		ADDI	\$1 1
503		EXCH	\$13 \$12	569		EXCH	\$8 \$1
504		ADDI	\$12 -2	570		ADDI	\$1 1
1	londMo+7dd 22 i.			- 1			\$6 \$1
505	loadMetAdd_33_i:	EXCH	\$12 \$11	571		EXCH	
506		XOR	\$11 \$10	572		ADDI	\$1 1
507		ADD	\$9 \$3	573		EXCH	\$3 \$1
1				- 1			
508		ADDI	\$9 5	574		ADD	\$9 \$3
509		EXCH	\$10 \$9	575		ADDI	\$9 5
				- 1			
510		ADDI	\$9 -5	576		EXCH	\$10 \$9
511		SUB	\$9 \$3	577		ADDI	\$9 -5
512		ADDI	\$1 1	578		SUB	\$9 \$3
				- 1			
513		EXCH	\$9 \$1	579		EXCH	\$12 \$11
514		XOR	\$9 \$0	580		ADDI	\$12 1
	logolDlogk 25 i.	XOR				EXCH	
515	localBlock_35_i:		\$8 \$1	581			\$13 \$12
516		ADD	\$9 \$3	582		XOR	\$14 \$13
517		ADDI	\$9 3	583		EXCH	\$13 \$12
518		EXCH	\$10 \$9	584		ADDI	\$12 -1
519		ADDI	\$9 -3	585	loadMetAdd_36_i:	EXCH	\$12 \$11
520		SUB	\$9 \$3	586		XOR	\$11 \$10
1				- 1			
521		XORI	\$11 1	587		ADD	\$9 \$3
522		XOR	\$12 \$10	588		ADDI	\$9 5
				- 1			
523		ADD	\$12 \$11	589		EXCH	\$10 \$9
524	localBlock_38:	XOR	\$8 \$1	590		ADDI	\$9 -5
525		XOR	\$13 \$12	591		SUB	\$9 \$3
526		EXCH	\$13 \$1	592		ADD	\$9 \$3
527		ADDI	\$1 -1	593		ADDI	\$9 3
1		SUB	\$12 \$11	594			
528						EXCH	\$10 \$9
529		XOR	\$12 \$10	595		ADDI	\$9 -3
530		XORI	\$11 1	596		SUB	\$9 \$3
1							
531		ADD	\$9 \$3	597		XORI	\$11 1
532		ADDI	\$9 3	598		XOR	\$12 \$10
533		EXCH	\$10 \$9	599		ADD	\$12 \$11
534		ADDI	\$9 -3	600		ADDI	\$1 1
535		SUB	\$9 \$3	601		EXCH	\$13 \$1
536		ADD	\$9 \$3	602		XOR	\$13 \$12
1				- 1	1 1 1 1 2 2 4		
537		ADDI	\$9 5	603	localBlock_38_i:	XOR	\$8 \$1
538		EXCH	\$10 \$9	604		SUB	\$12 \$11
539		ADDI	\$9 -5	605		XOR	\$12 \$10
540		SUB	\$9 \$3	606		XORI	\$11 1
541		XOR	\$11 \$10	607		ADD	\$9 \$3
	1			- 1			
542	loadMetAdd_36:	EXCH	\$12 \$11	608		ADDI	\$9 3
543		ADDI	\$12 1	609		EXCH	\$10 \$9
,				'			

610		ADDI	\$9 -3		f_top_51		
611		SUB	\$9 \$3	660		XOR	\$7 \$11
612		XORI	\$7 1	661	f_bot_52_i:	BEQ	\$10 \$0
613	assert_true_22:	BRA	assert_24		f_top_51_i		
614	test_false_23:	BRA	test_21	662		XORI	\$11 1
615	assert_24:	BNE	\$7 \$0	663	f_top_51_i:	BEQ	\$10 \$0
	assert_true_22			İ	f_bot_52_i		
616		ADD	\$8 \$3	664	cmp_bot_50_i:	BEQ	\$9 \$0
617		ADDI	\$8 5		cmp_top_49_i		
618		EXCH	\$9 \$8	665		XORI	\$10 1
619		ADDI	\$8 -5	666	cmp_top_49_i:	BEQ	\$9 \$0
620		SUB	\$8 \$3	İ	cmp_bot_50_i		
621	cmp_top_39:	BEQ	\$9 \$0	667	<del>-</del>	ADD	\$8 \$3
	cmp_bot_40			668		ADDI	\$8 5
622		XORI	\$10 1	669		EXCH	\$9 \$8
623	cmp_bot_40:	BEQ	\$9 \$0	670		ADDI	\$8 -5
	cmp_top_39	_		671		SUB	\$8 \$3
624	1 = 1 =	EXCH	\$11 \$6	672	test_45:	BEQ	\$7 \$0
625	cmp_top_41:	BNE	\$11 \$0		test_false_47	_	
	cmp_bot_42			673		XORI	\$7 1
626		XORI	\$12 1	674		ADD	\$8 \$3
627	cmp_bot_42:	BNE	\$11 \$0	675		ADDI	\$8 5
	cmp_top_41			676		EXCH	\$9 \$8
628	1 = 1 1 =	ANDX	\$13 \$10 \$12	677		ADDI	\$8 -5
629	f_top_43:	BEQ	\$13 \$0	678		SUB	\$8 \$3
	f bot 44	~		679		XOR	\$10 \$9
630		XORI	\$14 1	680	loadMetAdd_53:	EXCH	\$11 \$10
631	f_bot_44:	BEQ	\$13 \$0	681		ADDI	\$11 5
	f_top_43			682		EXCH	\$12 \$11
632		XOR	\$7 \$14	683		XOR	\$13 \$12
633	f_bot_44_i:	BEQ	\$13 \$0	684		EXCH	\$12 \$11
	f_top_43_i			685		ADDI	\$11 -5
634		XORI	\$14 1	686		EXCH	\$11 \$10
635	f_top_43_i:	BEQ	\$13 \$0	687		ADD	\$8 \$3
	f_bot_44_i			688		ADDI	\$8 5
636		ANDX	\$13 \$10 \$12	689		EXCH	\$9 \$8
637	cmp_bot_42_i:	BNE	\$11 \$0	690		ADDI	\$8 -5
	cmp_top_41_i			691		SUB	\$8 \$3
638		XORI	\$12 1	692		EXCH	\$3 \$1
639	cmp_top_41_i:	BNE	\$11 \$0	693		ADDI	\$1 -1
0.40	cmp_bot_42_i	<b>511011</b>	A11 AC	694		EXCH	\$6 \$1
640	l+ 40 ÷	EXCH	\$11 \$6	695		ADDI	\$1 -1
641	cmp_bot_40_i: cmp_top_39_i	BEQ	\$9 \$0	696 697		EXCH ADDI	\$10 \$1 \$1 -1
642	Cmp_cop_39_1	XORI	\$10 1	698		ADDI	\$13 -697
643	cmp_top_39_i:	BEQ	\$9 \$0		l_jmp_54:	SWAPBR	
040	cmp_bot_40_i	<u>_</u>	7 7 7 0	700	JK · ·	NEG	\$13
644	<u>-</u>	ADD	\$8 \$3	701		ADDI	\$13 697
645		ADDI	\$8 5	702		ADDI	\$1 1
646		EXCH	\$9 \$8	703		EXCH	\$10 \$1
647		ADDI	\$8 -5	704		ADDI	\$1 1
648		SUB	\$8 \$3	705		EXCH	\$6 \$1
649		ADD	\$8 \$3	706		ADDI	\$1 1
650		ADDI	\$8 5	707		EXCH	\$3 \$1
651		EXCH	\$9 \$8	708		ADD	\$8 \$3
652		ADDI	\$8 -5	709		ADDI	\$8 5
653		SUB	\$8 \$3	710		EXCH	\$9 \$8
654	cmp_top_49:	BEQ	\$9 \$0	711		ADDI	\$8 -5
	cmp_bot_50			712		SUB	\$8 \$3
655		XORI	\$10 1	713		EXCH	\$11 \$10
656	cmp_bot_50:	BEQ	\$9 \$0	714		ADDI	\$11 5
	cmp_top_49			715		EXCH	\$12 \$11
657	f_top_51:	BEQ	\$10 \$0	716		XOR	\$13 \$12
	f_bot_52		***	717		EXCH	\$12 \$11
658	C 1	XORI	\$11 1	718	1 126	ADDI	\$11 -5
659	f_bot_52:	BEQ	\$10 \$0	719	loadMetAdd_53_i:	EXCH	\$11 \$10

720		XOR	\$10 \$9	775		XORI	\$10 1
721		ADD	\$8 \$3	776	cmp_bot_64:	BNE	\$9 \$0
722		ADDI	\$8 5		cmp_top_63		
723		EXCH	\$9 \$8	777		EXCH	\$11 \$6
724		ADDI	\$8 -5	778	cmp_top_65:	BEQ	\$11 \$0
725		SUB	\$8 \$3		cmp_bot_66	_	
726		XORI	\$7 1	779	1 = 1 = 1 1	XORI	\$12 1
727	assert_true_46:	BRA	assert_48	780	cmp_bot_66:	BEQ	\$11 \$0
728	test_false_47:	BRA	test_45		cmp_top_65	2	722 70
729	assert_48:	BNE	\$7 \$0	781	Cb_cop_03	ANDX	\$13 \$10 \$12
123	assert_true_46	DILL	Ψ7 Ψ0	782	f_top_67:	BEQ	\$13 \$0
730	assert_true_40	ADD	\$8 \$3	102	_ *-	DEQ	713 70
				=00	f_bot_68	VODT	Ċ1 / 1
731		ADDI	\$8 5	783	6.160	XORI	\$14 1
732		EXCH	\$9 \$8	784	f_bot_68:	BEQ	\$13 \$0
733		ADDI	\$8 -5		f_top_67		
734		SUB	\$8 \$3	785		XOR	\$7 \$14
735	cmp_top_55:	BEQ	\$9 \$0	786		BEQ	\$13 \$0
	cmp_bot_56				f_top_67_i		
736		XORI	\$10 1	787		XORI	\$14 1
737	cmp_bot_56:	BEQ	\$9 \$0	788	f_top_67_i:	BEQ	\$13 \$0
	cmp_top_55				f_bot_68_i		
738	f_top_57:	BEQ	\$10 \$0	789		ANDX	\$13 \$10 \$12
	f_bot_58			790	cmp_bot_66_i:	BEQ	\$11 \$0
739		XORI	\$11 1		cmp_top_65_i		
740	f_bot_58:	BEQ	\$10 \$0	791		XORI	\$12 1
	f_top_57			792	cmp_top_65_i:	BEQ	\$11 \$0
741		XOR	\$7 \$11		cmp_bot_66_i	_	
742	f_bot_58_i:	BEQ	\$10 \$0	793		EXCH	\$11 \$6
	 f_top_57_i	_		794	cmp_bot_64_i:	BNE	\$9 \$0
743		XORI	\$11 1		cmp_top_63_i		
744	f_top_57_i:	BEQ	\$10 \$0	795		XORI	\$10 1
	f_bot_58_i	2	720 70	796	cmp_top_63_i:	BNE	\$9 \$0
745	cmp_bot_56_i:	BEQ	\$9 \$0		cmp_bot_64_i		73 70
743	cmp_top_55_i	DEQ	<b>Ψ 9 Ψ 0</b>	797	CIIID_DOC_04_1	ADD	\$8 \$3
746	Cmp_cop_33_1	XORI	\$10 1	798		ADDI	\$8 2
747	amp top 55 i.	BEQ	\$9 \$0	799		EXCH	\$9 \$8
141	cmp_top_55_i:	PEQ	79 70			ADDI	\$8 -2
<b>7</b> 40	cmp_bot_56_i	3.DD	¢0 ¢3	800			
748		ADD	\$8 \$3	801		SUB	\$8 \$3
749		ADDI	\$8 5	802	test_59:	BEQ	\$7 \$0
750		EXCH	\$9 \$8		test_false_61		±0.4
751		ADDI	\$8 -5	803		XORI	\$7 1
752		SUB	\$8 \$3	804		ADD	\$8 \$3
753	l_append_7_bot:	BRA	l_append_7_t	-		ADDI	\$8 2
754	<pre>l_appendCell_1_top:</pre>	BRA		806		EXCH	\$9 \$8
	l_appendCell_1_bot			807		ADDI	\$8 -2
755		ADDI	\$1 1	808		SUB	\$8 \$3
756		EXCH	\$2 \$1	809		EXCH	\$10 \$6
757		EXCH	\$6 \$1	810	swap_69:	XOR	\$9 \$10
758		ADDI	\$1 -1	811		XOR	\$10 \$9
759		EXCH	\$3 \$1	812		XOR	\$9 \$10
760		ADDI	\$1 -1	813		EXCH	\$10 \$6
761	<pre>l_appendCell_1:</pre>	SWAPBR	\$2	814		ADD	\$8 \$3
762		NEG	\$2	815		ADDI	\$8 2
763		ADDI	\$1 1	816		EXCH	\$9 \$8
764		EXCH	\$3 \$1	817		ADDI	\$8 -2
765		ADDI	\$1 1	818		SUB	\$8 \$3
766		EXCH	\$6 \$1	819		XORI	\$7 1
767		EXCH	\$2 \$1		assert_true_60:	BRA	assert_62
768		ADDI	\$1 -1		test_false_61:	BRA	test_59
769		ADD	\$8 \$3		assert_62:	BNE	\$7 \$0
770		ADDI	\$8 2		assert_true_60		
771		EXCH	\$9 \$8	823		ADD	\$8 \$3
772		ADDI	\$8 -2	824		ADDI	\$8 2
773		SUB	\$8 \$3	825		EXCH	\$9 \$8
774	cmp_top_63:	BNE	\$9 \$0	826		ADDI	\$8 -2
114	cmp_bot_64	<b></b>	42 YU	827		SUB	\$8 \$3
ļ	CP_20C_04			941		202	70 70

828	cmp_top_70:	BEQ	\$9 \$0	874		ADD	\$8 \$3
	cmp_bot_71			875		ADDI	\$8 2
829		XORI	\$10 1	876		EXCH	\$9 \$8
830	cmp_bot_71:	BEQ	\$9 \$0	877		ADDI	\$8 -2
	cmp_top_70	2	1 - 1 -	878		SUB	\$8 \$3
831	Cmp_cop_70	EXCH	\$11 \$6	879	test_76:	BEQ	\$7 \$0
	amn + an 72.			019		PEQ	Ş / Ş U
832	cmp_top_72:	BNE	\$11 \$0		test_false_78	WORT	67 1
	cmp_bot_73			880		XORI	\$7 1
833		XORI	\$12 1	881		ADD	\$8 \$3
834	cmp_bot_73:	BNE	\$11 \$0	882		ADDI	\$8 2
	cmp_top_72			883		EXCH	\$9 \$8
835		ANDX	\$13 \$10 \$12	884		ADDI	\$8 -2
836	f_top_74:	BEQ	\$13 \$0	885		SUB	\$8 \$3
	f_bot_75	_		886		XOR	\$10 \$9
837		XORI	\$14 1	887	loadMetAdd 84:	EXCH	\$11 \$10
838	f_bot_75:	BEQ	\$13 \$0	888	10441001144_01.	ADDI	\$11 5
030		DDQ	Ψ13 Ψ0			EXCH	
	f_top_74	WOD	67 614	889			\$12 \$11
839	6.1 . 75 .	XOR	\$7 \$14	890		XOR	\$13 \$12
840	f_bot_75_i:	BEQ	\$13 \$0	891		EXCH	\$12 \$11
	f_top_74_i			892		ADDI	\$11 -5
841		XORI	\$14 1	893		EXCH	\$11 \$10
842	f_top_74_i:	BEQ	\$13 \$0	894		ADD	\$8 \$3
	f_bot_75_i			895		ADDI	\$8 2
843		ANDX	\$13 \$10 \$12	896		EXCH	\$9 \$8
844	cmp_bot_73_i:	BNE	\$11 \$0	897		ADDI	\$8 -2
	cmp_top_72_i			898		SUB	\$8 \$3
845	op_cop_/11	XORI	\$12 1	899		EXCH	\$3 \$1
846	cmp_top_72_i:	BNE	\$11 \$0	900		ADDI	\$1 -1
040	= =	DINE	411 40			EXCH	\$6 \$1
0.45	cmp_bot_73_i	EVOII	¢11 ¢C	901			
847	1 . 21 .	EXCH	\$11 \$6	902		ADDI	\$1 -1
848	cmp_bot_71_i:	BEQ	\$9 \$0	903		EXCH	\$10 \$1
	cmp_top_70_i			904		ADDI	\$1 -1
849		XORI	\$10 1	905		ADDI	¢12 _00/
049				903		ADDI	\$13 -904
850	cmp_top_70_i:	BEQ	\$9 \$0	906	l_jmp_85:	SWAPBR	
	<pre>cmp_top_70_i:     cmp_bot_71_i</pre>				1_jmp_85:		
	= =			906	1_jmp_85:	SWAPBR	\$13
850	= =	BEQ	\$9 \$0	906 907	1_jmp_85:	SWAPBR NEG	\$13 \$13
850 851	= =	BEQ ADD	\$9 \$0 \$8 \$3	906 907 908	1_jmp_85:	SWAPBR NEG ADDI	\$13 \$13 \$13 904
850 851 852	= =	BEQ ADD ADDI	\$9 \$0 \$8 \$3 \$8 2	906 907 908 909	1_jmp_85:	SWAPBR NEG ADDI ADDI	\$13 \$13 \$13 904 \$1 1
850 851 852 853 854	= =	BEQ ADD ADDI EXCH	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2	906 907 908 909 910	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1
850 851 852 853 854 855	= =	ADD ADDI EXCH ADDI SUB	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3	906 907 908 909 910 911 912	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH ADDI EXCH	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1
850 851 852 853 854 855 856	= =	BEQ  ADD  ADDI  EXCH  ADDI  SUB  ADD	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3	906 907 908 909 910 911 912 913	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI ADDI	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1 \$1 1
850 851 852 853 854 855 856 857	= =	BEQ ADD ADDI EXCH ADDI SUB ADD ADDI	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3 \$8 \$3 \$8 \$2	906 907 908 909 910 911 912 913 914	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1
850 851 852 853 854 855 856 857 858	= =	BEQ  ADD  ADDI  EXCH  ADDI  SUB  ADD  ADDI  ADDI  EXCH	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3 \$8 \$3 \$8 \$3 \$8 \$3	906 907 908 909 910 911 912 913 914 915	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$8 \$3
850 851 852 853 854 855 856 857 858	= =	BEQ ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3 \$8 \$2 \$9 \$8 \$8 2	906 907 908 909 910 911 912 913 914 915	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI ADDI ADDI ADDI ADDI	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$8 \$3 \$8 2
850 851 852 853 854 855 856 857 858 859 860	cmp_bot_71_i	BEQ ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI EXCH ADDI SUB	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3 \$8 \$2 \$9 \$8 \$8 2 \$9 \$8 \$8 -2 \$8 \$3	906 907 908 909 910 911 912 913 914 915 916	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADD ADDI EXCH	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$8 \$3 \$8 2 \$9 \$8
850 851 852 853 854 855 856 857 858	cmp_bot_71_i cmp_top_80:	BEQ ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3 \$8 \$2 \$9 \$8 \$8 2	906 907 908 909 910 911 912 913 914 915 916 917	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADD ADDI EXCH ADDI	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$8 \$3 \$8 2 \$9 \$8 \$8 -2
850 851 852 853 854 855 856 857 858 859 860 861	cmp_bot_71_i	BEQ ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$8 \$3 \$8 9 \$0	906 907 908 909 910 911 912 913 914 915 916 917 918	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADD ADDI EXCH ADD SUB	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3
850 851 852 853 854 855 856 857 858 859 860 861	cmp_bot_71_i  cmp_top_80:  cmp_bot_81	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$9 \$8 \$8 -2 \$9 \$8 \$9 \$0 \$10 1	906 907 908 909 910 911 912 913 914 915 916 917 918 919	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADD ADDI EXCH ADD SUB EXCH	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$11 \$10
850 851 852 853 854 855 856 857 858 859 860 861	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:</pre>	BEQ ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$8 \$3 \$8 9 \$0	906 907 908 909 910 911 912 913 914 915 916 917 918 919 920	1_jmp_85:	SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADD SUB EXCH ADDI	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$11 \$10 \$11 5
850 851 852 853 854 855 856 857 858 859 860 861	cmp_bot_71_i  cmp_top_80:  cmp_bot_81	BEQ  ADD  ADDI  EXCH  ADDI  SUB  ADD  ADDI  EXCH  ADDI  EXCH  ADDI  EXCH  ADDI  SUB  BEQ  XORI  BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$9 \$8 \$8 -2 \$9 \$8 \$9 \$0 \$10 1	906 907 908 909 910 911 912 913 914 915 916 917 918 919	1_jmp_85:	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADD SUB EXCH ADDI SUB EXCH ADDI EXCH ADDI SUB EXCH ADDI	\$13 \$13 \$13 904 \$1 1 \$10 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$11 \$10
850 851 852 853 854 855 856 857 858 859 860 861	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80 f_top_82:</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$9 \$8 \$8 -2 \$9 \$8 \$9 \$0 \$10 1	906 907 908 909 910 911 912 913 914 915 916 917 918 919 920	1_jmp_85:	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI	\$13 \$13 \$13 \$13 \$04 \$1 \$1 \$10 \$1 \$1 \$1 \$6 \$1 \$1 \$3 \$1 \$8 \$3 \$1 \$8 \$3 \$8 \$2 \$9 \$8 \$8 \$2 \$9 \$8 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 858 859 860 861 862	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80</pre>	BEQ  ADD  ADDI  EXCH  ADDI  SUB  ADD  ADDI  EXCH  ADDI  EXCH  ADDI  EXCH  ADDI  SUB  BEQ  XORI  BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0	906 907 908 909 910 911 912 913 914 915 916 917 918 920 921 922	1_jmp_85:	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADD SUB EXCH ADDI SUB EXCH ADDI EXCH ADDI SUB EXCH ADDI	\$13 \$13 \$13 \$13 \$04 \$1 \$1 \$10 \$1 \$1 \$1 \$6 \$1 \$1 \$1 \$3 \$1 \$8 \$3 \$1 \$8 \$3 \$8 \$2 \$9 \$8 \$8 \$2 \$9 \$8 \$8 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 858 859 860 861 862	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80 f_top_82:</pre>	BEQ  ADD  ADDI  EXCH  ADDI  SUB  ADD  ADDI  EXCH  ADDI  EXCH  ADDI  EXCH  ADDI  SUB  BEQ  XORI  BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0	906 907 908 909 910 911 913 914 915 916 917 918 919 920 921 922	1_jmp_85:	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI	\$13 \$13 \$13 \$13 \$04 \$1 \$1 \$10 \$1 \$1 \$1 \$6 \$1 \$1 \$3 \$1 \$8 \$3 \$1 \$8 \$3 \$8 \$2 \$9 \$8 \$8 \$2 \$9 \$8 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 858 860 861 862 863	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80 f_top_82:</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI BEQ  BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0	906 907 908 909 910 911 913 914 915 916 917 918 919 920 921 922 923	<pre>1_jmp_85:  loadMetAdd_84_i:</pre>	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADD SUB EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH ADDI	\$13 \$13 \$13 \$13 \$14 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 858 860 861 862 863	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80 f_top_82:     f_bot_83</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$11 1	906 907 908 909 910 911 912 913 914 915 916 917 918 920 921 922 923 924		SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 \$13 \$13 \$14 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 858 860 861 862 863	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80  f_top_82:     f_bot_83  f_bot_83:</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$2 2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$11 1 \$10 \$0	906 907 908 909 910 911 912 913 914 915 916 917 920 921 922 923 924 925 926		SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH	\$13 \$13 \$13 \$13 \$14 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 858 860 861 862 863 864	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80  f_top_82:     f_bot_83  f_bot_83:     f_top_82</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$10 \$0 \$11 1 \$10 \$0 \$7 \$11	906 907 908 909 910 911 912 913 914 915 916 917 918 920 921 922 923 924 925 926 927		SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH XOR EXCH ADDI EXCH ADDI	\$13 \$13 \$13 \$13 \$04 \$1 \$1 \$10 \$1 \$1 \$6 \$1 \$1 \$3 \$1 \$8 \$3 \$1 \$8 \$2 \$9 \$8 \$2 \$8 \$8 \$2 \$8 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 858 860 861 862 863 864	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81:     cmp_bot_81:     cmp_top_80 f_top_82:     f_bot_83:     f_top_82 f_bot_83: f_top_82</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$2 2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$11 1 \$10 \$0	906 907 908 909 910 911 912 913 914 915 916 917 920 921 922 923 924 925 926 927 928		SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 \$13 \$13 \$04 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 860 861 862 863 864 865 866 867	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80  f_top_82:     f_bot_83  f_bot_83:     f_top_82</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$11 1 \$10 \$0 \$7 \$11 \$10 \$0	906 907 908 909 910 911 912 913 914 915 916 917 920 921 922 923 924 925 926 927 928 929 930		SWAPBR NEG ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI EXCH	\$13 \$13 \$13 \$13 \$04 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 861 862 863 864 865 866 867 868	<pre>cmp_top_80:     cmp_bot_81:     cmp_bot_81:     cmp_top_80 f_top_82:     f_bot_83 f_bot_83:     f_top_82 f_bot_83_i:     f_top_82_i</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$10 \$0 \$11 1 \$10 \$0 \$7 \$11 \$10 \$0 \$11 1	906 907 908 909 910 911 912 913 914 916 917 920 921 922 923 924 925 926 927 928 929 930 931		SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 \$13 \$13 \$14 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 860 861 862 863 864 865 866 867	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81:     cmp_bot_81:     cmp_top_80 f_top_82:     f_bot_83  f_bot_83:     f_top_82  f_bot_83_i:     f_top_82_i  f_top_82_i: </pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$11 1 \$10 \$0 \$7 \$11 \$10 \$0	906 907 908 909 910 911 912 913 914 915 916 917 920 921 922 923 924 925 926 927 928 929 930 931 932		SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH SUB EXCH SUB EXCH SUB EXCH SUB	\$13 \$13 \$13 \$13 \$13 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 868 861 862 863 864 865 866 867 868	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80 f_top_82:     f_bot_83  f_bot_83:     f_top_82  f_bot_83_i:     f_top_82_i  f_top_82_i:     f_bot_83_i:     f_top_82_i</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$2 \$9 \$8 \$3 \$8 2 \$9 \$9 \$0 \$10 1 \$9 \$0 \$10 \$0 \$11 1 \$10 \$0 \$7 \$11 \$10 \$0 \$11 1 \$10 \$0	906 907 908 909 910 911 912 913 914 915 916 917 920 921 922 923 924 925 926 927 928 929 930 931 932 933	loadMetAdd_84_i:	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH SOR ADD SUB XORI	\$13 \$13 \$13 \$13 \$14 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 861 862 863 864 865 866 867 868	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80 f_top_82:     f_bot_83  f_bot_83:     f_top_82  f_bot_83_i:     f_top_82_i  f_top_82_i:     f_top_82_i:     f_bot_83_i:     cmp_bot_83_i:     f_top_82_i:     f_bot_83_i:     f_top_82_i:     f_bot_83_i:     cmp_bot_81_i:</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$10 \$0 \$11 1 \$10 \$0 \$7 \$11 \$10 \$0 \$11 1	906 907 908 909 910 911 912 913 914 915 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934	<pre>loadMetAdd_84_i: assert_true_77:</pre>	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH EXCH EXCH	\$13 \$13 \$13 \$13 \$14 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 860 861 862 863 864 865 866 867 868 869 870	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80 f_top_82:     f_bot_83  f_bot_83:     f_top_82  f_bot_83_i:     f_top_82_i  f_top_82_i:     f_bot_83_i:     f_top_82_i</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$11 1 \$10 \$0 \$7 \$11 \$10 \$0 \$11 1 \$10 \$0 \$11 1 \$10 \$0 \$11 5 \$10 \$0 \$10 906 907 908 909 910 911 912 913 914 915 916 917 922 923 924 925 926 927 928 929 930 931 932 933 934	<pre>loadMetAdd_84_i:  assert_true_77: test_false_78:</pre>	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH SOR EXCH ADDI EXCH SOR ADD ADDI EXCH SOR ADD ADDI EXCH SOR ADD ADDI EXCH ADDI EXCH SOR ADD ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 \$13 \$13 \$14 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1	
850 851 852 853 854 855 856 867 868 867 868 869 870 871	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80  f_top_82:     f_bot_83  f_bot_83:     f_top_82  f_bot_83_i:     f_top_82_i  f_top_82_i:     f_bot_83_i:     cmp_top_80_i</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$11 1 \$10 \$0 \$7 \$11 \$10 \$0 \$11 1 \$10 \$0 \$10 1 \$10 \$0 \$10 \$0	906 907 908 909 910 911 912 913 914 915 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934	<pre>loadMetAdd_84_i:  assert_true_77: test_false_78: assert_79:</pre>	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH EXCH EXCH	\$13 \$13 \$13 \$13 \$14 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1
850 851 852 853 854 855 856 857 860 861 862 863 864 865 866 867 868 869 870	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80  f_top_82:     f_bot_83  f_bot_83:     f_top_82  f_bot_83_i:     f_top_82_i  f_top_82_i:     cmp_bot_81_i:     cmp_top_80_i  cmp_top_80_i:</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$11 1 \$10 \$0 \$7 \$11 \$10 \$0 \$11 1 \$10 \$0 \$11 1 \$10 \$0 \$11 5 \$10 \$0 \$10 906 907 908 909 910 911 912 913 914 915 916 917 922 923 924 925 926 927 928 929 930 931 932 933 934	<pre>loadMetAdd_84_i:  assert_true_77: test_false_78:</pre>	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH SOR ADD ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH EXCH EXCH	\$13 \$13 \$13 \$13 \$14 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1	
850 851 852 853 854 855 856 867 868 867 868 869 870 871	<pre>cmp_bot_71_i  cmp_top_80:     cmp_bot_81  cmp_bot_81:     cmp_top_80  f_top_82:     f_bot_83  f_bot_83:     f_top_82  f_bot_83_i:     f_top_82_i  f_top_82_i:     f_bot_83_i:     cmp_top_80_i</pre>	BEQ  ADD ADDI EXCH ADDI SUB ADD ADDI EXCH ADDI EXCH ADDI SUB BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ  XORI BEQ	\$9 \$0 \$8 \$3 \$8 2 \$9 \$8 \$8 -2 \$8 \$3 \$8 \$2 \$9 \$8 \$8 -2 \$9 \$8 \$8 -2 \$8 \$3 \$9 \$0 \$10 1 \$9 \$0 \$11 1 \$10 \$0 \$7 \$11 \$10 \$0 \$11 1 \$10 \$0 \$10 1 \$10 \$0 \$10 \$0	906 907 908 909 910 911 912 913 914 915 916 917 922 923 924 925 926 927 928 929 930 931 932 933 934	<pre>loadMetAdd_84_i:  assert_true_77: test_false_78: assert_79:</pre>	SWAPBR NEG ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH SOR EXCH ADDI EXCH SOR ADD ADDI EXCH SOR ADD ADDI EXCH SOR ADD ADDI EXCH ADDI EXCH SOR ADD ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 \$13 \$13 \$14 \$10 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1

938		ADDI	\$8 2	995		ADDI	\$1 1
939		EXCH	\$9 \$8	996		EXCH	\$8 \$1
940		ADDI	\$8 -2	997	obj_con_90_i:	ADDI	\$8 -4
941		SUB	\$8 \$3	998		ADDI	\$1 1
	+ 06.						
942	cmp_top_86:	BEQ	\$9 \$0	999		EXCH	\$3 \$1
	cmp_bot_87			1000		ADD	\$6 \$3
943		XORI	\$10 1	1001		ADDI	\$6 2
944	cmp_bot_87:	BEQ	\$9 \$0	1002		XORI	\$8 4
	cmp_top_86			1003		EXCH	\$8 \$7
945	f_top_88:	BEQ	\$10 \$0	1004		ADDI	\$7 1
010	f_bot_89	z	710 70	1005		XORI	\$8 1
0.40	1_000_03	VODT	č11 1				
946		XORI	\$11 1	1006		EXCH	\$8 \$7
947		BEQ	\$10 \$0	1007	obj_con_90_bot:	ADDI	\$7 -1
	f_top_88			1008		EXCH	\$7 \$6
948		XOR	\$7 \$11	1009		ADDI	\$6 -2
949	f_bot_89_i:	BEQ	\$10 \$0	1010		SUB	\$6 \$3
	 f_top_88_i	_		1011		ADD	\$6 \$3
950	1_00p_00_1	XORI	\$11 1	1012		ADDI	\$6 3
				- 1			
951	f_top_88_i:	BEQ	\$10 \$0	1013		EXCH	\$7 \$6
	f_bot_89_i			1014		ADDI	\$6 -3
952	cmp_bot_87_i:	BEQ	\$9 \$0	1015		SUB	\$6 \$3
	cmp_top_86_i			1016		XORI	\$8 10
953		XORI	\$10 1	1017		ADD	\$7 \$8
954	cmp_top_86_i:	BEQ	\$9 \$0	1018		XORI	\$8 10
334		DHQ	42 40				\$6 \$3
	cmp_bot_87_i		40.40	1019		ADD	
955		ADD	\$8 \$3	1020		ADDI	\$6 3
956		ADDI	\$8 2	1021		EXCH	\$7 \$6
957		EXCH	\$9 \$8	1022		ADDI	\$6 -3
958		ADDI	\$8 -2	1023		SUB	\$6 \$3
959		SUB		1024	localBlock_113:	XOR	\$6 \$1
960		ADD		1025		XOR	\$7 \$0
		ADDI					
961				1026		EXCH	\$7 \$1
962		EXCH		1027		ADDI	\$1 -1
963		ADDI	\$7 -3	1028		XORI	\$7 1
964		SUB	\$7 \$3	1029	entry_91:	BEQ	\$7 \$0
965		XORI	\$9 1	İ	assert_93		
966		ADD		1030		EXCH	\$8 \$6
					+ OF -		
967		XORI	\$9 1	1031	cmp_top_95:	BNE	\$8 \$0
968		ADD	\$7 \$3		cmp_bot_96		
969		ADDI	\$7 3	1032		XORI	\$9 1
970		EXCH	\$8 \$7	1033	cmp_bot_96:	BNE	\$8 \$0
971		ADDI	\$7 -3		cmp_top_95		
972		SUB	\$7 \$3	1034	f_top_97:	BEQ	\$9 \$0 f_bot_98
973	l_appendCell_1_bot:	BRA		1035		XORI	\$10 1
313		Ditti			£ 00.		
	l_appendCell_1_top			1036	f_bot_98:	BEQ	\$9 \$0 f_top_97
	l_main_0_top:	BRA	l_main_0_bot			XOR	\$7 \$10
975		ADDI	\$1 1	1038		BEQ	\$9 \$0
976		EXCH	\$2 \$1		f_top_97_i		
977		EXCH	\$3 \$1	1039		XORI	\$10 1
978		ADDI	\$1 -1	1040	f_top_97_i:	BEQ	\$9 \$0
979	l_main_0:	SWAPBR		-	f_bot_98_i	_	
980		NEG	\$2	1041	cmp_bot_96_i:	BNE	\$8 \$0
				1041		DNE	70 70
981		ADDI	\$1 1		cmp_top_95_i		40.1
982		EXCH	\$3 \$1	1042		XORI	\$9 1
983		EXCH	\$2 \$1	1043	cmp_top_95_i:	BNE	\$8 \$0
984		ADDI	\$1 -1	ĺ	cmp_bot_96_i		
985		EXCH	\$3 \$1	1044		EXCH	\$8 \$6
986		ADDI	\$1 -1	1045		EXCH	\$8 \$6
987	obj_con_90:	ADDI	\$8 4	1046		ADD	\$9 \$3
	05_001_50.						
988		EXCH	\$8 \$1	1047		ADDI	\$9 3
989		ADDI	\$1 -1	1048		EXCH	\$10 \$9
990		EXCH	\$7 \$1	1049		ADDI	\$9 -3
991		ADDI	\$1 -1	1050		SUB	\$9 \$3
992		BRA	l_malloc	1051	cmp_top_109:	BNE	\$8 \$10
993		ADDI	\$1 1		cmp_bot_110		
994		EXCH	\$7 \$1	1052	_	XORI	\$11 1
334	I		T / Y ±	1002			7 + + +

1053	cmp_bot_110:	BNE	\$8 \$10	1112		ADDI	\$11 1
1055	cmp_top_109	DRE	AO ATO	1113		EXCH	\$12 \$11
1054	f_top_111:	BEQ	\$11 \$0	1114		ADDI	\$12 1
	f_bot_112	~		1115		EXCH	\$12 \$11
1055		XORI	\$12 1	1116		ADDI	\$11 -1
1056	f_bot_112:	BEQ	\$11 \$0	1117		EXCH	\$11 \$8
	f_top_111			1118		EXCH	\$10 \$9
1057		XOR	\$7 \$12	1119		EXCH	\$10 \$8
1058	f_bot_112_i:	BEQ	\$11 \$0	1120		XOR	\$11 \$10
	f_top_111_i			1121	loadMetAdd_101:	EXCH	\$12 \$11
1059		XORI	\$12 1	1122		ADDI	\$12 4
1060	f_top_111_i:	BEQ	\$11 \$0	1123		EXCH	\$13 \$12
	f_bot_112_i		40 410	1124		XOR	\$14 \$13
1061	cmp_bot_110_i:	BNE	\$8 \$10	1125		EXCH	\$13 \$12
1062	cmp_top_109_i	XORI	\$11 1	1126 1127		ADDI EXCH	\$12 -4 \$12 \$11
1062	cmp_top_109_i:	BNE	\$8 \$10	1127		EXCH	\$10 \$8
1003	cmp_bot_110_i	DNE	40 ATO	1129		EXCH	\$3 \$1
1064	6mp_200_110_1	ADD	\$9 \$3	1130		ADDI	\$1 -1
1065		ADDI	\$9 3	1131		EXCH	\$8 \$1
1066		EXCH	\$10 \$9	1132		ADDI	\$1 -1
1067		ADDI	\$9 -3	1133		EXCH	\$6 \$1
1068		SUB	\$9 \$3	1134		ADDI	\$1 -1
1069		EXCH	\$8 \$6	1135		EXCH	\$9 \$1
1070	test_92:	BNE	\$7 \$0 e	exit_9 <b>4</b> 136		ADDI	\$1 -1
1071	localBlock_108:	XOR	\$8 \$1	1137		EXCH	\$11 \$1
1072		XOR	\$9 \$0	1138		ADDI	\$1 -1
1073		EXCH	\$9 \$1	1139		ADDI	\$14 -1138
1074		ADDI	\$1 -1	1140	l_jmp_102:	SWAPBR	
1075		EXCH	\$3 \$1	1141		NEG	\$14
1076		ADDI	\$1 -1	1142		ADDI	\$14 1138
1077		EXCH	\$8 \$1	1143		ADDI	\$1 1
1078		ADDI EXCH	\$1 -1 \$6 \$1	1144		EXCH ADDI	\$11 \$1 \$1 1
1079 1080		ADDI	\$1 -1	1145 1146		EXCH	\$9 \$1
1080	obj_con_99:	ADDI	\$10 8	1146		ADDI	\$1 1
1081	05_05_	EXCH	\$10 \$1	1148		EXCH	\$6 \$1
1083		ADDI	\$1 -1	1149		ADDI	\$1 1
1084		EXCH	\$9 \$1	1150		EXCH	\$8 \$1
1085		ADDI	\$1 -1	1151		ADDI	\$1 1
1086		BRA	l_mallo	C 1152		EXCH	\$3 \$1
1087		ADDI	\$1 1	1153		EXCH	\$10 \$8
1088		EXCH	\$9 \$1	1154		EXCH	\$12 \$11
1089		ADDI	\$1 1	1155		ADDI	\$12 4
1090		EXCH	\$10 \$1	1156		EXCH	\$13 \$12
1091	obj_con_99_i:	ADDI	\$10 -8	1157		XOR	\$14 \$13
1092		ADDI	\$1 1	1158		EXCH	\$13 \$12
1093		EXCH	\$6 \$1	1159		ADDI	\$12 -4
1094		ADDI	\$1 1	1160	loadMetAdd_101_i:	EXCH	\$12 \$11
1095		EXCH	\$8 \$1	1161		XOR	\$11 \$10
1096		ADDI	\$1 1	1162		EXCH	\$10 \$8
1097 1098		EXCH	\$3 \$1 \$10 5	1163		ADDI EXCH	\$1 1 \$10 \$1
1098		EXCH	\$10 5	1164 1165		XOR	\$10 \$1
1100		ADDI	\$9 1	1165	localBlock_103_i:	XOR	\$9 \$1
1101		XORI	\$10 1	1167		EXCH	\$9 \$8
1102		EXCH	\$10 \$9	1168		XOR	\$10 \$9
1103	obj_con_99_bot:	ADDI	\$9 -1	1169	loadMetAdd_104:	EXCH	\$11 \$10
1104		EXCH	\$9 \$8	1170		ADDI	\$11 0
1105	localBlock_103:	XOR	\$9 \$1	1171		EXCH	\$12 \$11
1106		XOR	\$10 \$0	1172		XOR	\$13 \$12
1107		EXCH	\$10 \$1	1173		EXCH	\$12 \$11
1108		ADDI	\$1 -1	1174		ADDI	\$11 0
1109		EXCH	\$10 \$9	1175		EXCH	\$11 \$10
1110		EXCH	\$11 \$8	1176		EXCH	\$9 \$8
1111	copy_100:	XOR	\$10 \$11	. 1177		EXCH	\$3 \$1

1170	ADDI	\$1 -1	1244		EXCH	\$3 \$1
1178 1179	EXCH	\$8 \$1	1244		ADD	\$9 \$3
1180	ADDI	\$1 -1	1245		ADDI	\$9 2
1181	EXCH	\$6 \$1	1247		EXCH	\$10 \$9
1182	ADDI	\$1 -1	1248		ADDI	\$9 -2
1183	EXCH	\$10 \$1	1249		SUB	\$9 \$3
1184	ADDI	\$1 -1	1250		EXCH	\$12 \$11
1185	ADDI	\$13 -1184	1251		ADDI	\$12 0
1186 l_jmp_105:	SWAPBR		1251		EXCH	\$13 \$12
1187 1 1187	NEG	\$13	1253		XOR	\$14 \$13
1188	ADDI	\$13 1184	1254		EXCH	\$13 \$12
1189	ADDI	\$1 1	1255		ADDI	\$12 0
1190	EXCH	\$10 \$1	1256	loadMetAdd_106_i:	EXCH	\$12 \$11
1191	ADDI	\$1 1	1257	10441101144_100_1.	XOR	\$11 \$10
1192	EXCH	\$6 \$1	1258		ADD	\$9 \$3
1193	ADDI	\$1 1	1259		ADDI	\$9 2
1194	EXCH	\$8 \$1	1260		EXCH	\$10 \$9
1195	ADDI	\$1 1	1261		ADDI	\$9 -2
1196	EXCH	\$3 \$1	1262		SUB	\$9 \$3
1197	EXCH	\$9 \$8	1263		ADDI	\$1 1
1198	EXCH	\$11 \$10	1264		EXCH	\$9 \$1
1199	ADDI	\$11 0	1265		XOR	\$9 \$0
1200	EXCH	\$12 \$11	1266	localBlock_108_i:	XOR	\$8 \$1
1201	XOR	\$13 \$12	1267		EXCH	\$8 \$6
1202	EXCH	\$12 \$11	1268		XORI	\$9 1
1203	ADDI	\$11 0	1269		ADD	\$8 \$9
1204 loadMetAdd_104_i:	EXCH	\$11 \$10	1270		XORI	\$9 1
1205	XOR	\$10 \$9	1271		EXCH	\$8 \$6
1206	EXCH	\$9 \$8	1272	assert_93:	BRA	entry_91
1207	ADD	\$9 \$3	1273	exit_94:	BRA	test_92
1208	ADDI	\$9 2	1274		XORI	\$7 1
1209	EXCH	\$10 \$9	1275		ADD	\$7 \$3
1210	ADDI	\$9 -2	1276		ADDI	\$7 3
1211	SUB	\$9 \$3	1277		EXCH	\$8 \$7
1212	XOR	\$11 \$10	1278		ADDI	\$7 -3
1213 loadMetAdd_106:	EXCH	\$12 \$11	1279		SUB	\$7 \$3
1214	ADDI	\$12 0	1280		ADDI	\$1 1
1215	EXCH	\$13 \$12	1281		EXCH	\$9 \$1
1216	XOR	\$14 \$13	1282		XOR	\$9 \$8
1217	EXCH	\$13 \$12	1283	localBlock_113_i:	XOR	\$6 \$1
1218	ADDI	\$12 0	1284		ADD	\$7 \$3
1219	EXCH	\$12 \$11	1285		ADDI	\$7 3
1220	ADD	\$9 \$3	1286		EXCH	\$8 \$7
1221	ADDI	\$9 2	1287		ADDI	\$7 -3
1222	EXCH	\$10 \$9	1288		SUB	\$7 \$3
1223	ADDI	\$9 -2	1289	l_main_0_bot:	BRA	l_main_0_top
1224	SUB	\$9 \$3		start:	BRA	top
1225	EXCH	\$3 \$1	1291		START	44 1000
1226	ADDI	\$1 -1	1292		ADDI	\$4 1338
1227	EXCH	\$6 \$1	1293		XOR	\$5 \$4
1228	ADDI	\$1 -1	1294		ADDI	\$5 10
1229	EXCH	\$8 \$1	1295		XOR	\$7 \$5
1230	ADDI	\$1 -1	1296		ADDI	\$4 10
1231	EXCH	\$11 \$1	1297		ADDI	\$4 -1
1232	ADDI ADDI	\$1 -1 \$14 -1232	1298		EXCH ADDI	\$7 \$4 \$4 1
1233	SWAPBR		1299		ADDI	\$4 1 \$4 -10
1234 l_jmp_107: 1235	NEG	\$14	1300 1301		XOR	\$4 -10 \$1 \$5
1235	ADDI	\$14 1232	1301		ADDI	\$1 2048
1237	ADDI	\$14 1232	1302		ADDI	\$1 -4
1238	EXCH	\$11 \$1	1304		XOR	\$3 \$1
1239	ADDI	\$1 1	1304		XORI	\$6 3
1240	EXCH	\$8 \$1	1306		EXCH	\$6 \$3
1241	ADDI	\$1 1	1307		ADDI	\$1 -1
1241	EXCH	\$6 \$1	1308		EXCH	\$3 \$1
1243	ADDI	\$1 1	1309		ADDI	\$1 -1
ı			1			

1310	BRA	1_m	ain_0	1325		EXCH	\$6	\$7
1311	ADDI	\$1	1	1326		XORI	\$7	2
1312	EXCH	\$3	\$1	1327		ADDI	\$3	-2
1313	ADDI	\$3	1	1328		ADDI	\$3	-1
1314	ADDI	\$3	1	1329		ADDI	\$1	1
1315	EXCH	\$6	\$3	1330		EXCH	\$6	\$3
1316	XORI	\$7	1	1331		XORI	\$6	3
1317	EXCH	\$6	\$7	1332		XOR	\$3	\$1
1318	XORI	\$7	1	1333		ADDI	\$1	4
1319	ADDI	\$3	-1	1334		ADDI	\$1	-2048
1320	ADDI	\$3	-1	1335		XOR	\$1	\$5
1321	ADDI	\$3	1	1336		ADDI	\$5	-10
1322	ADDI	\$3	2	1337		XOR	\$5	\$4
1323	EXCH	\$6	\$3	1338		ADDI	\$4	-1338
1324	XORI	\$7	2	1339	finish:	FINISH		

## RTM.rplpp

```
class Cell
       Cell self
2
 3
       Cell right
       Cell left
5
       int data
 6
       method getLeft(Cell cell)
           right <=> cell
9
       method getRight(Cell cell)
10
11
           left <=> cell
12
       method getSelf(Cell cell)
13
14
           self <=> cell
15
       method getSymbol(int symbol)
16
17
           symbol <=> data
18
19
  class RTM
       Cell tapeHead
20
       \mathbf{int}\,[\,]\ q1
21
22
       int[] q2
       int[] s1
23
24
       int[] s2
25
       int SLASH
       int LEFT
26
27
       int RIGHT
28
       int BLANK
       int state
29
30
       int Qs
       int Qf
31
       int symbol
32
33
       int PC_MAX
       int pc
34
35
       method initLiterals()
36
           // Initialize string literals
37
38
           SLASH += 9999
           LEFT += 9998
39
           RIGHT += 9997
40
41
           BLANK += 9996
42
43
           // Set max program counter
           PC\_MAX += 7
44
45
46
       method initRules()
47
           // Initialize transition rule arrays
48
           new int[8] q1
49
           new int[8] q2
           new int[8] s1
50
51
           new int[8] s2
52
           \ensuremath{//} Define transition rules for binary number incrementation
53
54
           q1[0] += 1
           s1[0] += BLANK
55
           s2[0] += BLANK
56
57
           q2[0] += 2
58
59
           q1[1] += 2
           s1[1] += SLASH
60
           s2[1] += RIGHT
61
62
           q2[1] += 3
63
```

```
q1[2] += 3
            s1[2] += 0
 65
            s2[2] += 1
 66
 67
            q2[2] += 4
 68
 69
            q1[3] += 3
            s1[3] += 1
 70
            s2[3] += 0
 71
 72
            q2[3] += 2
 73
            q1[4] += 3
 74
 75
            s1[4] += BLANK
 76
            s2[4] += BLANK
 77
            q2[4] += 4
 78
 79
            q1[5] += 4
 80
            s1[5] += SLASH
            s2[5] += LEFT
 81
 82
            q2[5] += 5
 83
            q1[6] += 5
 84
            s1[6] += 0
 85
 86
            s2[6] += 0
            q2[6] += 4
 87
 88
 89
            q1[7] += 5
            s1[7] += BLANK
 90
 91
            s2[7] += BLANK
            q2[7] += 6
 92
 93
        method initTape()
 94
            local Cell cell0 = nil
 95
 96
            local Cell cell1 = nil
 97
            local Cell cell2 = nil
            local Cell cell3 = nil
 98
 99
            local Cell cell4 = nil
100
101
            // Init cells
            new Cell cell0
102
            new Cell cell1
103
104
            new Cell cell2
            new Cell cell3
105
            new Cell cell4
106
107
            // Write 1 1 0 1 on tape symbol += BLANK
108
109
            uncall cell0::getSymbol(symbol)
110
            symbol += 1
111
112
            uncall cell1::getSymbol(symbol)
            symbol += 1
113
114
            uncall cell2::getSymbol(symbol)
115
            symbol += 1
116
            uncall cell4::getSymbol(symbol)
117
            // Set tape head
118
            tapeHead <=> cell0
119
120
121
            // Set self pointers
            copy Cell tapeHead cell0
122
123
            uncall tapeHead::getSelf(cell0)
            copy Cell cell1 cell0
124
125
            uncall cell1::getSelf(cell0)
126
            copy Cell cell2 cell0
            uncall cell2::getSelf(cell0)
127
128
            copy Cell cell3 cell0
            uncall cell3::getSelf(cell0)
129
```

```
130
            copy Cell cell4 cell0
131
            uncall cell4::getSelf(cell0)
132
133
            // Link cell 3 and 4
            copy Cell cell3 cell0
134
135
            uncall cell4::getLeft(cell0)
136
            uncall cell3::getRight(cell4)
137
            // Link cell 2 and 3
            copy Cell cell2 cell0
139
            uncall cell3::getLeft(cell0)
140
            uncall cell2::getRight(cell3)
141
142
143
            // Link cell1 and cell 2
            copy Cell cell1 cell0
144
            uncall cell2::getLeft(cell0)
145
146
            uncall cell1::getRight(cell2)
147
148
            // Link tapeHead and cell 1
            copy Cell tapeHead cell0
149
            uncall cell1::getLeft(cell0)
150
151
            uncall tapeHead::getRight(cell1)
152
            delocal Cell cell4 = nil
153
            delocal Cell cell3 = nil
154
            delocal Cell cell2 = nil
155
            delocal Cell cell1 = nil
156
            delocal Cell cell0 = nil
157
158
159
        method init()
            // Prepare for simulation
160
            call initLiterals()
161
162
            call initRules()
163
            call initTape()
164
165
            // Init pc, start and finishing state
            state += 1
166
167
            Qs += 1
            Qf += 6
168
169
170
            // Start simulation
            call simulate()
171
172
        method simulate()
173
            from state = Os do
174
                                                        // Fetch current symbol
175
                call tapeHead::getSymbol(symbol)
176
                call inst()
                uncall tapeHead::getSymbol(symbol)
                                                       // Zero-clear symbol
177
178
                pc += 1
                                                        // Increment pc
179
                if pc = PC_MAX then
   pc ^= PC_MAX
180
                                                        // Reset pc
181
                else skip
182
183
                fi pc = 0
            loop skip
184
            until state = Qf
185
186
        method inst()
187
            if state = q1[pc] & symbol = s1[pc] then
                                                            // Symbol rule:
188
                state += q2[pc]-q1[pc]
189
                                                            // set state to q2[pc]
                symbol += s2[pc]-s1[pc]
                                                            // set symbol to s2[pc]
190
191
            else skip
192
            fi state = q2[pc] & symbol = s2[pc]
            if state = q1[pc] & s1[pc] = SLASH then
                                                            // Move rule:
193
194
                state += q2[pc]-q1[pc]
                                                            // set state to q2[pc]
                if s2[pc] = RIGHT then
195
```

```
call moveRight()
                                                             // Move tape head right
197
                else skip
                 fi s2[pc] = RIGHT
198
199
                 if s2[pc] = LEFT then
                                                             // Move tape head left
                     uncall moveRight()
200
201
                 else skip
202
                fi s2[pc] = LEFT
            else skip
203
            fi state = q2[pc] & s1[pc] = SLASH
204
205
        method moveRight()
206
            local Cell right = nil
207
            local Cell tmp = nil
208
209
            uncall tapeHead::getSymbol(symbol)
                                                     // Put symbol back in current cell
                                                     // Get right neighbour
210
            call tapeHead::getRight(right)
211
212
            if right = nil & symbol = BLANK then
                symbol ^= BLANK
                                                    // Zero clear symbol
213
214
                new Cell right
                                                     // Init new neighbour
215
                copy Cell right tmp
                                                     // Copy reference to self
                uncall right::getSelf(tmp)
                                                     // Store self reference
216
                uncall right::getLeft(tapeHead) // Set tape head as left of new cell
217
218
                right <=> tapeHead
            else
219
220
                 call right::getLeft(tmp)
                                                     // Get copy of tape head reference
                uncopy Cell tmp tapeHead
                                                     // Clear reference to tape head
221
222
                 if tapeHead = nil & symbol = BLANK then
223
                     call tmp::getSelf(tapeHead) // rev: set self pointer
uncopy Cell tmp tapeHead // rev: new self pointer
224
225
                                                     // rev: new left neighbour
226
                     delete Cell tmp
                     symbol ^= BLANK
227
228
                 else skip
                                                     // In reverse:
229
                fi tmp = nil
                                                     // Allocate new left if current is nil
230
231
                 uncall right::getLeft(tmp)
                                                     // Put tape head reference back
                tapeHead <=> right
232
233
                 call tapeHead::getRight(right) // Get right of new tape head
                call tapeHead::getSymbol(symbol) // Get symbol of new tape head
234
            fi right = nil
235
236
            uncall tapeHead::getRight(right)
                                                    // Set right neighbour
237
            delocal Cell right = nil
238
            delocal Cell tmp = nil
239
240
241
   class Program
        RTM bni
242
243
244
        method main()
            // This program contains a RTM implementing
245
246
            \ensuremath{//} incrementation of a non-negative n-bit binary number by 1 (modulo 2n).
            // The tape is initialized with \mid b \mid 1Â \mid 1 \mid 0 \mid 0 \mid and after execution,
247
            // the tape is left with \mid b \mid 0 \mid 0Â \mid 1 \mid 1 \mid
248
249
            new RTM bni
            call bni::init()
250
```

## RTM.pal

1	;; pendulum pal file			61		XOR	\$13 \$9
2	top:	BRA	start	62	l_o_test:	BEQ	\$10 \$0
3	l_r_bni:	DATA	0		l_o_test_false		
4	l_Program_vt:	DATA	6592	63		XORI	\$10 1
5	l_RTM_vt:	DATA	348	64		ADDI	\$8 1
6		DATA	425	65		EXCH	\$19 \$17
7		DATA	2181	66		XOR	\$18 \$19
8		DATA	3606	67		EXCH	\$19 \$17
9		DATA	3677	68		RL	\$9 1
10		DATA	3976	69		EXCH	\$10 \$1
- 1							
11		DATA	5727	70		ADDI	\$1 -1
12	l_Cell_vt:	DATA	226	71		EXCH	\$11 \$1
13		DATA	257	72		ADDI	\$1 -1
14		DATA	288	73		EXCH	\$12 \$1
15		DATA	319	74		ADDI	\$1 -1
	l mallog ton.					EXCH	
16	l_malloc_top:	BRA	l_malloc_bot	75			\$14 \$1
17	l_malloc:	SWAPBR		76		ADDI	\$1 -1
18		NEG	\$2	77		EXCH	\$16 \$1
19		ADDI	\$9 2	78		ADDI	\$1 -1
20		XOR	\$8 \$0	79		EXCH	\$17 \$1
21		ADDI	\$1 1	80		ADDI	\$1 -1
		EXCH	\$6 \$1			EXCH	\$18 \$1
22				81			
23		ADDI	\$1 1	82		ADDI	\$1 -1
24		EXCH	\$7 \$1	83		EXCH	\$20 \$1
25		EXCH	\$2 \$1	84		ADDI	\$1 -1
26		ADDI	\$1 -1	85		EXCH	\$21 \$1
27		BRA	l_malloc1	86		ADDI	\$1 -1
		ADDI	\$1 1			EXCH	\$22 \$1
28				87			
29		EXCH	\$2 \$1	88		ADDI	\$1 -1
30		EXCH	\$7 \$1	89		EXCH	\$23 \$1
31		ADDI	\$1 -1	90		ADDI	\$1 -1
32		EXCH	\$6 \$1	91		BRA	l_malloc1
33		ADDI	\$1 -1	92		ADDI	\$1 1
34		XOR	\$8 \$0	93		EXCH	\$23 \$1
-							
35		ADDI	\$9 -2	94		ADDI	\$1 1
36	l_malloc_bot:	BRA	l_malloc_top	95		EXCH	\$22 \$1
37	l_malloc1_top:	BRA	l_malloc1_bot	96		ADDI	\$1 1
38		ADDI	\$1 1	97		EXCH	\$21 \$1
39		EXCH	\$2 \$1	98		ADDI	\$1 1
40		SUB	\$17 \$8	99		EXCH	\$20 \$1
		XOR					
41	1 111 -		\$17 \$4	100		ADDI	\$1 1
42	l_malloc1:	SWAPBR		101		EXCH	\$18 \$1
43		NEG	\$2	102		ADDI	\$1 1
44		EXCH	\$2 \$1	103		EXCH	\$17 \$1
45		ADDI	\$1 -1	104		ADDI	\$1 1
46		XOR	\$17 \$4	105		EXCH	\$16 \$1
47		ADD	\$17 \$8	106		ADDI	\$1 1
48		EXCH	\$19 \$17	107		EXCH	\$14 \$1
49		XOR	\$18 \$19	108		ADDI	\$1 1
50		EXCH	\$19 \$17	109		EXCH	\$12 \$1
51		XOR	\$13 \$9	110		ADDI	\$1 1
52		SUB	\$13 \$7	111		EXCH	\$11 \$1
53	cmp_top_12:	BGEZ	\$13 cmp_bot_1			ADDI	\$1 1
- 1		XORI	\$14 1	113		EXCH	\$10 \$1
54	amp bot 13.						
55	cmp_bot_13:	BGEZ	\$13 cmp_top_1			RR	\$9 1
56		XOR	\$10 \$14	115		ADDI	\$8 -1
57	cmp_bot_13_i:	BGEZ	\$13	116		XORI	\$10 1
	cmp_top_12_i			117	l_o_assert_true:	BRA	l_o_assert
58		XORI	\$14 1		l_o_test_false:	BRA	l_o_test
59	cmp_top_12_i:	BGEZ	\$13	119		BEQ	\$18 \$0
53			7 4 9	113	cmp_bot_17	2	+ + 0 + 0
0.0	cmp_bot_13_i	A D D	¢12 ¢7	100	cmp_boc_1/	VODT	¢20 1
60		ADD	\$13 \$7	120		XORI	\$20 1

ı							
121	cmp_bot_17:	BEQ	\$18 \$0	183		RR	\$9 1
	cmp_top_16			184		ADDI	\$8 -1
122		XOR	\$11 \$20	185		XOR	\$12 \$6
123	cmp_bot_17_i:	BEQ	\$18 \$0	186		EXCH	\$12 \$17
	cmp_top_16_i			187		ADD	\$6 \$9
124		XORI	\$20 1	188	l_i_assert:	BNE	\$11 \$0
125	cmp_top_16_i:	BEQ	\$18 \$0		l_i_assert_true		
	cmp_bot_17_i			189		EXCH	\$12 \$17
126	l_i_test:	BEQ	\$11 \$0	190		SUB	\$6 \$9
	l_i_test_false			191	cmp_top_18:	BEQ	\$6 \$12
127		XORI	\$11 1		cmp_bot_19		
128		ADD	\$6 \$18	192		XORI	\$21 1
129		SUB	\$18 \$6	193	cmp_bot_19:	BEQ	\$6 \$12
130		EXCH	\$12 \$6		cmp_top_18		
131		EXCH	\$12 \$17	194	cmp_top_20:	BNE	\$12 \$0
132		XOR	\$12 \$6		cmp_bot_21		
133		XORI	\$11 1	195		XORI	\$22 1
134	l_i_assert_true:	BRA	l_i_assert	196	cmp_bot_21:	BNE	\$12 \$0
135	l_i_test_false:	BRA	l_i_test		cmp_top_20		
136		ADDI	\$8 1	197		ORX	\$23 \$21 \$22
137		RL	\$9 1	198		XOR	\$11 \$23
138		EXCH	\$10 \$1	199		ORX	\$23 \$21 \$22
139		ADDI	\$1 -1	200	cmp_bot_21_i:	BNE	\$12 \$0
140		EXCH	\$11 \$1		cmp_top_20_i		
141		ADDI	\$1 -1	201		XORI	\$22 1
142		EXCH	\$12 \$1	202	cmp_top_20_i:	BNE	\$12 \$0
143		ADDI	\$1 -1		cmp_bot_21_i		
144		EXCH	\$14 \$1	203	cmp_bot_19_i:	BEQ	\$6 \$12
145		ADDI	\$1 -1		cmp_top_18_i	~	
146		EXCH	\$16 \$1	204		XORI	\$21 1
147		ADDI	\$1 -1	205	cmp_top_18_i:	BEQ	\$6 \$12
148		EXCH	\$17 \$1		cmp_bot_19_i		1 - 1
149		ADDI	\$1 -1	206	omb_200_13_1	ADD	\$6 \$9
150		EXCH	\$18 \$1	207		EXCH	\$12 \$17
151		ADDI	\$1 -1	208	l_o_assert:	BNE	\$10 \$0
152		EXCH	\$20 \$1	200	l_o_assert_true	DIVE	V10 V0
153		ADDI	\$1 -1	209	1_0_033010_0140	XOR	\$15 \$9
154		EXCH	\$21 \$1	210		SUB	\$15 \$7
155		ADDI	\$1 -1	211	cmp_top_14:	BGEZ	\$15 cmp_bot_15
156		EXCH	\$22 \$1	212	cmp_cop_14.	XORI	\$16 1
					amp bot 15.		
157		ADDI	\$1 -1	213	cmp_bot_15:	BGEZ	\$15 cmp_top_14
158		EXCH ADDI	\$23 \$1	214		XOR	\$10 \$16
159			\$1 -1	215	cmp_bot_15_i:	BGEZ	\$15
160		BRA	l_malloc1		cmp_top_14_i	WORT	616 1
161		ADDI	\$1 1	216	14	XORI	\$16 1
162		EXCH	\$23 \$1	217	cmp_top_14_i:	BGEZ	\$15
163		ADDI	\$1 1	010	cmp_bot_15_i	NDD.	¢15 ¢7
164		EXCH	\$22 \$1	218		ADD	\$15 \$7
165		ADDI	\$1 1	219	1 111 1	XOR	\$15 \$9
166		EXCH	\$21 \$1		l_malloc1_bot:	BRA	l_malloc1_top
167		ADDI	\$1 1	221	<pre>l_getLeft_8_top:</pre>	BRA	
168		EXCH	\$20 \$1		l_getLeft_8_bot		61 1
169		ADDI	\$1 1	222		ADDI	\$1 1
170		EXCH	\$18 \$1	223		EXCH	\$2 \$1
171		ADDI	\$1 1	224		EXCH	\$6 \$1
172		EXCH	\$17 \$1	225		ADDI	\$1 -1
173		ADDI	\$1 1	226		EXCH	\$3 \$1
174		EXCH	\$16 \$1	227		ADDI	\$1 -1
175		ADDI	\$1 1	228	l_getLeft_8:	SWAPBR	
176		EXCH	\$14 \$1	229		NEG	\$2
177		ADDI	\$1 1	230		ADDI	\$1 1
178		EXCH	\$12 \$1	231		EXCH	\$3 \$1
179		ADDI	\$1 1	232		ADDI	\$1 1
180		EXCH	\$11 \$1	233		EXCH	\$6 \$1
181		ADDI	\$1 1	234		EXCH	\$2 \$1
182		EXCH	\$10 \$1	235		ADDI	\$1 -1
				,			

236		ADD	ĊТ	\$3	298		ADD	ĊТ	\$3
237		ADDI	\$7	3	299		ADDI	\$7	2
238		EXCH	\$8	\$7	300		EXCH	\$8	\$7
239		ADDI	Ş /	-3	301		ADDI	Ş /	-2
240		SUB	\$7	\$3	302		SUB	\$7	\$3
241		EXCH	\$9	\$6	303		EXCH	\$9	\$6
	0.0					0.4			
242	swap_22:	XOR	\$8	\$9	304	swap_24:	XOR	\$8	\$9
243		XOR	\$9	\$8	305		XOR	\$9	\$8
244		XOR	ĠΩ	\$9	306		XOR	\$8	\$9
245		EXCH	\$9	\$6	307		EXCH	\$9	\$6
246		ADD	\$7	\$3	308		ADD	\$7	\$3
247		ADDI	\$7		309		ADDI	\$7	
248		EXCH	\$8	\$7	310		EXCH	\$8	\$7
249		ADDI	\$7	-3	311		ADDI	\$7	-2
250		SUB	¢7	\$3	312		SUB		\$3
			۱ ب	ŲΣ				۱ ب	ŲΟ
251	l_getLeft_8_bot:	BRA			313	l_getSelf_10_bot:	BRA		
	l_getLeft_8_top					l_getSelf_10_top			
050	_	DDA			21.4		DDA		
252	l_getRight_9_top:	BRA			314	l_getSymbol_11_top:	BRA		
	l_getRight_9_bot					l_getSymbol_11_bot			
253		ADDI	\$1	1	315		ADDI	\$1	1
		EXCH		\$1			EXCH		
254					316				\$1
255		EXCH	\$6	\$1	317		EXCH	\$6	\$1
256		ADDI	\$1	-1	318		ADDI	\$1	-1
257		EXCH		\$1	319		EXCH		\$1
258		ADDI	\$1	-1	320		ADDI	\$1	-1
259	l_getRight_9:	SWAPBR	\$2.		321	l_getSymbol_11:	SWAPBR	\$2.	
		NEG	\$2			=32 = .	NEG	\$2	
260					322				
261		ADDI	\$1	1	323		ADDI	\$1	1
262		EXCH	\$3	\$1	324		EXCH	\$3	\$1
263		ADDI	\$1				ADDI	\$1	1
					325				
264		EXCH	\$6	\$1	326		EXCH	\$6	\$1
265		EXCH	\$2	\$1	327		EXCH	\$2	\$1
		ADDI		-1	328		ADDI		-1
266									
267		ADD	\$7	\$3	329		EXCH	\$7	\$6
268		ADDI	\$7	4	330		ADD	\$8	\$3
269		EXCH		\$7	331		ADDI		5
270		ADDI	\$7	-4	332		EXCH	\$9	\$8
271		SUB	\$7	\$3	333		ADDI	\$8	-5
272		EXCH	\$9	\$6	334		SUB	\$8	\$3
	0.2					0.5			
273	swap_23:	XOR		\$9	335	swap_25:	XOR	\$ /	\$9
274		XOR	\$9	\$8	336		XOR	\$9	\$7
275		XOR	\$8	\$9	337		XOR	\$7	\$9
276		EXCH		\$6	338		ADD	\$8	\$3
277		ADD	\$7	\$3	339		ADDI	\$8	5
278		ADDI	\$7	4	340		EXCH	\$9	\$8
279		EXCH		\$7	341		ADDI		-5
280		ADDI		-4	342		SUB		\$3
281		SUB	\$7	\$3	343		EXCH	\$7	\$6
282	l_getRight_9_bot:	BRA			344	l_getSymbol_11_bot:	BRA		
202	_, ,				011	_,	2141		
	l_getRight_9_top					l_getSymbol_11_top			
283	l_getSelf_10_top:	BRA			345	<pre>l_initLiterals_1_top:</pre>	BRA		
	l getSelf 10 bot					l_initLiterals_1_bot			
	1_9000011_10_000		A 1	1	_	1_1111011001415_1_000		A 1	1
284		ADDI	\$1		346		ADDI	\$1	
285		EXCH	\$2	\$1	347		EXCH	\$2	\$1
286		EXCH		\$1	348		EXCH		\$1
287		ADDI		-1	349		ADDI		-1
288		EXCH	\$3	\$1	350	l_initLiterals_1:	SWAPBR	\$2	
289		ADDI		-1	351	· ·	NEG	\$2	
	1 co+Colf 10-			_					1
290	l_getSelf_10:	SWAPBR			352		ADDI	\$1	
291		NEG	\$2		353		EXCH	\$3	\$1
292		ADDI	\$1	1	354		EXCH	\$2	\$1
293		EXCH		\$1	355		ADDI		-1
294		ADDI	\$1	1	356		ADD	\$6	\$3
295		EXCH	\$6	\$1	357		ADDI	\$6	7
296		EXCH		\$1			EXCH		\$6
					358				
297		ADDI	\$1	-1	359		ADDI	\$6	-7

360		SUB	\$6	\$3	424		EXCH	\$2 \$1
361		XORI		9999	425		EXCH	\$3 \$1
362		ADD	\$7		426		ADDI	\$1 -1
363		XORI	\$8	9999	427	l_initRules_2:	SWAPBR	\$2
364		ADD	\$6	\$3	428		NEG	\$2
365		ADDI		7	429		ADDI	\$1 1
366		EXCH	\$7	\$6	430		EXCH	\$3 \$1
367		ADDI	\$6	-7	431		EXCH	\$2 \$1
368		SUB	\$6	\$3	432		ADDI	\$1 -1
369		ADD	\$6		433		XORI	\$7 8
370		ADDI	\$6	8	434	arr_con_26:	ADDI	\$9 2
371		EXCH	\$7	\$6	435		ADD	\$9 \$7
372		ADDI	\$6	-8	436		XORI	\$7 8
373		SUB	\$6		437		EXCH	\$3 \$1
374		XORI	\$8	9998	438		ADDI	\$1 -1
375		ADD	\$7	\$8	439		EXCH	\$9 \$1
376		XORI	\$8	9998	440		ADDI	\$1 -1
377		ADD		\$3	441		EXCH	\$8 \$1
378		ADDI	\$6	8	442		ADDI	\$1 -1
379		EXCH	\$7	\$6	443		BRA	l_malloc
380		ADDI	\$6	-8	444		ADDI	- \$1 1
381		SUB	\$6		445		EXCH	\$8 \$1
382		ADD	\$6	\$3	446		ADDI	\$1 1
383		ADDI	\$6	9	447		EXCH	\$9 \$1
384		EXCH		\$6	448		ADDI	\$1 1
385		ADDI	\$6	-9	449		EXCH	\$3 \$1
386		SUB	\$6	\$3	450		XORI	\$7 8
387		XORI	\$8	9997	451		SUB	\$9 \$7
						ann ann 26 i .		\$9 -2
388		ADD	\$7	\$8	452	arr_con_26_i:	ADDI	
389		XORI	\$8	9997	453		XORI	\$7 8
390		ADD	\$6	\$3	454		ADD	\$6 \$3
391		ADDI		9	455		ADDI	\$6 3
392		EXCH		\$6	456		XORI	\$7 8
393		ADDI	\$6	-9	457		XOR	\$9 \$7
394		SUB	\$6	\$3	458		EXCH	\$9 \$8
395		ADD	\$6		459		ADDI	\$8 1
396		ADDI	\$6		460		XORI	\$9 1
397		EXCH	\$7	\$6	461		EXCH	\$9 \$8
398		ADDI	\$6	-10	462		ADDI	\$8 -1
399		SUB		\$3	463	arr_con_26_bot:	EXCH	\$8 \$6
						411_0011_20_b00:		
400		XORI		9996	464		XORI	\$7 8
401		ADD	\$7	\$8	465		ADDI	\$6 -3
402		XORI	\$8	9996	466		SUB	\$6 \$3
403		ADD	\$6		467		XORI	\$7 8
						27.		
404		ADDI	\$6		468	arr_con_27:	ADDI	\$9 2
405		EXCH	\$7	\$6	469		ADD	\$9 \$7
406		ADDI	\$6	-10	470		XORI	\$7 8
407		SUB		\$3	471		EXCH	\$3 \$1
408		ADD		\$3	472		ADDI	\$1 -1
409		ADDI	\$6	15	473		EXCH	\$9 \$1
410		EXCH	\$7	\$6	474		ADDI	\$1 -1
411		ADDI		-15	475		EXCH	\$8 \$1
412		SUB		\$3	476		ADDI	\$1 -1
413		XORI	\$8	7	477		BRA	l_malloc
414		ADD	\$7	\$8	478		ADDI	\$1 1
415		XORI	\$8		479		EXCH	\$8 \$1
416		ADD		\$3	480		ADDI	\$1 1
417		ADDI	\$6	15	481		EXCH	\$9 \$1
418		EXCH	\$7	\$6	482		ADDI	\$1 1
419		ADDI		-15	483		EXCH	\$3 \$1
420		SUB	ÞЬ	\$3	484		XORI	\$7 8
421	l_initLiterals_1_bot:	BRA			485		SUB	\$9 \$7
	l_initLiterals_1_top				486	arr_con_27_i:	ADDI	\$9 -2
422	l_initRules_2_top:	BRA			487	<b>_</b>	XORI	\$7 8
144								
	l_initRules_2_bot				488		ADD	\$6 \$3
423		ADDI	\$1	1	489		ADDI	\$6 4
						-		

490		XORI	\$7 8	556	I	ADD	\$6	\$3
491		XOR	\$9 \$7	557		ADDI		6
492		EXCH	\$9 \$8	558		XORI	\$7	8
493		ADDI	\$8 1	559		XOR		\$7
494		XORI	\$9 1	560		EXCH	\$9	\$8
495		EXCH	\$9 \$8	561		ADDI	\$8	
496		ADDI	\$8 -1	562		XORI	\$9	
497	arr_con_27_bot:	EXCH	\$8 \$6	563		EXCH	\$9	
498		XORI	\$7 8	564		ADDI	\$8	
499		ADDI	\$6 -4	565	arr_con_29_bot:	EXCH		\$6
500		SUB	\$6 \$3	566		XORI	\$7	8
501		XORI	\$7 8	567		ADDI	\$6	-6
502	arr_con_28:	ADDI	\$9 2	568		SUB	\$6	\$3
503		ADD	\$9 \$7	569		ADD	\$6	\$3
504		XORI	\$7 8	570		ADDI	\$6	3
505		EXCH	\$3 \$1	571		EXCH	\$8	\$6
506		ADDI	\$1 -1	572		XOR	\$7	\$8
507		EXCH	\$9 \$1	573		EXCH	\$8	\$6
508		ADDI	\$1 -1	574		ADDI	\$7	2
509		EXCH	\$8 \$1	575		ADD	\$7	\$0
510		ADDI	\$1 -1	576		ADDI	\$6	
511		BRA	l_malloc	577		SUB	\$6	
512		ADDI	\$1 1	578		EXCH	\$9	
513		EXCH	\$8 \$1	579		ADD		\$3
514		ADDI	\$1 1	580		ADDI	\$6	
515		EXCH	\$9 \$1	581		SUB	\$7	
516		ADDI	\$1 1	582		ADDI		-2
517		EXCH	\$3 \$1	583		EXCH		\$6
518		XORI	\$7 8	584		XOR	\$7	\$8
519	20 30 i	SUB	\$9 \$7	585		EXCH		\$6
520 $521$	arr_con_28_i:	ADDI XORI	\$9 -2 \$7 8	586 587		ADDI SUB	\$6 \$6	
522		ADD	\$6 \$3			XORI	\$10	
022								
				588 580	assArrFlom 30.			
523		ADDI	\$6 5	589	assArrElem_30:	ADD	\$9	\$10
523 $524$		ADDI XORI	\$6 5 \$7 8	589 590	assArrElem_30:	ADD XORI	\$9 \$10	\$10 ) 1
523 524 525		ADDI XORI XOR	\$6 5 \$7 8 \$9 \$7	589 590 591	assArrElem_30:	ADD XORI ADD	\$9 \$10 \$6	\$10 ) 1 \$3
523 524 525 526		ADDI XORI XOR EXCH	\$6 5 \$7 8 \$9 \$7 \$9 \$8	589 590 591 592	assArrElem_30:	ADD XORI ADD ADDI	\$9 \$10 \$6 \$6	\$10 1 \$3 3
523 524 525 526 527		ADDI XORI XOR	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1	589 590 591	assArrElem_30:	ADD XORI ADD	\$9 \$10 \$6 \$6 \$8	\$10 ) 1 \$3 3 \$6
523 524 525 526		ADDI XORI XOR EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8	589 590 591 592 593	assArrElem_30:	ADD XORI ADD ADDI EXCH	\$9 \$10 \$6 \$6	\$10 \$3 3 \$6 \$8
523 524 525 526 527 528		ADDI XORI XOR EXCH ADDI XORI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1	589 590 591 592 593 594	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR	\$9 \$10 \$6 \$6 \$8 \$7	\$10 \$3 3 \$6 \$8
523 524 525 526 527 528 529	arr_con_28_bot:	ADDI XORI XOR EXCH ADDI XORI EXCH	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8	589 590 591 592 593 594 595	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH	\$9 \$10 \$6 \$6 \$8 \$7 \$8	\$10 ) 1 \$3 3 \$6 \$8 \$6 2
523 524 525 526 527 528 529 530	arr_con_28_bot:	ADDI XORI XOR EXCH ADDI XORI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$8 -1	589 590 591 592 593 594 595 596	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI	\$9 \$10 \$6 \$6 \$8 \$7 \$8 \$7	\$10 \$3 3 \$6 \$8 \$6 2 \$0
523 524 525 526 527 528 529 530 531	arr_con_28_bot:	ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$8 -1 \$8 \$6	589 590 591 592 593 594 595 596 597	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADD	\$9 \$10 \$6 \$6 \$8 \$7 \$8 \$7	\$10 \$3 3 \$6 \$8 \$6 2 \$0 -3
523 524 525 526 527 528 529 530 531 532	arr_con_28_bot:	ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8	589 590 591 592 593 594 595 596 597 598	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI ADD ADDI	\$9 \$10 \$6 \$8 \$7 \$8 \$7 \$8 \$7 \$6	\$10 \$3 3 \$6 \$8 \$6 2 \$0 -3
523 524 525 526 527 528 529 530 531 532 533 534 535		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB XORI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$8 -1 \$8 \$-1 \$7 8 \$6 \$-5 \$6 \$3 \$7 8	589 590 591 592 593 594 595 596 597 598 599 600 601	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADD ADDI SUB EXCH ADD	\$9 \$6 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$8	\$10 ) 1 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$7 \$3
523 524 525 526 527 528 529 530 531 532 533 534 535 536	<pre>arr_con_28_bot: arr_con_29:</pre>	ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2	589 590 591 592 593 594 595 596 597 598 599 600 601 602	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADD ADDI SUB EXCH ADD ADDI ADD ADDI	\$9 \$10 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6	\$10 ) 1 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$7 \$3 3
523 524 525 526 527 528 529 530 531 532 533 534 535 536		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADDI ADD	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$1 \$9 \$8 \$1 \$9 \$8 \$6 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7	589 590 591 592 593 594 595 596 597 598 599 600 601 602 603	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADD ADDI SUB EXCH ADD ADDI SUB EXCH ADD ADDI SUB	\$9 \$10 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$8 \$7 \$6 \$6 \$7 \$6 \$6 \$7 \$6 \$7 \$6 \$6 \$7 \$6 \$7 \$6 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7	\$10 \$3 3 \$6 \$8 \$6 \$2 \$0 -3 \$3 \$7 \$3 \$9
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$1 \$9 \$8 \$1 \$9 \$8 \$6 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8	589 590 591 592 593 594 595 596 597 598 600 601 602 603 604	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADD ADDI SUB EXCH ADD ADDI SUB EXCH ADD	\$9 \$10 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$6 \$6 \$6 \$7 \$6 \$6 \$7 \$7 \$6 \$6 \$7 \$7 \$6 \$6 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7	\$10 ) 1 \$3 3 \$6 \$8 \$6 \$2 \$0 -3 \$3 \$7 \$3 \$0 -2
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI EXCH	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$3 \$1	589 590 591 592 593 594 595 596 597 600 601 602 603 604 605	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADD ADDI SUB EXCH ADD ADDI SUB EXCH ADD ADDI SUB EXCH ADD ADDI SUB EXCH	\$9 \$10 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$8 \$7 \$6 \$6 \$6 \$7 \$7 \$6 \$6 \$7 \$7 \$6 \$7 \$7 \$6 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7	\$10 ) 1 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$3 \$7 \$3 3 \$6 -3 \$5 56 57 \$5 57 \$5 57 57 57 57 57 57 57 57 57 57 57 57 57
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$1 1 \$9 \$8 \$1 1 \$8 \$6 \$7 8 \$6 \$5 \$7 8 \$9 2 \$9 \$7 \$7 8 \$3 \$1 \$1 -1	589 590 591 592 593 594 595 596 600 601 602 603 604 605 606	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI	\$9 \$6 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$8 \$7 \$6 \$6 \$7 \$6 \$6 \$7 \$7 \$6 \$6 \$6 \$7 \$7 \$7 \$6 \$6 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7	\$10 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$3 \$7 \$3 \$6 \$8 \$8 \$6 \$8 \$8 \$6 \$8 \$8 \$6 \$8 \$8 \$6 \$8 \$8 \$8 \$8 \$8 \$8 \$8 \$8 \$8 \$8
523 524 525 526 527 528 530 531 532 533 534 535 536 537 538 539 540 541		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI EXCH ADDI EXCH ADDI EXCH	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$1 4 \$9 \$8 \$1 5 \$1 8 \$6 -5 \$6 -5 \$6 -5 \$6 8 \$7 8 \$9 2 \$9 \$7 \$7 8 \$3 \$1 \$1 -1 \$9 \$1	589 590 591 592 593 594 596 597 598 600 601 602 603 604 605 606 607	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADD ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI EXCH XOR EXCH	\$9 \$6 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$8 \$7 \$6 \$6 \$7 \$7 \$6 \$6 \$7 \$7 \$6 \$6 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7	\$10 1 \$3 3 \$6 \$8 \$6 \$2 \$0 -3 \$7 \$3 \$0 -2 \$6 \$8 \$8
523 524 525 526 527 528 530 531 532 533 534 535 536 537 538 539 540 541		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI EXCH ADDI EXCH ADDI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$1 4 \$9 \$8 \$1 5 \$1 8 \$6 -5 \$6 6 -5 \$6 6 -5 \$6 83 \$7 8 \$9 2 \$9 \$7 \$7 8 \$9 2 \$9 \$7 \$7 8 \$1 1 \$1 1 \$2 1 \$3 1 \$4 1 \$5 1 \$6 1 \$7 1 \$7 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1 \$8 1	589 590 591 592 593 594 595 596 597 600 601 602 603 604 605 606 607 608	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADD ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI EXCH ADDI ADDI ADDI ADDI ADDI ADDI ADDI ADD	\$9 \$10 \$6 \$8 \$7 \$8 \$7 \$6 \$9 \$6 \$7 \$8 \$7 \$8 \$7 \$8 \$6 \$6 \$6 \$6 \$7 \$8 \$6 \$6 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7	\$10 1 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$7 \$3 \$0 -2 \$6 \$8 \$6 -3
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI EXCH ADDI EXCH ADDI EXCH ADDI EXCH	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$1 4 \$9 \$8 \$6 -5 \$6 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$3 \$1 \$1 -1 \$9 \$1 \$1 -1 \$8 \$1	589 590 591 592 593 594 595 596 600 601 602 603 604 605 606 607 608 609	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADD ADDI SUB EXCH ADD ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB	\$9 \$10 \$6 \$8 \$7 \$6 \$6 \$6 \$7 \$6 \$6 \$7 \$8 \$7 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6	\$10 1 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$3 \$7 \$3 \$6 \$8 \$6 -2 \$6 \$8 \$6 -2 \$6 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 \$6 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2 -2
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH ADDI SUB XORI ADDI ADDI XORI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$6 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 \$7 \$7 8 \$9 \$7 \$7 8 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1	589 590 591 592 593 594 595 596 600 601 602 603 604 605 606 607 608 609 610	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADD ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI	\$9.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00 \$6.00	\$100 1 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$3 \$7 \$3 3 \$6 \$8 \$6 -3 \$3 \$3
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH ADDI SUB XORI ADDI ADDI ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$3 \$1 \$1 -1 \$9 \$1 \$1 -1 \$8 \$1 \$1 -1 \$1 -1 \$2 \$1 \$2 \$1 \$3 \$1 \$4 \$1 \$5 \$1 \$6 \$1 \$6 \$1 \$7 \$1 \$7 \$1 \$8 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1	589 590 591 592 593 594 595 596 600 601 602 603 604 605 606 607 608 609 610 611	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADD ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI EXCH XOR EXCH ADDI ADDI SUB ADDI ADDI	\$9 \$6 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$9 \$6 \$7 \$8 \$7 \$8 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6	\$10 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$3 \$7 \$3 \$3 \$6 \$8 \$6 \$2 \$7 \$3 \$7 \$3 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545		ADDI XORI XOR EXCH ADDI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$9 2 \$9 \$7 \$7 8 \$1 -1 \$9 \$1 \$1 -1 \$1 -1 \$1 -1 \$1 -1 \$2 \$1 \$1 -1 \$2 \$1 \$1 -1 \$3 \$1 \$1 -1 \$3 \$1 \$1 -1 \$4 \$1 \$5 \$1 \$1 -1 \$5 \$1 \$1 -1 \$6 \$1 \$1 -1 \$1 -1	589 590 591 592 593 594 595 596 601 602 603 604 605 606 607 608 609 610 611 611	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI EXCH XOR EXCH ADDI SUB ADDI EXCH XOR EXCH ADDI SUB ADDI EXCH SUB ADDI EXCH ADDI EXCH	\$9 \$6 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$9 \$6 \$6 \$7 \$8 \$7 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6	\$10 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$3 \$7 \$3 \$3 \$6 \$8 \$6 2 \$7 \$7 \$7 \$3 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7 \$7
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545		ADDI XORI XOR EXCH ADDI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 \$1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$9 2 \$9 \$7 \$7 8 \$1 -1 \$9 \$1 \$1 -1 \$1 -1 \$8 \$1 \$1 -1 \$1 -	589 590 591 592 593 594 595 596 600 601 602 603 604 605 606 607 608 609 610 611 612 613	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADD ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI EXCH ADD ADDI EXCH ADD ADDI EXCH ADDI EXCH ADDI SUB ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$9 \$6 \$6 \$8 \$7 \$8 \$7 \$6 \$9 \$6 \$6 \$7 \$8 \$7 \$6 \$6 \$6 \$7 \$7 \$8 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6	\$10 \$3 3 \$6 \$8 \$6 2 0 -3 \$7 \$3 \$7 \$3 \$6 \$8 6 -3 \$7 \$5 5 5 6 8 8 6 8 6 7 8 7 8 7 8 7 8 7 8 8 7 8 7
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545		ADDI XORI XOR EXCH ADDI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$9 2 \$9 \$7 \$7 8 \$1 -1 \$9 \$1 \$1 -1 \$1 -1 \$1 -1 \$1 -1 \$2 \$1 \$1 -1 \$2 \$1 \$1 -1 \$3 \$1 \$1 -1 \$3 \$1 \$1 -1 \$4 \$1 \$5 \$1 \$1 -1 \$5 \$1 \$1 -1 \$6 \$1 \$1 -1 \$1 -1	589 590 591 592 593 594 595 596 601 602 603 604 605 606 607 608 609 610 611 611	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI EXCH XOR EXCH ADDI SUB ADDI EXCH XOR EXCH ADDI SUB ADDI EXCH SUB ADDI EXCH ADDI EXCH	\$9 \$6 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$9 \$6 \$6 \$7 \$8 \$7 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6	\$10 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$7 \$3 \$7 \$3 \$6 \$8 6 -2 \$6 8 8 6 8 8 6 8 8 6 8 8 8 8 8 8 8 8 8
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 540 541 542 543 544 543 544 545		ADDI XORI XOR EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI SUB XORI ADDI ADD XORI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$9 2 \$9 \$7 \$7 8 \$1 -1 \$9 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$1 -1	589 590 591 592 593 594 595 598 600 601 602 603 604 606 607 608 609 610 611 612 613	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI EXCH ADD ADDI EXCH XOR EXCH ADDI SUB ADDI EXCH ADDI EXCH ADDI SUB ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH EXCH	\$9 \$6 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$6 \$6 \$7 \$7 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6	\$10 \$3 3 \$6 \$8 \$6 2 \$0 -3 \$3 \$7 \$3 \$5 -2 \$6 \$8 6 -3 \$5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547		ADDI XORI XOR EXCH ADDI EXCH ADDI EXCH ADDI SUB XORI ADDI ADD XORI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 \$1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$3 \$1 \$1 -1 \$9 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 1 \$8 \$1 \$1 1 \$9 \$1	589 590 591 592 593 594 595 598 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI EXCH ADD ADDI EXCH XOR EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH ADDI EXCH ADDI	\$9 \$6 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$6 \$6 \$7 \$7 \$8 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6	\$10 \$3 \$3 \$6 \$8 \$6 \$2 \$0 -3 \$3 \$7 \$3 \$3 \$6 \$8 6 -2 \$6 -3 \$5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
523 524 525 526 527 528 529 530 531 532 533 534 535 540 541 542 543 544 545 546 547 548 549 550		ADDI XORI XOR EXCH ADDI EXCH ADDI EXCH ADDI SUB XORI ADDI ADD XORI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 \$1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$3 \$1 \$1 -1 \$8 \$1 \$1 -1 \$1 -1 \$2 \$1 \$3 \$1 \$1 1 \$3 \$1 \$4 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1 \$5 \$1	589 590 591 592 593 594 595 598 600 601 602 603 604 606 607 608 609 611 612 613 614 615 616	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI EXCH XOR EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH ADDI ADDI SUB ADDI EXCH ADDI ADDI ADDI ADDI ADDI ADDI ADDI ADD	\$9 \$6 \$6 \$8 \$7 \$8 \$7 \$6 \$6 \$6 \$6 \$7 \$7 \$8 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6	\$10 \$3 \$3 \$6 \$8 \$6 2 \$-3 \$3 \$7 \$3 \$3 \$-2 \$6 \$6 -3 \$3 \$5 \$6 -3 \$5 \$5 \$6 -3 \$5 \$5 \$5 \$6 -3 \$5 \$5 \$5 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6
523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 540 541 542 543 544 545 546 547 548 549 549 549 549 549 549 540 541 542 543 544 545 546 547 548 549 549 540 540 540 540 540 540 540 540		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 \$1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$3 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$1	589 590 591 592 593 594 595 599 600 601 602 603 604 606 607 608 609 611 612 613 614 615 616 616	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI EXCH XOR EXCH ADDI SUB ADDI EXCH ADDI ADDI EXCH ADDI ADDI ADDI ADDI ADDI ADDI ADDI ADD	\$910 \$6688788776669667788786666688787766	\$10 \$3 \$3 \$6 \$8 \$6 \$2 \$0 -33 \$7 \$3 \$3 \$6 \$8 6 2 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
523 524 525 526 527 528 529 530 531 533 534 535 536 537 538 540 541 542 543 544 545 546 547 548 549 550 551 552		ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 \$1 \$9 \$8 \$1 -1 \$8 \$6 \$7 8 \$6 -5 \$6 -5 \$6 -5 \$7 8 \$9 2 \$9 \$7 \$7 8 \$3 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1	589 590 591 592 593 594 595 596 600 601 602 603 604 606 607 608 609 610 611 612 613 614 615 616 617 618	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADDI SUB ADDI SUB ADDI SUB ADDI EXCH XOR EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH SUB ADDI EXCH SUB ADDI EXCH SUB ADDI EXCH SUB ADDI EXCH SUB ADDI EXCH SUB ADDI EXCH SUB ADDI EXCH SUB	\$916668778669966778786666887877666	\$10 \$3 \$3 \$6 \$8 \$6 2 \$0 -3 3 \$7 \$3 3 \$0 -2 \$5 \$6 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
523 524 525 526 527 528 529 530 531 533 534 535 536 537 538 540 541 542 543 544 545 546 547 548 549 550 551 552 553	arr_con_29:	ADDI XORI XOR EXCH ADDI XORI EXCH ADDI EXCH XORI ADDI SUB XORI ADDI ADD XORI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$6 5 \$7 8 \$9 \$7 \$9 \$8 \$8 1 \$9 \$1 \$9 \$8 \$8 -1 \$8 \$6 \$7 8 \$6 -5 \$6 \$3 \$7 8 \$9 2 \$9 \$7 \$7 8 \$3 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 -1 \$8 \$1 \$1 1 \$8 1 \$8 \$1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 3 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$	589 590 591 592 593 594 595 596 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619	assArrElem_30:	ADD XORI ADD ADDI EXCH XOR EXCH ADDI ADDI SUB EXCH ADDI SUB ADDI SUB ADDI EXCH XOR EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH ADDI SUB ADDI EXCH SUB ADDI EXCH SUB ADDI EXCH SUB ADDI EXCH SUB EXCH ADDI SUB EXCH	\$9 \$6 \$6 \$8 \$7 \$6 \$6 \$6 \$6 \$7 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6 \$6	\$10 \$3 \$3 \$6 \$8 \$6 \$2 \$0 \$3 \$7 \$3 \$3 \$6 \$8 \$6 \$2 \$0 \$0 \$2 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0 \$0

622		SUB	\$7 \$0	688		SUB	\$10 \$3
623		ADDI	\$7 -2	689		ADD	\$6 \$3
624		EXCH	\$8 \$6	690		ADDI	\$6 6
625		XOR	\$7 \$8	691		EXCH	\$8 \$6
626		EXCH	\$8 \$6	692		XOR	\$7 \$8
627		ADDI	\$6 -5	693		EXCH	\$8 \$6
628		SUB ADD	\$6 \$3	694		ADDI ADD	\$7 2 \$7 \$0
629		ADDI	\$10 \$3 \$10 10	695 696		ADDI	\$7 \$0 \$6 -6
630 631		EXCH	\$10 10	697		SUB	\$6 \$3
632		ADDI	\$10 -10	698		EXCH	\$9 \$7
633		SUB	\$10 \$3	699		ADD	\$6 \$3
634	assArrElem_31:	ADD	\$9 \$11	700		ADDI	\$6 6
635		ADD	\$10 \$3	701		SUB	\$7 \$0
636		ADDI	\$10 10	702		ADDI	\$7 -2
637		EXCH	\$11 \$10	703		EXCH	\$8 \$6
638		ADDI	\$10 -10	704		XOR	\$7 \$8
639		SUB	\$10 \$3	705		EXCH	\$8 \$6
640		ADD	\$6 \$3	706		ADDI	\$6 -6
641		ADDI	\$6.5	707		SUB	\$6 \$3
642 643		EXCH XOR	\$8 \$6 \$7 \$8	708 709		ADD ADDI	\$6 \$3 \$6 4
644		EXCH	\$8 \$6	710		EXCH	\$8 \$6
645		ADDI	\$7 2	711		XOR	\$7 \$8
646		ADD	\$7 \$0	712		EXCH	\$8 \$6
647		ADDI	\$6 -5	713		ADDI	\$7 2
648		SUB	\$6 \$3	714		ADD	\$7 \$0
649		EXCH	\$9 \$7	715		ADDI	\$6 -4
650		ADD	\$6 \$3	716		SUB	\$6 \$3
651		ADDI	\$6 5	717		EXCH	\$9 \$7
652		SUB	\$7 \$0	718		ADD	\$6 \$3
653		ADDI	\$7 -2	719		ADDI	\$6 4
654 655		EXCH XOR	\$8 \$6 \$7 \$8	720 721		SUB ADDI	\$7 \$0 \$7 -2
656		EXCH	\$8 \$6	721		EXCH	\$8 \$6
657		ADDI	\$6 -5	723		XOR	\$7 \$8
658		SUB	\$6 \$3	724		EXCH	\$8 \$6
659		ADD	\$6 \$3	725		ADDI	\$6 -4
660		ADDI	\$6 6	726		SUB	\$6 \$3
661		EXCH	\$8 \$6	727		XORI	\$10 2
662		XOR	\$7 \$8		ssArrElem_33:	ADD	\$9 \$10
663		EXCH	\$8 \$6	729		XORI	\$10 2
664		ADDI ADD	\$7 2 \$7 \$0	730		ADD ADDI	\$6 \$3 \$6 4
665 666		ADDI	\$6 -6	731 732		EXCH	\$6 4 \$8 \$6
667		SUB	\$6 \$3	733		XOR	\$7 \$8
668		EXCH	\$9 \$7	734		EXCH	\$8 \$6
669		ADD	\$6 \$3	735		ADDI	\$7 2
670		ADDI	\$6 6	736		ADD	\$7 \$0
671		SUB	\$7 \$0	737		ADDI	\$6 -4
672		ADDI	\$7 -2	738		SUB	\$6 \$3
673		EXCH	\$8 \$6	739		EXCH	\$9 \$7
674		XOR	\$7 \$8	740		ADD ADDI	\$6 \$3
675 676		EXCH ADDI	\$8 \$6 \$6 -6	741 742		SUB	\$6 4 \$7 \$0
677		SUB	\$6 \$3	743		ADDI	\$7 -2
678		ADD	\$10 \$3	744		EXCH	\$8 \$6
679		ADDI	\$10 10	745		XOR	\$7 \$8
680		EXCH	\$11 \$10	746		EXCH	\$8 \$6
681		ADDI	\$10 -10	747		ADDI	\$6 -4
682		SUB	\$10 \$3	748		SUB	\$6 \$3
683	assArrElem_32:	ADD	\$9 \$11	749		ADD	\$6 \$3
684		ADD	\$10 \$3	750		ADDI	\$6 3
685		ADDI	\$10 10	751		XORI	\$7 1
686 687		EXCH ADDI	\$11 \$10 \$10 -10	752 753		EXCH XOR	\$9 \$6 \$8 \$9
001		ADDI	ÅT0 -T0	199		AUK	<b>υ</b> υ υ υ

75.4		EXCH	ċΩ	06	990	I	CIID	¢6 ¢3
754		ADDI		\$6 2	820		SUB ADD	\$6 \$3
755					821			\$11 \$3
756		ADD		\$7	822		ADDI	\$11 7
757		XORI	\$7		823		EXCH	\$12 \$11
758		ADDI		-3	824		ADDI	\$11 -7
759		SUB		\$3	825		SUB	\$11 \$3
760		EXCH		0 \$8	826	assArrElem_35:	ADD	\$10 \$12
761		ADD	\$6	\$3	827		ADD	\$11 \$3
762		ADDI	\$6	3	828		ADDI	\$11 7
763		XORI	\$7	1	829		EXCH	\$12 \$11
764		SUB	\$8	\$7	830		ADDI	\$11 -7
765		ADDI	\$8	-2	831		SUB	\$11 \$3
766		EXCH	\$9	\$6	832		ADD	\$6 \$3
767		XOR	\$8	\$9	833		ADDI	\$6 5
768		EXCH	\$9	\$6	834		XORI	\$7 1
769		XORI	\$7	1	835		EXCH	\$9 \$6
770		ADDI	\$6	-3	836		XOR	\$8 \$9
771		SUB		\$3	837		EXCH	\$9 \$6
772		XORI		1 2	838		ADDI	\$8 2
773	assArrElem_34:	ADD		0 \$11			ADD	\$8 \$7
774		XORI		1 2	840		XORI	\$7 1
775		ADD		\$3	841		ADDI	\$6 -5
		ADDI	\$6		842		SUB	\$6 \$3
776								
777		XORI	\$7		843		EXCH	\$10 \$8
778		EXCH		\$6	844		ADD	\$6 \$3
779		XOR		\$9	845		ADDI	\$6 5
780		EXCH		\$6	846		XORI	\$7 1
781		ADDI	\$8		847		SUB	\$8 \$7
782		ADD	\$8	\$7	848		ADDI	\$8 -2
783		XORI	\$7		849		EXCH	\$9 \$6
784		ADDI		-3	850		XOR	\$8 \$9
785		SUB	\$6	\$3	851		EXCH	\$9 \$6
786		EXCH	\$10	0 \$8	852		XORI	\$7 1
787		ADD	\$6	\$3	853		ADDI	\$6 -5
788		ADDI	\$6	3	854		SUB	\$6 \$3
789		XORI	\$7	1	855		ADD	\$6 \$3
790		SUB	\$8	\$7	856		ADDI	\$6 6
791		ADDI	\$8	-2	857		XORI	\$7 1
792		EXCH	\$9	\$6	858		EXCH	\$9 \$6
793		XOR	\$8	\$9	859		XOR	\$8 \$9
794		EXCH	\$9	\$6	860		EXCH	\$9 \$6
795		XORI	\$7	1	861		ADDI	\$8 2
796		ADDI	\$6	-3	862		ADD	\$8 \$7
797		SUB		\$3	863		XORI	\$7 1
798		ADD		\$3	864		ADDI	\$6 -6
799		ADDI	\$6		865		SUB	\$6 \$3
800		XORI	\$7		866		EXCH	\$10 \$8
801		EXCH		\$6	867		ADD	\$6 \$3
802		XOR		\$9	868		ADDI	\$6 6
803		EXCH		\$6	869		XORI	\$7 1
804		ADDI	\$8		870		SUB	\$8 \$7
805		ADD		\$7	871		ADDI	\$8 -2
		XORI	\$7		872		EXCH	\$9 \$6
806 807		ADDI		_5	872 873		XOR	\$8 \$9
1		SUB		-3 \$3			EXCH	
808					874			\$9 \$6 \$7 1
809		EXCH		8\$ 0	875		XORI	\$7 1
810		ADD		\$3	876		ADDI	\$6 -6
811		ADDI	\$6		877		SUB	\$6 \$3
812		XORI	\$7		878		ADD	\$11 \$3
813		SUB		\$7	879		ADDI	\$11 9
814		ADDI		-2	880		EXCH	\$12 \$11
815		EXCH		\$6	881		ADDI	\$11 -9
816		XOR		\$9	882	_	SUB	\$11 \$3
817		EXCH		\$6	883	assArrElem_36:	ADD	\$10 \$12
818		XORI	\$7		884		ADD	\$11 \$3
819		ADDI	\$6	-5	885		ADDI	\$11 9

886		EXCH	\$12 \$11	952		XORI	\$7 1
887		ADDI	\$11 -9	953		SUB	\$8 \$7
888		SUB	\$11 \$3	954		ADDI	\$8 -2
889		ADD	\$6 \$3	955		EXCH	\$9 \$6
890		ADDI	\$6 6	956		XOR	\$8 \$9
891		XORI	\$7 1	957		EXCH	\$9 \$6
892		EXCH	\$9 \$6	958		XORI	\$7 1
893		XOR	\$8 \$9	959		ADDI	\$6 -4
894 895		EXCH ADDI	\$9 \$6 \$8 2	960 961		SUB ADD	\$6 \$3 \$6 \$3
896		ADDI	\$8 \$7	962		ADDI	\$6 3
897		XORI	\$7 1	963		XORI	\$7 2
898		ADDI	\$6 -6	964		EXCH	\$9 \$6
899		SUB	\$6 \$3	965		XOR	\$8 \$9
900		EXCH	\$10 \$8	966		EXCH	\$9 \$6
901		ADD	\$6 \$3	967		ADDI	\$8 2
902		ADDI	\$6 6	968		ADD	\$8 \$7
903		XORI	\$7 1	969		XORI	\$7 2
904		SUB	\$8 \$7	970		ADDI	\$6 -3
905		ADDI	\$8 -2	971		SUB	\$6 \$3
906		EXCH	\$9 \$6	972		EXCH	\$10 \$8
907		XOR	\$8 \$9	973		ADD	\$6 \$3
908		EXCH XORI	\$9 \$6 \$7 1	974		ADDI XORI	\$6 3 \$7 2
909 910		ADDI	\$6 -6	975 976		SUB	\$8 \$7
911		SUB	\$6 \$3	977		ADDI	\$8 -2
912		ADD	\$6 \$3	978		EXCH	\$9 \$6
913		ADDI	\$6 4	979		XOR	\$8 \$9
914		XORI	\$7 1	980		EXCH	\$9 \$6
915		EXCH	\$9 \$6	981		XORI	\$7 2
916		XOR	\$8 \$9	982		ADDI	\$6 -3
917		EXCH	\$9 \$6	983		SUB	\$6 \$3
918		ADDI	\$8 2	984		XORI	\$11 3
919		ADD	\$8 \$7	1	assArrElem_38:	ADD	\$10 \$11
920		XORI	\$7 1	986		XORI	\$11 3
921		ADDI	\$6 -4	987		ADD	\$6 \$3
922		SUB	\$6 \$3	988		ADDI	\$6 3
923 924		EXCH ADD	\$10 \$8 \$6 \$3	989 990		XORI EXCH	\$7 2 \$9 \$6
925		ADDI	\$6 4	991		XOR	\$8 \$9
926		XORI	\$7 1	992		EXCH	\$9 \$6
927		SUB	\$8 \$7	993		ADDI	\$8 2
928		ADDI	\$8 -2	994		ADD	\$8 \$7
929		EXCH	\$9 \$6	995		XORI	\$7 2
930		XOR	\$8 \$9	996		ADDI	\$6 -3
931		EXCH	\$9 \$6	997		SUB	\$6 \$3
932		XORI	\$7 1	998		EXCH	\$10 \$8
933							66 63
		ADDI	\$6 -4	999		ADD	\$6 \$3
934		SUB	\$6 -4 \$6 \$3	999 1000		ADDI	\$6 3
935	agglawElem 27.	SUB XORI	\$6 -4 \$6 \$3 \$11 3	999 1000 1001		ADDI XORI	\$6 3 \$7 2
935 936	assArrElem_37:	SUB XORI ADD	\$6 -4 \$6 \$3 \$11 3 \$10 \$11	999 1000 1001 1002		ADDI XORI SUB	\$6 3 \$7 2 \$8 \$7
935 936 937	assArrElem_37:	SUB XORI ADD XORI	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3	999 1000 1001 1002 1003		ADDI XORI SUB ADDI	\$6 3 \$7 2 \$8 \$7 \$8 -2
935 936	assArrElem_37:	SUB XORI ADD	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3	999 1000 1001 1002 1003 1004		ADDI XORI SUB	\$6 3 \$7 2 \$8 \$7 \$8 -2 \$9 \$6
935 936 937 938	assArrElem_37:	SUB XORI ADD XORI ADD	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3	999 1000 1001 1002 1003		ADDI XORI SUB ADDI EXCH	\$6 3 \$7 2 \$8 \$7 \$8 -2
935 936 937 938 939	assArrElem_37:	SUB XORI ADD XORI ADD ADDI	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3 \$6 4	999 1000 1001 1002 1003 1004 1005		ADDI XORI SUB ADDI EXCH XOR	\$6 3 \$7 2 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9
935 936 937 938 939 940	assArrElem_37:	SUB XORI ADD XORI ADD ADDI XORI	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3 \$6 4 \$7 1	999 1000 1001 1002 1003 1004 1005 1006		ADDI XORI SUB ADDI EXCH XOR EXCH	\$6 3 \$7 2 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6
935 936 937 938 939 940 941	assArrElem_37:	SUB XORI ADD XORI ADD ADDI XORI EXCH XOR	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3 \$6 4 \$7 1 \$9 \$6 \$8 \$9 \$9 \$6	999 1000 1001 1002 1003 1004 1005 1006 1007		ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB	\$6 3 \$7 2 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 2
935 936 937 938 939 940 941 942 943	assArrElem_37:	SUB XORI ADD XORI ADD ADDI XORI EXCH XOR EXCH ADDI	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3 \$6 4 \$7 1 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2	999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009		ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB ADD	\$6 3 \$7 2 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 2 \$6 -3 \$6 \$3 \$6 \$3
935 936 937 938 939 940 941 942 943 944	assArrElem_37:	SUB XORI ADD XORI ADD ADDI XORI EXCH XOR EXCH ADDI ADDI ADDI	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3 \$6 4 \$7 1 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7	999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010		ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB ADD ADDI	\$6 3 \$7 2 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 2 \$6 -3 \$6 \$3 \$6 \$3 \$6 \$5
935 936 937 938 939 940 941 942 943 944 945	assArrElem_37:	SUB XORI ADD XORI ADDI XORI EXCH XOR EXCH ADDI ADDI XORI EXCH XOR	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3 \$6 4 \$7 1 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 1	999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012		ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB ADDI XORI	\$6 3 \$7 2 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 2 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$7 \$7 2
935 936 937 938 939 940 941 942 943 944 945 946 947	assArrElem_37:	SUB XORI ADD XORI ADDI XORI EXCH XOR EXCH ADDI ADDI ADDI ADDI ADD	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3 \$6 4 \$7 1 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 1 \$6 \$3	999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013		ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB ADD ADDI XORI EXCH	\$6 3 \$7 2 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 2 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$5 \$7 2 \$9 \$6
935 936 937 938 939 940 941 942 943 944 945 946 947	assArrElem_37:	SUB XORI ADD XORI ADDI XORI EXCH XOR EXCH ADDI ADDI ADDI XORI SUB	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3 \$6 4 \$7 1 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 1 \$6 -4 \$6 \$3	999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014		ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB ADDI XORI EXCH XORI	\$6 3 \$7 2 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 2 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$5 \$7 2 \$9 \$6 \$8 \$9
935 936 937 938 939 940 941 942 943 944 945 946 947 948	assArrElem_37:	SUB XORI ADD XORI ADDI XORI EXCH XOR EXCH ADDI ADDI XORI ADDI SUB EXCH	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3 \$6 4 \$7 1 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$7 \$7 1 \$6 -4 \$6 \$3 \$10 \$8	999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015		ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB ADDI XORI EXCH XORI	\$6 3 \$7 2 \$8 \$7 \$8 \$7 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$7 2 \$6 \$3 \$6 \$3 \$6 \$5 \$7 2 \$9 \$6 \$8 \$9 \$9 \$6
935 936 937 938 939 940 941 942 943 944 945 946 947	assArrElem_37:	SUB XORI ADD XORI ADDI XORI EXCH XOR EXCH ADDI ADDI ADDI XORI SUB	\$6 -4 \$6 \$3 \$11 3 \$10 \$11 \$11 3 \$6 \$3 \$6 4 \$7 1 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 1 \$6 -4 \$6 \$3	999 1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014		ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB ADDI XORI EXCH XORI	\$6 3 \$7 2 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 2 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$5 \$7 2 \$9 \$6 \$8 \$9

1018		XORI	\$7 2	1084		ADDI	\$6 6
1019		ADDI	\$6 -5	1085		XORI	\$7 2
1020		SUB	\$6 \$3	1086		EXCH	\$9 \$6
1021		EXCH	\$10 \$8	1087		XOR	\$8 \$9
1022		ADD	\$6 \$3	1088		EXCH	\$9 \$6
1023		ADDI	\$6 5	1089		ADDI	\$8 2
1024		XORI	\$7 2	1090		ADD	\$8 \$7
1025		SUB	\$8 \$7	1091		XORI	\$7 2
1026		ADDI	\$8 -2	1092		ADDI	\$6 -6
1027		EXCH	\$9 \$6	1093		SUB	\$6 \$3
1028		XOR	\$8 \$9	1094		EXCH	\$10 \$8
1029		EXCH	\$9 \$6	1095		ADD	\$6 \$3
1030		XORI	\$7 2	1096		ADDI	\$6 6
1031		ADDI	\$6 -5	1097		XORI	\$7 2
1032	agglantion 20.	SUB	\$6 \$3	1098		SUB ADDI	\$8 \$7 \$8 -2
1033	assArrElem_39:	ADD ADD	\$10 \$0	1099		EXCH	
1034 1035		ADDI	\$6 \$3 \$6 5	1100 1101		XOR	\$9 \$6 \$8 \$9
1036		XORI	\$7 2	1101		EXCH	\$9 \$6
1037		EXCH	\$9 \$6	1102		XORI	\$7 2
1038		XOR	\$8 \$9	1104		ADDI	\$6 -6
1039		EXCH	\$9 \$6	1104		SUB	\$6 \$3
1040		ADDI	\$8 2	1106		ADD	\$6 \$3
1041		ADD	\$8 \$7	1107		ADDI	\$6 4
1042		XORI	\$7 2	1108		XORI	\$7 2
1043		ADDI	\$6 -5	1109		EXCH	\$9 \$6
1044		SUB	\$6 \$3	1110		XOR	\$8 \$9
1045		EXCH	\$10 \$8	1111		EXCH	\$9 \$6
1046		ADD	\$6 \$3	1112		ADDI	\$8 2
1047		ADDI	\$6 5	1113		ADD	\$8 \$7
1048		XORI	\$7 2	1114		XORI	\$7 2
1049		SUB	\$8 \$7	1115		ADDI	\$6 -4
1050		ADDI	\$8 -2	1116		SUB	\$6 \$3
1051		EXCH	\$9 \$6	1117		EXCH	\$10 \$8
1052		XOR	\$8 \$9	1118		ADD	\$6 \$3
1053 $1054$		EXCH XORI	\$9 \$6 \$7 2	1119		ADDI XORI	\$6 4 \$7 2
1054		ADDI	\$6 -5	1120 $1121$		SUB	\$8 \$7
1056		SUB	\$6 \$3	1122		ADDI	\$8 -2
1057		ADD	\$6 \$3	1123		EXCH	\$9 \$6
1058		ADDI	\$6 6	1124		XOR	\$8 \$9
1059		XORI	\$7 2	1125		EXCH	\$9 \$6
1060		EXCH	\$9 \$6	1126		XORI	\$7 2
1061		XOR	\$8 \$9	1127		ADDI	\$6 -4
1062		EXCH	\$9 \$6	1128		SUB	\$6 \$3
1063		ADDI	\$8 2	1129		XORI	\$11 4
1064		ADD	\$8 \$7	1130	assArrElem_41:	ADD	\$10 \$11
1065		XORI	\$7 2	1131		XORI	\$11 4
1066		ADDI	\$6 -6	1132		ADD	\$6 \$3
1067		SUB	\$6 \$3	1133		ADDI	\$6 4
1068		EXCH	\$10 \$8	1134		XORI	\$7 2
1069		ADD	\$6 \$3	1135		EXCH	\$9 \$6
1070		ADDI XORI	\$6 6 \$7 2	1136		XOR EXCH	\$8 \$9 \$9 \$6
1071		SUB		1137		ADDI	\$8 2
1072 $1073$		ADDI	\$8 \$7 \$8 -2	1138 1139		ADDI	\$8 \$7
1073		EXCH	\$9 \$6	1140		XORI	\$7 2
1074		XOR	\$8 \$9	1140		ADDI	\$6 -4
1076		EXCH	\$9 \$6	1142		SUB	\$6 \$3
1077		XORI	\$7 2	1143		EXCH	\$10 \$8
1078		ADDI	\$6 -6	1144		ADD	\$6 \$3
1079		SUB	\$6 \$3	1145		ADDI	\$6 4
1080		XORI	\$11 1	1146		XORI	\$7 2
1081	assArrElem_40:	ADD	\$10 \$11	1147		SUB	\$8 \$7
1082		XORI	\$11 1	1148		ADDI	\$8 -2
1083		ADD	\$6 \$3	1149		EXCH	\$9 \$6

1150		XOR	\$8	\$9	1216		ADD	\$6 \$3
1151		EXCH	\$9	\$6	1217		ADDI	\$6.5
1152		XORI	\$7	2	1218		XORI	\$7 3
1153		ADDI	\$6	-4	1219		SUB	\$8 \$7
1154		SUB	\$6	\$3	1220		ADDI	
1155		ADD	\$6		1221		EXCH	
1156		ADDI	\$6		1222		XOR	\$8 \$9
1157		XORI	\$7		1223		EXCH	
1158		EXCH	\$9		1224		XORI	
1159		XOR	\$8		1225		ADDI	
1160		EXCH		\$6	1226		SUB	\$6 \$3
1161		ADDI	\$8		1227		XORI	
1162		ADD	\$8	\$7	1228	assArrElem_43:	ADD	\$10 \$11
1163		XORI	\$7	3	1229		XORI	\$11 1
1164		ADDI	\$6	-3	1230		ADD	\$6 \$3
1165		SUB	\$6	\$3	1231		ADDI	\$6 5
1166		EXCH	\$10	\$8	1232		XORI	\$7 3
1167		ADD	\$6	\$3	1233		EXCH	\$9 \$6
1168		ADDI	\$6		1234	l .	XOR	\$8 \$9
1169		XORI	\$7	3	1235		EXCH	
1170		SUB		\$7	1236		ADDI	
1171		ADDI	\$8		1237		ADD	\$8 \$7
1172		EXCH		\$6	1237		XORI	
		XOR	\$8			l.	ADDI	
1173					1239			
1174		EXCH		\$6	1240		SUB	\$6 \$3
1175		XORI	\$7		1241		EXCH	
1176		ADDI	\$6		1242		ADD	\$6 \$3
1177		SUB	\$6		1243		ADDI	
1178		XORI	\$11		1244		XORI	
1179	assArrElem_42:	ADD		\$11	1245		SUB	\$8 \$7
1180		XORI	\$11		1246		ADDI	
1181		ADD	\$6	\$3	1247		EXCH	\$9 \$6
1182		ADDI	\$6	3	1248		XOR	\$8 \$9
1183		XORI	\$7	3	1249		EXCH	\$9 \$6
1184		EXCH	\$9	\$6	1250		XORI	\$7 3
1185		XOR	\$8	\$9	1251		ADDI	\$6 -5
1186		EXCH	\$9	\$6	1252		SUB	\$6 \$3
1187		ADDI	\$8	2	1253		ADD	\$6 \$3
1188		ADD	\$8	\$7	1254		ADDI	
1189		XORI	\$7		1255		XORI	
1190		ADDI	\$6		1256		EXCH	
1191		SUB		\$3	1257		XOR	\$8 \$9
1192		EXCH		\$8	1258		EXCH	
1193		ADD	\$6		1259		ADDI	
1194		ADDI	\$6		1260		ADD	\$8 \$7
1194		XORI	\$7		1260		XORI	
1195		SUB	\$8		1261		ADDI	
			\$8			l.		\$6 \$3
1197		ADDI			1263		SUB	
1198		EXCH		\$6	1264		EXCH	
1199		XOR		\$9	1265	1	ADD	\$6 \$3
1200		EXCH		\$6	1266		ADDI	
1201		XORI	\$7		1267		XORI	
1202		ADDI	\$6		1268		SUB	\$8 \$7
1203		SUB	\$6		1269		ADDI	
1204		ADD	\$6		1270		EXCH	
1205		ADDI	\$6	5	1271		XOR	\$8 \$9
1206		XORI	\$7		1272		EXCH	
1207		EXCH	\$9	\$6	1273		XORI	\$7 3
1208		XOR	\$8	\$9	1274		ADDI	\$6 -6
1209		EXCH	\$9	\$6	1275		SUB	\$6 \$3
1210		ADDI	\$8	2	1276	assArrElem_44:	ADD	\$10 \$0
1211		ADD	\$8		1277		ADD	\$6 \$3
1212		XORI	\$7		1278		ADDI	
1213		ADDI	\$6		1279		XORI	
1214		SUB	\$6		1280		EXCH	
1215		EXCH		\$8	1281		XOR	\$8 \$9
-210			~ ± (	- +0	1201	I	AOR	+0 40

1282	EXCH	\$9 \$6	1348		SUB	\$6 \$3
1283	ADDI	\$8 2	1349		ADD	\$6 \$3
1284	ADD	\$8 \$7	1350		ADDI	\$6 3
1285	XORI	\$7 3	1351		XORI	\$7 4
1286	ADDI	\$6 -6	1352		EXCH	\$9 \$6
1287	SUB	\$6 \$3	1353		XOR	\$8 \$9
1288	EXCH	\$10 \$8	1354		EXCH	\$9 \$6
1289	ADD	\$6 \$3	1355		ADDI	\$8 2
1290	ADDI	\$6 6	1356		ADD	\$8 \$7
1291	XORI	\$7 3	1357		XORI	\$7 4
1292	SUB	\$8 \$7	1358		ADDI	\$6 -3
1293	ADDI	\$8 -2	1359		SUB	\$6 \$3
1294	EXCH	\$9 \$6	1360		EXCH	\$10 \$8
1295	XOR	\$8 \$9	1361		ADD	\$6 \$3
1296	EXCH	\$9 \$6	1362		ADDI	\$6 3
1297	XORI	\$7 3	1363		XORI	\$7 4
1298	ADDI	\$6 -6			SUB	\$8 \$7
			1364			
1299	SUB		1365		ADDI	\$8 -2
1300	ADD	\$6 \$3	1366		EXCH	\$9 \$6
1301	ADDI	\$6 4	1367		XOR	\$8 \$9
1302	XORI	\$7 3	1368		EXCH	\$9 \$6
1303	EXCH	\$9 \$6	1369		XORI	\$7 4
1304	XOR	\$8 \$9	1370		ADDI	\$6 -3
1305	EXCH	\$9 \$6	1371		SUB	\$6 \$3
1306	ADDI	\$8 2	1372		XORI	\$11 3
1307	ADD	\$8 \$7	1373	assArrElem_46:	ADD	\$10 \$11
1308	XORI	\$7 3	1374		XORI	\$11 3
1309	ADDI	\$6 -4	1375		ADD	\$6 \$3
1310	SUB	\$6 \$3	1376		ADDI	\$6 3
1311	EXCH	\$10 \$8	1377		XORI	\$7 4
1312	ADD	\$6 \$3	1378		EXCH	\$9 \$6
1313	ADDI	\$6 4	1379		XOR	\$8 \$9
1314	XORI	\$7 3	1380		EXCH	\$9 \$6
1315	SUB	\$8 \$7	1381		ADDI	\$8 2
1316	ADDI	\$8 -2	1382		ADD	\$8 \$7
1317	EXCH	\$9 \$6	1383		XORI	\$7 4
1318	XOR	\$8 \$9	1384		ADDI	\$6 -3
1319	EXCH	\$9 \$6	1385		SUB	\$6 \$3
1320	XORI		1386		EXCH	\$10 \$8
1321	ADDI	\$6 -4	1387		ADD	\$6 \$3
1322	SUB	\$6 \$3	1388		ADDI	\$6 3
1323	XORI	\$11 2	1389		XORI	\$7 4
1324 assArrElem_45:	ADD	\$10 \$11	1390		SUB	\$8 \$7
1325	XORI	\$11 2	1391		ADDI	\$8 -2
1326	ADD	\$6 \$3	1392		EXCH	\$9 \$6
1327	ADDI	\$6 4	1393		XOR	\$8 \$9
1328	XORI	\$7 3	1394		EXCH	\$9 \$6
1329	EXCH	\$9 \$6	1395		XORI	\$7 4
1330	XOR	\$8 \$9	1396		ADDI	\$6 -3
1331	EXCH	\$9 \$6	1397		SUB	\$6 \$3
1332	ADDI	\$8 2	1398		ADD	\$6 \$3
1333	ADD	\$8 \$7	1399		ADDI	\$6 5
1334	XORI	\$7 3	1400		XORI	\$7 4
1335	ADDI	\$6 -4	1401		EXCH	\$9 \$6
1336	SUB	\$6 \$3	1402		XOR	\$8 \$9
1337	EXCH	\$10 \$8	1403		EXCH	\$9 \$6
1338	ADD	\$6 \$3	1404		ADDI	\$8 2
1339	ADDI	\$6 4	1405		ADD	\$8 \$7
1340	XORI	\$7 3	1406		XORI	\$7 4
1341	SUB	\$8 \$7	1407		ADDI	\$6 -5
1342	ADDI	\$8 -2	1408		SUB	\$6 \$3
1342	EXCH	\$9 \$6	1409		EXCH	\$10 \$8
1344	XOR	\$8 \$9	1410		ADD	\$6 \$3
	EXCH	\$9 \$6			ADDI	\$6.5
1345			1411			
1346	XORI	\$7 3 \$6 -4	1412		XORI	\$7 4
1347	ADDI	\$6 -4	1413		SUB	\$8 \$7

1414		ADDI	\$8 -2	1480		EXCH	\$12 \$11
1415		EXCH	\$9 \$6	1481		ADDI	\$11 -10
1416		XOR	\$8 \$9	1482		SUB	\$11 \$3
1417		EXCH	\$9 \$6	1483	assArrElem_48:	ADD	\$10 \$12
1418		XORI	\$7 4	1484	_	ADD	\$11 \$3
1419		ADDI	\$6 -5	1485		ADDI	\$11 10
1420		SUB	\$6 \$3	1486		EXCH	\$12 \$11
1421		ADD	\$11 \$3	1487		ADDI	\$11 -10
1422		ADDI	\$11 10	1488		SUB	\$11 \$3
1423		EXCH	\$12 \$11	1489		ADD	\$6 \$3
1424		ADDI	\$11 -10	1490		ADDI	\$6 6
1425		SUB	\$11 \$3	1491		XORI	\$7 4
1426	assArrElem_47:	ADD	\$10 \$12	1492		EXCH	\$9 \$6
1427		ADD	\$11 \$3	1493		XOR	\$8 \$9
1428		ADDI	\$11 10	1494		EXCH	\$9 \$6
1429		EXCH	\$12 \$11	1495		ADDI	\$8 2
1430		ADDI	\$11 -10	1496		ADD	\$8 \$7
1431		SUB	\$11 \$3	1497		XORI	\$7 4
1432		ADD	\$6 \$3	1498		ADDI	\$6 -6
1433		ADDI	\$6 5	1499		SUB	\$6 \$3
1434		XORI	\$7 4	1500		EXCH	\$10 \$8
1435		EXCH	\$9 \$6	1501		ADD	\$6 \$3
1436		XOR	\$8 \$9	1502		ADDI	\$6 6
1437		EXCH	\$9 \$6	1502		XORI	\$7 4
1438		ADDI	\$8 2	1504		SUB	\$8 \$7
1439		ADD	\$8 \$7	1505		ADDI	\$8 -2
1440		XORI	\$7 4	1506		EXCH	\$9 \$6
1441		ADDI	\$6 -5	1507		XOR	\$8 \$9
1442		SUB	\$6 \$3	1508		EXCH	\$9 \$6
1443		EXCH	\$10 \$8	1509		XORI	\$7 4
1444		ADD	\$6 \$3	1510		ADDI	\$6 -6
1445		ADDI	\$6 5	1511		SUB	\$6 \$3
1446		XORI	\$7 4	1512		ADD	\$6 \$3
1447		SUB	\$8 \$7	1513		ADDI	\$6 4
1448		ADDI	\$8 -2	1514		XORI	\$7 4
1449		EXCH	\$9 \$6	1515		EXCH	\$9 \$6
1450		XOR	\$8 \$9	1516		XOR	\$8 \$9
1451		EXCH	\$9 \$6	1517		EXCH	\$9 \$6
1452		XORI	\$7 4	1518		ADDI	\$8 2
1453		ADDI	\$6 -5	1519		ADD	\$8 \$7
1454		SUB	\$6 \$3	1520		XORI	\$7 4
1455		ADD	\$6 \$3	1521		ADDI	\$6 -4
1456		ADDI	\$6 6	1522		SUB	\$6 \$3
1457		XORI	\$7 4	1523		EXCH	\$10 \$8
1458		EXCH	\$9 \$6	1524		ADD	\$6 \$3
		XOR				ADDI	
1459		EXCH	\$8 \$9 \$9 \$6	1525		XORI	\$6 4 \$7 4
1460				1526			
1461		ADDI	\$8 2	1527		SUB	\$8 \$7
1462		ADD	\$8 \$7	1528		ADDI	\$8 -2
1463		XORI	\$7 4	1529		EXCH	\$9 \$6
1464		ADDI	\$6 -6	1530		XOR	\$8 \$9
1465		SUB	\$6 \$3	1531		EXCH	\$9 \$6
1466		EXCH	\$10 \$8	1532		XORI	\$7 4
1467		ADD	\$6 \$3	1533		ADDI	\$6 -4
1468		ADDI	\$6 6	1534		SUB	\$6 \$3
1469		XORI	\$7 4	1535		XORI	\$11 4
1470		SUB	\$8 \$7	1536	assArrElem_49:	ADD	\$10 \$11
1471		ADDI	\$8 -2	1537		XORI	\$11 4
1472		EXCH	\$9 \$6	1538		ADD	\$6 \$3
1473		XOR	\$8 \$9	1539		ADDI	\$6 4
1474		EXCH	\$9 \$6	1540		XORI	\$7 4
1475		XORI	\$7 4	1541		EXCH	\$9 \$6
1476		ADDI	\$6 -6	1542		XOR	\$8 \$9
1477		SUB	\$6 \$3	1543		EXCH	\$9 \$6
1478		ADD	\$11 \$3	1544		ADDI	\$8 2
1479		ADDI	\$11 10	1545		ADD	\$8 \$7
1113	l		,	1340	I		F ~ Y /

1546		XORI	\$7 4	1612		XORI	\$7 5
1547		ADDI	\$6 -4	1613		EXCH	\$9 \$6
1548		SUB	\$6 \$3	1614		XOR	\$8 \$9
1549		EXCH	\$10 \$8	1615		EXCH	\$9 \$6
1550		ADD	\$6 \$3	1616		ADDI	\$8 2
1551		ADDI	\$6 4	1617		ADD	\$8 \$7
1552		XORI	\$7 4	1618		XORI	\$7 5
1553		SUB	\$8 \$7	1619		ADDI	\$6 -5
1554		ADDI	\$8 -2	1620		SUB	\$6 \$3
1555		EXCH	\$9 \$6	1621		EXCH	\$10 \$8
1556		XOR	\$8 \$9	1622		ADD	\$6 \$3
1557		EXCH	\$9 \$6	1623		ADDI	\$6 5
1558		XORI	\$7 4	1624		XORI	\$7 5
1559		ADDI	\$6 -4	1625		SUB	\$8 \$7
1560		SUB	\$6 \$3	1626		ADDI	\$8 -2
1561		ADD	\$6 \$3	1627		EXCH	\$9 \$6
1562		ADDI	\$6 3	1628		XOR	\$8 \$9
1563		XORI	\$7 5	1629		EXCH	\$9 \$6
1564		EXCH	\$9 \$6	1630		XORI	\$7 5
1565		XOR	\$8 \$9	1631		ADDI	\$6 -5
1566		EXCH	\$9 \$6	1632		SUB	\$6 \$3
1567		ADDI	\$8 2	1633		ADD	\$11 \$3
1568		ADD	\$8 \$7	1634		ADDI	\$11 7
1569		XORI	\$7 5	1635		EXCH	\$12 \$11
1570		ADDI	\$6 -3	1636		ADDI	\$11 -7
1571		SUB	\$6 \$3	1637		SUB	\$11 \$3
1572		EXCH	\$10 \$8		ssArrElem_51:	ADD	\$10 \$12
1573		ADD	\$6 \$3	1639		ADD	\$11 \$3
1574		ADDI XORI	\$6 3 \$7 5	1640		ADDI EXCH	\$11 7
1575		SUB	\$8 \$7	1641 1642		ADDI	\$12 \$11 \$11 -7
1576 $1577$		ADDI	\$8 -2	1643		SUB	\$11 \$3
1578		EXCH	\$9 \$6	1644		ADD	\$6 \$3
1579		XOR	\$8 \$9	1645		ADDI	\$6 5
1580		EXCH	\$9 \$6	1646		XORI	\$7 5
1581		XORI	\$7 5	1647		EXCH	\$9 \$6
1582		ADDI	\$6 -3	1648		XOR	\$8 \$9
1583		SUB	\$6 \$3	1649		EXCH	\$9 \$6
1584		XORI	\$11 4	1650		ADDI	\$8 2
1585	assArrElem_50:	ADD	\$10 \$11	1651		ADD	\$8 \$7
1586		XORI	\$11 4	1652		XORI	\$7 5
1587		ADD	\$6 \$3	1653		ADDI	\$6 -5
1588		ADDI	\$6 3	1654		SUB	\$6 \$3
1589		XORI	\$7 5	1655		EXCH	\$10 \$8
1590		EXCH	\$9 \$6	1656		ADD	\$6 \$3
1591		XOR	\$8 \$9	1657		ADDI	\$6 5
1592		EXCH	\$9 \$6	1658		XORI	\$7 5
1593		ADDI	\$8 2	1659		SUB	\$8 \$7
1594		ADD	\$8 \$7	1660		ADDI	\$8 -2
1595		XORI	\$7 5	1661		EXCH	\$9 \$6
1596		ADDI	\$6 -3	1662		XOR	\$8 \$9
1597		SUB	\$6 \$3	1663		EXCH	\$9 \$6
1598		EXCH	\$10 \$8	1664		XORI	\$7 5
1599		ADD	\$6 \$3	1665		ADDI	\$6 -5
1600		ADDI	\$6 3 \$7 5	1666		SUB ADD	\$6 \$3 \$6 \$3
1601 $1602$		XORI SUB	\$7 5 \$8 \$7	1667		ADDI	\$6 \$3 \$6 6
1602		ADDI	\$8 \$1 \$8 -2	1668 1669		XORI	\$0 0 \$7 5
1604		EXCH	\$9 \$6	1670		EXCH	\$9 \$6
1605		XOR	\$8 \$9	1671		XOR	\$8 \$9
1606		EXCH	\$9 \$6	1672		EXCH	\$9 \$6
1607		XORI	\$7 5	1673		ADDI	\$8 2
1608		ADDI	\$6 -3	1674		ADD	\$8 \$7
1609		SUB	\$6 \$3	1675		XORI	\$7 5
1610		ADD	\$6 \$3	1676		ADDI	\$6 -6
1611		ADDI	\$6 5	1677		SUB	\$6 \$3
				1			•

1678		EXCH	\$10 \$8	1744		XORI	\$7 5
1679		ADD	\$6 \$3	1745		ADDI	\$6 -4
1680		ADDI	\$6 6	1746		SUB	\$6 \$3
1681		XORI	\$7 5	1747		XORI	\$11 5
1682		SUB	\$8 \$7	1748	assArrElem_53:	ADD	\$10 \$11
1683		ADDI	\$8 -2	1749		XORI	\$11 5
1684		EXCH	\$9 \$6	1750		ADD	\$6 \$3
		XOR	\$8 \$9			ADDI	
1685				1751			\$6 4
1686		EXCH	\$9 \$6	1752		XORI	\$7 5
1687		XORI	\$7 5	1753		EXCH	\$9 \$6
1688		ADDI	\$6 -6	1754		XOR	\$8 \$9
1689		SUB	\$6 \$3	1755		EXCH	\$9 \$6
1690		ADD	\$11 \$3	1756		ADDI	\$8 2
1691		ADDI	\$11 8	1757		ADD	\$8 \$7
1692		EXCH	\$12 \$11	1758		XORI	\$7 5
1693		ADDI	\$11 -8	1759		ADDI	\$6 -4
1694		SUB	\$11 \$3	1760		SUB	\$6 \$3
1695	assArrElem_52:	ADD	\$10 \$12	1761		EXCH	\$10 \$8
1	assairteem_J2.						
1696		ADD	\$11 \$3	1762		ADD	\$6 \$3
1697		ADDI	\$11 8	1763		ADDI	\$6 4
1698		EXCH	\$12 \$11	1764		XORI	\$7 5
1699		ADDI	\$11 -8	1765		SUB	\$8 \$7
1700		SUB	\$11 \$3	1766		ADDI	\$8 -2
1701		ADD	\$6 \$3	1767		EXCH	\$9 \$6
1702		ADDI	\$6 6	1768		XOR	\$8 \$9
1703		XORI	\$7 5	1769		EXCH	\$9 \$6
1704		EXCH	\$9 \$6	1770		XORI	\$7 5
1705		XOR	\$8 \$9	1771		ADDI	\$6 -4
1706		EXCH	\$9 \$6	1772		SUB	\$6 \$3
1707		ADDI	\$8 2	1773		ADD	\$6 \$3
1708		ADD	\$8 \$7	1774		ADDI	\$6 3
1709		XORI	\$7 5	1775		XORI	\$7 6
		ADDI	\$6 -6			EXCH	\$9 \$6
1710				1776			
1711		SUB	\$6 \$3	1777		XOR	\$8 \$9
1712		EXCH	\$10 \$8	1778		EXCH	\$9 \$6
1713		ADD	\$6 \$3	1779		ADDI	\$8 2
1714		ADDI	\$6 6	1780		ADD	\$8 \$7
1715		XORI	\$7 5	1781		XORI	\$7 6
1716		SUB	\$8 \$7	1782		ADDI	\$6 -3
1717		ADDI	\$8 -2	1783		SUB	\$6 \$3
1718		EXCH	\$9 \$6	1784		EXCH	\$10 \$8
1719		XOR	\$8 \$9	1785		ADD	\$6 \$3
1720		EXCH	\$9 \$6	1786		ADDI	\$6 3
1721		XORI	\$7 5	1787		XORI	\$7 6
1722		ADDI	\$6 -6	1788		SUB	\$8 \$7
1723		SUB	\$6 \$3	1789		ADDI	\$8 -2
1724		ADD	\$6 \$3	1790		EXCH	\$9 \$6
1725		ADDI	\$6 4	1791		XOR	\$8 \$9
1726		XORI	\$7 5	1792		EXCH	\$9 \$6
1727		EXCH	\$9 \$6	1793		XORI	\$7 6
1		XOR	\$8 \$9	1793		ADDI	\$6 -3
1728			\$9 \$6			SUB	
1729		EXCH		1795			\$6 \$3
1730		ADDI	\$8 2	1796		XORI	\$11 5
1731		ADD	\$8 \$7	1797	assArrElem_54:	ADD	\$10 \$11
1732		XORI	\$7 5	1798		XORI	\$11 5
1733		ADDI	\$6 -4	1799		ADD	\$6 \$3
1734		SUB	\$6 \$3	1800		ADDI	\$6 3
1735		EXCH	\$10 \$8	1801		XORI	\$7 6
1736		ADD	\$6 \$3	1802		EXCH	\$9 \$6
1737		ADDI	\$6 4	1803		XOR	\$8 \$9
1738		XORI	\$7 5	1804		EXCH	\$9 \$6
1739		SUB	\$8 \$7	1805		ADDI	\$8 2
1740		ADDI	\$8 -2	1806		ADD	\$8 \$7
1741		EXCH	\$9 \$6	1807		XORI	\$7 6
1742		XOR	\$8 \$9	1808		ADDI	\$6 -3
1743		EXCH	\$9 \$6	1809		SUB	\$6 \$3
1140			12 70	1000	I		+ 0 40

1010		EXCH	¢10	¢0 1070		ADD	\$8 \$7
1810 1811		ADD	\$10 \$6 \$			XORI	\$7 6
1812		ADDI	\$6 3			ADDI	\$6 -6
1813		XORI	\$7 6			SUB	\$6 \$3
1814		SUB	\$8 \$			EXCH	\$10 \$8
1815		ADDI	\$8 -			ADD	\$6 \$3
1816		EXCH	\$9 \$			ADDI	\$6 6
1817		XOR		9 1883		XORI	\$7 6
1818		EXCH	\$9 \$			SUB	\$8 \$7
1819		XORI	\$7 6			ADDI	\$8 -2
1820		ADDI	\$6 -			EXCH	\$9 \$6
1821		SUB	\$6 \$			XOR	\$8 \$9
1822		ADD	\$6 \$			EXCH	\$9 \$6
1823		ADDI	\$6 5			XORI	\$7 6
1824		XORI	\$7 6	1890		ADDI	\$6 -6
1825		EXCH	\$9 \$	66 1891		SUB	\$6 \$3
1826		XOR	\$8 \$	9 1892	assArrElem_56:	ADD	\$10 \$0
1827		EXCH	\$9 \$	66 1893		ADD	\$6 \$3
1828		ADDI	\$8 2	1894		ADDI	\$6 6
1829		ADD	\$8 \$	7 1895		XORI	\$7 6
1830		XORI	\$7 6			EXCH	\$9 \$6
1831		ADDI	\$6 -			XOR	\$8 \$9
1832		SUB	\$6 \$			EXCH	\$9 \$6
1833		EXCH	\$10			ADDI	\$8 2
1834		ADD	\$6 \$			ADD	\$8 \$7
1835		ADDI	\$6 5			XORI	\$7 6
1836		XORI	\$7 6			ADDI	\$6 -6
1837		SUB	\$8 \$			SUB	\$6 \$3
1838		ADDI	\$8 -			EXCH	\$10 \$8
1839		EXCH XOR	\$9 \$			ADD ADDI	\$6 \$3
1840 1841		EXCH	\$8 \$ \$9 \$			XORI	\$6 6 \$7 6
1842		XORI	\$7 6			SUB	\$8 \$7
1843		ADDI	\$6 -			ADDI	\$8 -2
1844		SUB	\$6 \$			EXCH	\$9 \$6
1845	assArrElem_55:	ADD	\$10			XOR	\$8 \$9
1846	_	ADD	\$6 \$			EXCH	\$9 \$6
1847		ADDI	\$6 5			XORI	\$7 6
1848		XORI	\$7 6	1914		ADDI	\$6 -6
1849		EXCH	\$9 \$	66 1915		SUB	\$6 \$3
1850		XOR	\$8 \$	9 1916		ADD	\$6 \$3
1851		EXCH	\$9 \$	66 1917		ADDI	\$6 4
1852		ADDI	\$8 2	1918		XORI	\$7 6
1853		ADD	\$8 \$			EXCH	\$9 \$6
1854		XORI	\$7 6			XOR	\$8 \$9
1855		ADDI	\$6 -			EXCH	\$9 \$6
1856		SUB	\$6 \$			ADDI	\$8 2
1857		EXCH	\$10			ADD	\$8 \$7
1858		ADD	\$6 \$			XORI	\$7 6
1859		ADDI	\$6 5			ADDI	\$6 -4
1860		XORI	\$7 6 \$8 \$			SUB EXCH	\$6 \$3 \$10 \$8
1861 1862		SUB ADDI	\$8 -			ADD	\$10 \$8 \$6 \$3
1863		EXCH	\$9 \$			ADDI	\$6 4
1864		XOR	\$8 \$			XORI	\$7 6
1865		EXCH	\$9 \$			SUB	\$8 \$7
1866		XORI	\$7 6			ADDI	\$8 -2
1867		ADDI	\$6 -			EXCH	\$9 \$6
1868		SUB	\$6 \$			XOR	\$8 \$9
1869		ADD	\$6 \$			EXCH	\$9 \$6
1870		ADDI	\$6 6			XORI	\$7 6
1871		XORI	\$7 6	1937		ADDI	\$6 -4
1872		EXCH	\$9 \$	66 1938		SUB	\$6 \$3
1873		XOR	\$8 \$	9 1939		XORI	\$11 4
1874		EXCH	\$9 \$		assArrElem_57:	ADD	\$10 \$11
1875		ADDI	\$8 2	1941		XORI	\$11 4

1942	ADD	\$6 \$3	2008		EXCH	\$9 \$6
1943	ADDI	\$6 4	2009		XOR	\$8 \$9
1944	XORI	\$7 6	2010		EXCH	\$9 \$6
1945	EXCH	\$9 \$6	2011		XORI	\$7 7
1946	XOR	\$8 \$9	2012		ADDI	\$6 -3
1947	EXCH	\$9 \$6	2013		SUB	\$6 \$3
1948	ADDI	\$8 2	2014		ADD	\$6 \$3
1949	ADD	\$8 \$7	2015		ADDI	\$6 5
1950	XORI	\$7 6	2016		XORI	\$7 7
1951	ADDI	\$6 -4	2017		EXCH	\$9 \$6
1952	SUB	\$6 \$3	2018		XOR	\$8 \$9
1953	EXCH	\$10 \$8	2019		EXCH	\$9 \$6
1954	ADD	\$6 \$3	2020		ADDI	\$8 2
1955	ADDI	\$6 4	2021		ADD	\$8 \$7
1956	XORI	\$7 6	2022		XORI	\$7 7
1957	SUB	\$8 \$7	2023		ADDI	\$6 -5
1958	ADDI	\$8 -2	2024		SUB	\$6 \$3
1959	EXCH	\$9 \$6	2025		EXCH	\$10 \$8
1960	XOR EXCH	\$8 \$9	2026		ADD ADDI	\$6 \$3 \$6 5
1961	XORI	\$9 \$6 \$7 6	2027		XORI	\$7 7
1962 1963	ADDI	\$6 -4	2028 2029		SUB	\$8 \$7
1964	SUB	\$6 \$3	2029		ADDI	\$8 -2
1965	ADD	\$6 \$3	2031		EXCH	\$9 \$6
1966	ADDI	\$6 3	2032		XOR	\$8 \$9
1967	XORI	\$7 7	2033		EXCH	\$9 \$6
1968	EXCH	\$9 \$6	2034		XORI	\$7 7
1969	XOR	\$8 \$9	2035		ADDI	\$6 -5
1970	EXCH	\$9 \$6	2036		SUB	\$6 \$3
1971	ADDI	\$8 2	2037		ADD	\$11 \$3
1972	ADD	\$8 \$7	2038		ADDI	\$11 10
1973	XORI	\$7 7	2039		EXCH	\$12 \$11
		AC 3			ADDI	\$11 -10
1974	ADDI	\$6 -3	2040		ADDI	ATT TO
1974 1975	ADDI SUB	\$6 = 3 \$6 \$3	2040 2041		SUB	\$11 \$3
				assArrElem_59:		
1975	SUB EXCH ADD	\$6 \$3 \$10 \$8 \$6 \$3	2041	assArrElem_59:	SUB	\$11 \$3
1975 1976	SUB EXCH ADD ADDI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3	2041 2042	assArrElem_59:	SUB ADD ADD ADDI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10
1975 1976 1977 1978 1979	SUB EXCH ADD ADDI XORI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7	2041 2042 2043 2044 2045	assArrElem_59:	SUB ADD ADD ADDI EXCH	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11
1975 1976 1977 1978 1979	SUB EXCH ADD ADDI XORI SUB	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7	2041 2042 2043 2044 2045 2046	assArrElem_59:	SUB ADD ADD ADDI EXCH ADDI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10
1975 1976 1977 1978 1979 1980	SUB EXCH ADD ADDI XORI SUB ADDI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2	2041 2042 2043 2044 2045 2046 2047	assArrElem_59:	SUB ADD ADDI ADDI EXCH ADDI SUB	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3
1975 1976 1977 1978 1979 1980 1981	EXCH ADD ADDI XORI SUB ADDI EXCH	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6	2041 2042 2043 2044 2045 2046 2047 2048	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3
1975 1976 1977 1978 1979 1980 1981 1982	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9	2041 2042 2043 2044 2045 2046 2047 2048 2049	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5
1975 1976 1977 1978 1979 1980 1981 1982 1983	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XOR	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XOR EXCH XORI ADDI SUB	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADDI SUB	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1984 1985 1986 1987 1988 1989 assArrElem_58:	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADDI SUB	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5 \$10 \$11	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI ADDI ADDI ADDI ADDI ADDI ADDI ADD	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 2 \$8 \$7
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADDI SUB	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5 \$10 \$11 \$11 5	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 assArrElem_58:	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADD XORI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3	2041 2042 2043 2044 2045 2046 2047 2048 2050 2051 2052 2053 2054 2055 2056 2057	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI ADDI XORI ADDI ADDI ADDI ADDI ADDI ADDI ADDI AD	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 assArrElem_58:	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADD XORI ADD	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5 \$10 \$11 \$11 5	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI XORI ADDI XORI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 2 \$8 \$7 \$7 7
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 assArrElem_58:	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADDI XORI ADD ADDI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 3	2041 2042 2043 2044 2045 2046 2047 2048 2050 2051 2052 2053 2054 2055 2056 2057 2058	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$9 \$6 \$8 \$3
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1986 1987 1988 1989 assArrElem_58:	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADD XORI ADD XORI ADD XORI ADD XORI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 3 \$7 7	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI XORI ADDI SUB EXCH ADDI SUB EXCH	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1988 1989 1990 1991 1992 1993 1994	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 3 \$7 7 \$9 \$6	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH ADDI XOR EXCH ADDI XORI ADDI XORI ADDI XORI ADDI ADDI ADDI ADDI ADDI ADDI ADDI AD	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 \$5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$6 \$3
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1988 1988 1989 1990 1991 1992 1993 1994 1995	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADDI XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 3 \$7 7 \$9 \$6 \$8 \$9	2041 2042 2043 2044 2045 2046 2047 2048 2049 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH ADDI XORI EXCH ADDI ADDI XORI ADDI ADDI ADDI ADDI ADDI ADDI ADDI AD	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1999 1991 1992 1993 1994 1995 1996	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADD XORI ADD XORI ADD XORI ADD ADDI XORI EXCH XOR EXCH XOR EXCH ADDI ADDI ADDI ADDI ADDI ADDI ADDI ADD	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 3 \$7 7 7 \$9 \$6 \$8 \$9 \$9 \$6	2041 2042 2043 2044 2045 2046 2047 2048 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI EXCH XORI EXCH XOR EXCH ADDI ADDI ADDI ADDI ADDI ADDI ADDI ADD	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 \$5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$10 \$10 \$8 \$10 \$10 \$8 \$10 \$10 \$8 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 assArrElem_58:  1990 1991 1992 1993 1994 1995 1996 1997 1998	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 -2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 -3 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 3 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$10 \$11 \$11 5 \$10	2041 2042 2043 2044 2045 2046 2047 2048 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2064	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI ADDI XORI ADDI XORI ADDI XORI ADDI SUB EXCH ADDI SUB EXCH ADDI ADDI SUB EXCH ADDI ADDI EXCH ADDI EXCH ADDI ADDI EXCH ADDI EXCH	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 \$5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$6 \$5 \$7 7 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$6 \$3 \$6
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1999 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI SUB XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD	\$6 \$3 \$10 \$8 \$6 \$3 \$6 \$3 \$7 7 \$8 \$7 \$8 \$-2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 \$-3 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$7 \$7 7 \$9 \$6 \$1 7 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5	2041 2042 2043 2044 2045 2046 2047 2059 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI ADDI XORI ADDI XORI ADDI XORI ADDI SUB EXCH ADDI SUB EXCH ADDI ADDI SUB EXCH ADDI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI XORI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 \$5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$10 \$10 \$8 \$10 \$10 \$8 \$10 \$8 \$10 \$10 \$8 \$10 \$10 \$8 \$10 \$10 \$8 \$10 \$10 \$8 \$10 \$10 \$10 \$8 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 assArrElem_58: 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI SUB	\$6 \$3 \$10 \$8 \$6 \$3 \$6 \$3 \$7 7 \$8 \$7 \$8 \$7 \$8 \$2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$7 \$7 7 \$9 \$6 \$3 \$10 \$11 \$11 5 \$10 \$11 \$10 \$10 \$11 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10	2041 2042 2043 2044 2045 2046 2047 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2066 2066	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI XORI ADD XORI ADDI XORI ADDI SUB EXCH ADDI SUB EXCH ADDI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI SUB ADDI XORI SUB ADDI XORI SUB ADDI EXCH XOR EXCH	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 \$5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 br>\$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 assArrElem_58: 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADDI XORI EXCH XOR EXCH ADDI XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADDI SUB EXCH	\$6 \$3 \$10 \$8 \$6 \$3 \$6 \$3 \$7 7 \$8 \$7 \$8 \$7 \$8 \$2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$7 7 \$9 \$6 \$3 \$11 5 \$10 \$11 \$11 \$10 \$10 br>\$10 \$10 \$10 br>\$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10	2041 2042 2043 2044 2045 2046 2047 2048 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2066	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI XORI XORI XORI XORI XORI XORI XOR	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$6 \$3 \$10 \$8 \$6 \$3 \$7 7 \$7 7 \$7 7 \$7 7 \$7 7 \$7 7 \$7 7 \$
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 assArrElem_58: 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD	\$6 \$3 \$10 \$8 \$6 \$3 \$6 \$3 \$7 7 \$8 \$7 \$8 \$7 \$8 \$2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$11 5 \$10 \$11 \$11 \$10 \$11 \$10 \$10 \$11 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10	2041 2042 2043 2044 2045 2046 2047 2048 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$6 5 \$7 7 \$6 \$3 \$6 \$3 \$7 7 \$7 \$6 \$6 \$8 \$7 \$7 \$7 \$6 \$6 \$8 \$7 \$7 \$8 \$6 \$8 \$7 \$7 \$8 \$6 \$8 \$7 \$7 7 \$8 \$6 \$8 \$7 \$7 7 \$7 \$6 \$6 \$8 \$7 7 \$7 \$6 \$6 \$7 7 \$8 \$6 \$8 \$7 7 \$8 \$6 \$7 7 \$8 \$6 \$7 7 \$8 \$6 \$7 7 \$7 7 \$7 86 \$7 7 \$7 86 \$7 7 \$7 7 \$7 86 \$7 7 \$7 7 \$7 86 \$7 7 \$7 7 \$7 86 \$7 7 \$7 7 \$7 7 \$7 86 \$7 7 \$7 7 \$7 8 \$7 8
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 assArrElem_58: 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD ADDI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 \$7 \$8 \$2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$7 \$7 7 \$9 \$6 \$8 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$3 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$	2041 2042 2043 2044 2045 2046 2047 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2069 2069	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI SUB ADDI XORI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$6 5 \$7 7 \$6 \$3 \$6 5 \$7 7 \$6 \$3 \$7 7 \$6 \$3 \$6 \$3 \$7 7 \$7 \$6 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$7 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$8 \$7 7 \$7 7 \$7 7 8 8 \$7 7 \$7 7 8 8 \$7 7 \$8 \$7 7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 assArrElem_58: 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 \$3 \$7 7 \$8 \$7 \$8 \$7 \$8 \$2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 3 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$7 \$7 7 \$9 \$6 \$8 \$7 7 \$9 \$6 \$8 \$7 7 \$9 \$6 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$	2041 2042 2043 2044 2045 2046 2047 2048 2050 2051 2052 2053 2054 2055 2056 2057 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2069 2070 2070	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI SUB EXCH ADD ADDI XORI ADDI SUB EXCH ADD ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 \$5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$6 \$5 \$7 7 \$6 \$6 \$7 7 \$8 \$6 \$7 7 \$8 \$6 \$8 \$7 \$7 \$7 \$6 \$6 \$7 7 \$6 \$6 \$7 7 \$7 \$6 \$6 \$7 7 \$7 \$6 \$6 \$7 7 \$7 \$6 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 8 \$7 7 \$7 7 \$7 8 \$7 9 \$7 8 \$7 9 \$7 8 \$7 7 \$7 8 \$7 9 \$7 8 \$7
1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 assArrElem_58: 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004	SUB EXCH ADD ADDI XORI SUB ADDI EXCH XOR EXCH XORI ADDI XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD XORI ADD ADDI	\$6 \$3 \$10 \$8 \$6 \$3 \$6 3 \$7 7 \$8 \$7 \$8 \$7 \$8 \$2 \$9 \$6 \$8 \$9 \$9 \$6 \$7 7 \$6 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$7 \$7 7 \$9 \$6 \$8 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$3 \$11 5 \$10 \$11 \$11 5 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$3 \$6 \$3 \$6 \$3 \$7 7 \$9 \$6 \$8 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$6 \$3 \$	2041 2042 2043 2044 2045 2046 2047 2050 2051 2052 2053 2054 2055 2056 2057 2058 2059 2060 2061 2062 2063 2064 2065 2066 2067 2068 2069 2069 2069	assArrElem_59:	SUB ADD ADDI EXCH ADDI SUB ADD ADDI XORI EXCH XOR EXCH ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI ADDI XORI SUB ADDI XORI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB ADDI SUB	\$11 \$3 \$10 \$12 \$11 \$3 \$11 10 \$12 \$11 \$11 -10 \$11 \$3 \$6 \$3 \$6 5 \$7 7 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$2 \$8 \$7 \$7 7 \$6 -5 \$6 \$3 \$10 \$8 \$6 \$3 \$6 5 \$7 7 \$6 \$3 \$6 5 \$7 7 \$6 \$3 \$7 7 \$6 \$3 \$6 \$3 \$7 7 \$7 \$6 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$9 \$9 \$6 \$8 \$7 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$6 \$7 7 \$7 \$8 \$7 7 \$7 7 \$7 7 8 8 \$7 7 \$7 7 8 8 \$7 7 \$8 \$7 7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8 8 \$7 8

1			40.46	1			26.42
2074		EXCH	\$9 \$6	2140		ADD	\$6 \$3
2075		XOR	\$8 \$9	2141		ADDI	\$6 4
2076		EXCH	\$9 \$6	2142		XORI	\$7 7
2077		ADDI	\$8 2	2143		SUB	\$8 \$7
2078		ADD	\$8 \$7	2144		ADDI	\$8 -2
2079		XORI	\$7 7	2145		EXCH	\$9 \$6
2080		ADDI	\$6 -6	2146		XOR	\$8 \$9
2081		SUB	\$6 \$3	2147		EXCH	\$9 \$6
2082		EXCH	\$10 \$8	2148		XORI	\$7 7
				1			
2083		ADD	\$6 \$3	2149		ADDI	\$6 -4
2084		ADDI	\$6 6	2150		SUB	\$6 \$3
2085		XORI	\$7 7	2151		XORI	\$11 6
2086		SUB	\$8 \$7	2152	assArrElem_61:	ADD	\$10 \$11
2087		ADDI	\$8 -2	2153		XORI	\$11 6
2088		EXCH	\$9 \$6	2154		ADD	\$6 \$3
2089		XOR	\$8 \$9	2155		ADDI	\$6 4
2090		EXCH	\$9 \$6	2156		XORI	\$7 7
2091		XORI	\$7 7	2157		EXCH	\$9 \$6
2092		ADDI	\$6 -6	2158		XOR	\$8 \$9
2093		SUB	\$6 \$3	2159		EXCH	\$9 \$6
2094		ADD	\$11 \$3	2160		ADDI	\$8 2
2095		ADDI	\$11 10	2161		ADD	\$8 \$7
2096		EXCH	\$12 \$11	2162		XORI	\$7 7
2097		ADDI	\$11 -10	2163		ADDI	\$6 -4
2098		SUB	\$11 \$3	2164		SUB	\$6 \$3
2099	assArrElem_60:	ADD	\$10 \$12	2165		EXCH	\$10 \$8
2100		ADD	\$11 \$3	2166		ADD	\$6 \$3
2101		ADDI	\$11 10	2167		ADDI	\$6 4
2102		EXCH	\$12 \$11	2168		XORI	\$7 7
2103		ADDI	\$11 -10	2169		SUB	\$8 \$7
2104		SUB	\$11 \$3	2170		ADDI	\$8 -2
2105		ADD	\$6 \$3	2171		EXCH	\$9 \$6
2106		ADDI	\$6 6	2172		XOR	\$8 \$9
2107		XORI	\$7 7	2173		EXCH	\$9 \$6
2108		EXCH	\$9 \$6	2174		XORI	\$7 7
2109		XOR	\$8 \$9	2175		ADDI	\$6 -4
2110		EXCH	\$9 \$6	2176		SUB	\$6 \$3
2111		ADDI	\$8 2	2177	l_initRules_2_bot:	BRA	
2112		ADD	\$8 \$7		l_initRules_2_top		
2113		XORI	\$7 7	2178	<pre>l_initTape_3_top:</pre>	BRA	
2114		ADDI	\$6 -6		l_initTape_3_bot		
2115		SUB	\$6 \$3	2179		ADDI	\$1 1
2116		EXCH	\$10 \$8	2180		EXCH	\$2 \$1
2117		ADD	\$6 \$3	2181		EXCH	\$3 \$1
2118		ADDI	\$6 6	2182		ADDI	\$1 -1
2119		XORI	\$7 7	2183	l_initTape_3:	SWAPBR	
2120		SUB	\$8 \$7	2184		NEG	\$2
2121		ADDI	\$8 -2	2185		ADDI	\$1 1
2122		EXCH	\$9 \$6	2186		EXCH	\$3 \$1
		XOR				EXCH	\$2 \$1
2123			\$8 \$9	2187			
2124		EXCH		2188	11D1 140	ADDI	\$1 -1
2125		XORI	\$7 7	2189	localBlock_149:	XOR	\$6 \$1
2126		ADDI	\$6 -6	2190		XOR	\$7 \$0
2127		SUB	\$6 \$3	2191		EXCH	\$7 \$1
2128		ADD	\$6 \$3	2192		ADDI	\$1 -1
2129		ADDI	\$6 4	2193	localBlock_148:	XOR	\$7 \$1
2130		XORI	\$7 7	2194		XOR	\$8 \$0
2131		EXCH	\$9 \$6	2195		EXCH	\$8 \$1
2132		XOR	\$8 \$9	2196		ADDI	\$1 -1
2133		EXCH	\$9 \$6	2197	localBlock_147:	XOR	\$8 \$1
2134		ADDI	\$8 2	2198	_	XOR	\$9 \$0
2135		ADD	\$8 \$7	2199		EXCH	\$9 \$1
2136		XORI	\$7 7	2200		ADDI	\$1 -1
2137		ADDI	\$6 -4	2200	localBlock_146:	XOR	\$9 \$1
2137		SUB	\$6 \$3	2201	10001D100N_110.	XOR	\$10 \$0
2139		EXCH	\$10 \$8	2203		EXCH	\$10 \$1

2204		ADDI	\$1 -1	2270	I	EXCH	\$11 \$1
	11011-145.						
2205	localBlock_145:	XOR	\$10 \$1	2271		ADDI	\$1 1
2206		XOR	\$11 \$0	2272		EXCH	\$12 \$1
2207		EXCH	\$11 \$1	2273	obj_con_63_i:	ADDI	\$12 -8
2208		ADDI	\$1 -1	2274		ADDI	\$1 1
2209		EXCH	\$3 \$1	2275		EXCH	\$6 \$1
2210		ADDI	\$1 -1	2276		ADDI	\$1 1
2211		EXCH	\$10 \$1	2277		EXCH	\$7 \$1
		ADDI	\$1 -1	2278		ADDI	\$1 1
2212							
2213		EXCH	\$9 \$1	2279		EXCH	\$8 \$1
2214		ADDI	\$1 -1	2280		ADDI	\$1 1
2215		EXCH	\$8 \$1	2281		EXCH	\$9 \$1
2216		ADDI	\$1 -1	2282		ADDI	\$1 1
2217		EXCH	\$7 \$1	2283		EXCH	\$10 \$1
2218		ADDI	\$1 -1	2284		ADDI	\$1 1
2219		EXCH	\$6 \$1	2285		EXCH	\$3 \$1
2220		ADDI	\$1 -1	2286		XORI	\$12 10
	-h-i 60.						
2221	obj_con_62:	ADDI	\$12 8	2287		EXCH	\$12 \$11
2222		EXCH	\$12 \$1	2288		ADDI	\$11 1
2223		ADDI	\$1 -1	2289		XORI	\$12 1
2224		EXCH	\$11 \$1	2290		EXCH	\$12 \$11
2225		ADDI	\$1 -1	2291	obj_con_63_bot:	ADDI	\$11 -1
2226		BRA	l_malloc	2292		EXCH	\$11 \$7
2227		ADDI	\$1 1	2293		EXCH	\$3 \$1
2228		EXCH	\$11 \$1	2294		ADDI	\$1 -1
			\$1 1				
2229		ADDI		2295		EXCH	\$10 \$1
2230		EXCH	\$12 \$1	2296		ADDI	\$1 -1
2231	obj_con_62_i:	ADDI	\$12 -8	2297		EXCH	\$9 \$1
2232		ADDI	\$1 1	2298		ADDI	\$1 -1
2233		EXCH	\$6 \$1	2299		EXCH	\$8 \$1
2234		ADDI	\$1 1	2300		ADDI	\$1 -1
2235		EXCH	\$7 \$1	2301		EXCH	\$7 \$1
2236		ADDI	\$1 1	2302		ADDI	\$1 -1
2237		EXCH	\$8 \$1	2303		EXCH	\$6 \$1
1							
2238		ADDI	\$1 1	2304		ADDI	\$1 -1
2239		EXCH	\$9 \$1	2305	obj_con_64:	ADDI	\$12 8
2240		ADDI	\$1 1	2306		EXCH	\$12 \$1
2241		EXCH	\$10 \$1	2307		ADDI	\$1 -1
2242		ADDI	\$1 1	2308		EXCH	\$11 \$1
2243		EXCH	\$3 \$1	2309		ADDI	\$1 -1
2244		XORI	\$12 10	2310		BRA	l_malloc
2245		EXCH	\$12 \$11	2311		ADDI	\$1 1
2246		ADDI	\$11 1	2312		EXCH	\$11 \$1
2247		XORI	\$12 1	2313		ADDI	\$1 1
2248		EXCH	\$12 \$11	2314		EXCH	\$12 \$1
2249	obj_con_62_bot:	ADDI	\$11 -1	2315	obj_con_64_i:	ADDI	\$12 -8
2250		EXCH	\$11 \$6	2316		ADDI	\$1 1
2251		EXCH	\$3 \$1	2317		EXCH	\$6 \$1
2252		ADDI	\$1 -1	2318		ADDI	\$1 1
2253		EXCH	\$10 \$1	2319		EXCH	\$7 \$1
2254		ADDI	\$1 -1	2320		ADDI	\$1 1
2255		EXCH	\$9 \$1	2321		EXCH	\$8 \$1
2256		ADDI	\$1 -1	2322		ADDI	\$1 1
2257		EXCH	\$8 \$1	2323		EXCH	\$9 \$1
2258		ADDI	\$1 -1	2324		ADDI	\$1 1
2259		EXCH	\$7 \$1	2325		EXCH	\$10 \$1
2260		ADDI	\$1 -1	2326		ADDI	\$1 1
2261		EXCH	\$6 \$1	2327		EXCH	\$3 \$1
2262		ADDI	\$1 -1	2328		XORI	\$12 10
2263	obj_con_63:	ADDI	\$12 8	2329		EXCH	\$12 \$11
2264	J= =	EXCH	\$12 \$1	2330		ADDI	\$11 1
2265		ADDI	\$1 -1	2331		XORI	\$12 1
1							
2266		EXCH	\$11 \$1	2332		EXCH	\$12 \$11
2267		ADDI	\$1 -1	2333	obj_con_64_bot:	ADDI	\$11 -1
2268		BRA	l_malloc	2334		EXCH	\$11 \$8
2269		ADDI	\$1 1	2335		EXCH	\$3 \$1

			61 1		İ		61 1
2336		ADDI	\$1 -1	2402		ADDI	\$1 1
2337		EXCH	\$10 \$1	2403		EXCH	\$7 \$1
2338		ADDI	\$1 -1	2404		ADDI	\$1 1
2339		EXCH	\$9 \$1	2405		EXCH	\$8 \$1
2340		ADDI	\$1 -1	2406		ADDI	\$1 1
2341		EXCH	\$8 \$1	2407		EXCH	\$9 \$1
2342		ADDI	\$1 -1	2408		ADDI	\$1 1
2343		EXCH	\$7 \$1	2409		EXCH	\$10 \$1
2344		ADDI	\$1 -1	2410		ADDI	\$1 1
2345		EXCH	\$6 \$1	2411		EXCH	\$3 \$1
2346		ADDI	\$1 -1	2412		XORI	\$12 10
2347	obj_con_65:	ADDI	\$12 8	2413		EXCH	\$12 \$11
2348	<u></u>	EXCH	\$12 \$1	2414		ADDI	\$11 1
2349		ADDI	\$1 -1	2415		XORI	\$12 1
2350		EXCH	\$11 \$1	2416		EXCH	\$12 \$11
		ADDI			lobi con 66 hot.	ADDI	
2351			\$1 -1	2417	obj_con_66_bot:		\$11 -1
2352		BRA	l_malloc	2418		EXCH	\$11 \$10
2353		ADDI	\$1 1	2419		ADD	\$11 \$3
2354		EXCH	\$11 \$1	2420		ADDI	\$11 14
2355		ADDI	\$1 1	2421		EXCH	\$12 \$11
2356		EXCH	\$12 \$1	2422		ADDI	\$11 -14
2357	obj_con_65_i:	ADDI	\$12 -8	2423		SUB	\$11 \$3
2358		ADDI	\$1 1	2424		ADD	\$13 \$3
2359		EXCH	\$6 \$1	2425		ADDI	\$13 10
2360		ADDI	\$1 1	2426		EXCH	\$14 \$13
2361		EXCH	\$7 \$1	2427		ADDI	\$13 -10
2362		ADDI	\$1 1	2428		SUB	\$13 \$3
2363		EXCH	\$8 \$1	2429		ADD	\$12 \$14
2364		ADDI	\$1 1	2430		ADD	\$13 \$3
2365		EXCH	\$9 \$1	2431		ADDI	\$13 10
2366		ADDI	\$1 1	2432		EXCH	\$14 \$13
2367		EXCH	\$10 \$1	2433		ADDI	\$13 -10
2368		ADDI	\$1 1	2434		SUB	\$13 \$3
		EXCH				ADD	
2369			\$3 \$1	2435			\$11 \$3
2370		XORI	\$12 10	2436		ADDI	\$11 14
2371		EXCH	\$12 \$11	2437		EXCH	\$12 \$11
2372		ADDI	\$11 1	2438		ADDI	\$11 -14
2373		XORI	\$12 1	2439		SUB	\$11 \$3
2374		EXCH	\$12 \$11	2440		EXCH	\$11 \$6
2375	obj_con_65_bot:	ADDI	\$11 -1	2441		XOR	\$12 \$11
2376		EXCH	\$11 \$9	2442	loadMetAdd_67:	EXCH	\$13 \$12
2377		EXCH	\$3 \$1	2443		ADDI	\$13 3
2378		ADDI	\$1 -1	2444		EXCH	\$14 \$13
2379		EXCH	\$10 \$1	2445		XOR	\$15 \$14
2380		ADDI	\$1 -1	2446		EXCH	\$14 \$13
2381		EXCH	\$9 \$1	2447		ADDI	\$13 -3
2382		ADDI	\$1 -1	2448		EXCH	\$13 \$12
2383		EXCH	\$8 \$1	2449		EXCH	\$11 \$6
2384		ADDI	\$1 -1	2450		ADD	\$16 \$3
2385		EXCH	\$7 \$1	2451		ADDI	\$16 14
2386		ADDI	\$1 -1	2452		EXCH	\$3 \$1
2387		EXCH	\$6 \$1	2453		ADDI	\$1 -1
2388		ADDI	\$1 -1	2454		EXCH	\$10 \$1
2388	obj_con_66:	ADDI	\$12 8	2454		ADDI	\$10 \$1
1	05]_01_00.						
2390		EXCH	\$12 \$1 \$1 _1	2456		EXCH	\$9 \$1 \$1 _1
2391		ADDI	\$1 -1	2457		ADDI	\$1 -1
2392		EXCH	\$11 \$1	2458		EXCH	\$8 \$1
2393		ADDI	\$1 -1	2459		ADDI	\$1 -1
2394		BRA	l_malloc	2460		EXCH	\$7 \$1
2395		ADDI	\$1 1	2461		ADDI	\$1 -1
2396		EXCH	\$11 \$1	2462		EXCH	\$6 \$1
2397		ADDI	\$1 1	2463		ADDI	\$1 -1
2398		EXCH	\$12 \$1	2464		EXCH	\$16 \$1
2399	obj_con_66_i:	ADDI	\$12 -8	2465		ADDI	\$1 -1
2400		ADDI	\$1 1	2466		EXCH	\$12 \$1
2401		EXCH	\$6 \$1	2467		ADDI	\$1 -1
'							

					1		
2468		ADDI	\$15 -2468	2534		ADDI	\$1 -1
2469	1 rimp top 60.	RBRA	l_rjmp_bot_	78525		EXCH	\$7 \$1
2409	l_rjmp_top_69:			/ 2000			
2470	1_jmp_68:	SWAPBR	\$15	2536		ADDI	\$1 -1
2471		NEG	\$15	2537		EXCH	\$6 \$1
2472	l_rjmp_bot_70:	BRA	l_rjmp_top_	6 2 5 3 8		ADDI	\$1 -1
2473		ADDI	\$15 2468	2539		EXCH	\$16 \$1
1							
2474		ADDI	\$1 1	2540		ADDI	\$1 -1
2475		EXCH	\$12 \$1	2541		EXCH	\$12 \$1
2476		ADDI	\$1 1	2542		ADDI	\$1 -1
2477		EXCH	\$16 \$1	2543		ADDI	\$15 -2543
2411				2343			
2478		ADDI	\$1 1	2544	l_rjmp_top_73:	RBRA	l_rjmp_bot_74
2479		EXCH	\$6 \$1	2545		SWAPBR	
2419				2545	l_jmp_72:	SWAPER	
2480		ADDI	\$1 1	2546		NEG	\$15
		EXCH	\$7 \$1		l_rjmp_bot_74:	BRA	
2481			၃/ ၃⊥	2547	1_r_r Jmp_bor_/4:		l_rjmp_top_73
2482		ADDI	\$1 1	2548		ADDI	\$15 2543
0.400		EVCU	¢0 ¢1	05.40		ADDT	č1 1
2483		EXCH	\$8 \$1	2549		ADDI	\$1 1
2484		ADDI	\$1 1	2550		EXCH	\$12 \$1
2485		EXCH	\$9 \$1	2551		ADDI	\$1 1
2486		ADDI	\$1 1	2552		EXCH	\$16 \$1
1							
2487		EXCH	\$10 \$1	2553		ADDI	\$1 1
2488		ADDI	\$1 1	2554		EXCH	\$6 \$1
2489		EXCH	\$3 \$1	2555		ADDI	\$1 1
2490		ADDI	\$16 -14	2556		EXCH	\$7 \$1
1							
2491		SUB	\$16 \$3	2557		ADDI	\$1 1
2492		EXCH	\$11 \$6	2558		EXCH	\$8 \$1
2493		EXCH	\$13 \$12	2559		ADDI	\$1 1
2494		ADDI	\$13 3	2560		EXCH	\$9 \$1
2495		EXCH	\$14 \$13	2561		ADDI	\$1 1
2496		XOR	\$15 \$14	2562		EXCH	\$10 \$1
						ADDT	
2497		EXCH	\$14 \$13	2563		ADDI	\$1 1
2498		ADDI	\$13 -3	2564		EXCH	\$3 \$1
1	loadMetAdd_67_i:	EXCH				ADDI	\$16 -14
2499	TOAUMELAUU_07_1.	EACH	\$13 \$12	2565			910 -14
2500		XOR	\$12 \$11	2566		SUB	\$16 \$3
0501		EXCH					
2501			\$11 \$6	2567		EXCH	\$11 \$7
2502		ADD	\$11 \$3	2568		EXCH	\$13 \$12
2502		ADDI		2560		ADDI	\$13 3
2503			\$11 14	2569		ADDI	713 3
2504		EXCH	\$12 \$11	2570		EXCH	\$14 \$13
		ADDI					
2505		ADDI	\$11 -14	2571		XOR	\$15 \$14
2506		SUB	\$11 \$3	2572		EXCH	\$14 \$13
2507		XORI	\$13 1	2573		ADDI	\$13 -3
2307			ATO T	2313		ADDI	913 -S
2508		ADD	\$12 \$13	2574	loadMetAdd_71_i:	EXCH	\$13 \$12
2509		XORI	\$13 1	2575		XOR	\$12 \$11
2510		ADD	\$11 \$3	2576		EXCH	\$11 \$7
2511		ADDI	\$11 14	2577		ADD	\$11 \$3
2512		EXCH	\$12 \$11	2578		ADDI	\$11 14
2513		ADDI	\$11 -14	2579		EXCH	\$12 \$11
1							
2514		SUB	\$11 \$3	2580		ADDI	\$11 -14
2515		EXCH	\$11 \$7	2581		SUB	\$11 \$3
2516		XOR	\$12 \$11	2582		XORI	\$13 1
2517	loadMetAdd_71:	EXCH	\$13 \$12	2583		ADD	\$12 \$13
1							
2518		ADDI	\$13 3	2584		XORI	\$13 1
2519		EXCH	\$14 \$13	2585		ADD	\$11 \$3
2520		XOR	\$15 \$14	2586		ADDI	\$11 14
2521		EXCH	\$14 \$13	2587		EXCH	\$12 \$11
2522		ADDI	\$13 -3	2588		ADDI	\$11 -14
2523		EXCH	\$13 \$12	2589		SUB	\$11 \$3
		EXCH	\$11 \$7			EXCH	
2524				2590			\$11 \$8
2525		ADD	\$16 \$3	2591		XOR	\$12 \$11
2526		ADDI	\$16 14	2592	loadMetAdd_75:	EXCH	\$13 \$12
2527		EXCH	\$3 \$1	2593		ADDI	\$13 3
2528		ADDI	\$1 -1	2594		EXCH	\$14 \$13
2529		EXCH	\$10 \$1	2595		XOR	\$15 \$14
2530		ADDI	\$1 -1	2596		EXCH	\$14 \$13
1							
2531		EXCH	\$9 \$1	2597		ADDI	\$13 -3
2532		ADDI	\$1 -1	2598		EXCH	\$13 \$12
2533		EXCH	\$8 \$1	2599		EXCH	\$11 \$8
,							

0000		ADD	\$16 \$3	0000		XOR	¢10 ¢11
2600				2666	loadMotAdd 70.		\$12 \$11
2601		ADDI	\$16 14	2667	loadMetAdd_79:	EXCH	\$13 \$12
2602		EXCH	\$3 \$1	2668		ADDI	\$13 3
2603		ADDI	\$1 -1	2669		EXCH	\$14 \$13
2604		EXCH	\$10 \$1	2670		XOR	\$15 \$14
2605		ADDI	\$1 -1	2671		EXCH	\$14 \$13
2606		EXCH	\$9 \$1	2672		ADDI	\$13 -3
2607		ADDI	\$1 -1	2673		EXCH	\$13 \$12
2608		EXCH	\$8 \$1	2674		EXCH	\$11 \$10
2609		ADDI	\$1 -1	2675		ADD	\$16 \$3
2610		EXCH	\$7 \$1	2676		ADDI	\$16 14
2611		ADDI	\$1 -1	2677		EXCH	\$3 \$1
2612		EXCH	\$6 \$1	2678		ADDI	\$1 -1
2613		ADDI	\$1 -1	2679		EXCH	\$10 \$1
2614		EXCH	\$16 \$1	2680		ADDI	\$1 -1
2615		ADDI	\$1 -1	2681		EXCH	\$9 \$1
2616		EXCH	\$12 \$1	2682		ADDI	\$1 -1
2617		ADDI	\$1 -1	2683		EXCH	\$8 \$1
2618		ADDI	\$15 -2618	2684		ADDI	\$1 -1
2619	l_rjmp_top_77:	RBRA	l_rjmp_bot_7			EXCH	\$7 \$1
2620	1_jmp_76:	SWAPBR		2686		ADDI	\$1 -1
2621		NEG	\$15			EXCH	\$6 \$1
	1 mimm bot 70.			2687		ADDI	\$1 -1
2622	l_rjmp_bot_78:	BRA	l_rjmp_top_7				
2623		ADDI	\$15 2618	2689		EXCH	\$16 \$1
2624		ADDI	\$1 1	2690		ADDI	\$1 -1
2625		EXCH	\$12 \$1	2691		EXCH	\$12 \$1
2626		ADDI	\$1 1	2692		ADDI	\$1 -1
2627		EXCH	\$16 \$1	2693		ADDI	\$15 -2693
2628		ADDI	\$1 1	2694	l_rjmp_top_81:	RBRA	l_rjmp_bot_82
2629		EXCH	\$6 \$1	2695	l_jmp_80:	SWAPBR	\$15
2630		ADDI	\$1 1	2696		NEG	\$15
2631		EXCH	\$7 \$1	2697	l_rjmp_bot_82:	BRA	l_rjmp_top_81
2632		ADDI	\$1 1	2698		ADDI	\$15 2693
2633		EXCH	\$8 \$1	2699		ADDI	\$1 1
2634		ADDI	\$1 1	2700		EXCH	\$12 \$1
2635		EXCH	\$9 \$1	2701		ADDI	\$1 1
2636		ADDI	\$1 1	2702		EXCH	\$16 \$1
2637		EXCH	\$10 \$1	2703		ADDI	\$1 1
2638		ADDI	\$1 1	2704		EXCH	\$6 \$1
2639		EXCH	\$3 \$1	2705		ADDI	\$1 1
2640		ADDI	\$16 -14	2706		EXCH	\$7 \$1
2641		SUB	\$16 \$3	2707		ADDI	\$1 1
2642		EXCH	\$11 \$8	2708		EXCH	\$8 \$1
2643		EXCH	\$13 \$12	2709		ADDI	\$1 1
2644		ADDI	\$13 3	2710		EXCH	\$9 \$1
2645		EXCH	\$14 \$13	2711		ADDI	\$1 1
2646		XOR	\$15 \$14	2711		EXCH	\$10 \$1
						ADDI	\$1 1
2647		EXCH	\$14 \$13	2713			
2648	loadMa+Add 75 :-	ADDI	\$13 -3	2714		EXCH	\$3 \$1
2649	loadMetAdd_75_i:	EXCH	\$13 \$12	2715		ADDI	\$16 -14
2650		XOR	\$12 \$11	2716		SUB	\$16 \$3
2651		EXCH	\$11 \$8	2717		EXCH	\$11 \$10
2652		ADD	\$11 \$3	2718		EXCH	\$13 \$12
2653		ADDI	\$11 14	2719		ADDI	\$13 3
2654		EXCH	\$12 \$11	2720		EXCH	\$14 \$13
2655		ADDI	\$11 -14	2721		XOR	\$15 \$14
2656		SUB	\$11 \$3	2722		EXCH	\$14 \$13
2657		XORI	\$13 1	2723		ADDI	\$13 -3
2658		ADD	\$12 \$13	2724	loadMetAdd_79_i:	EXCH	\$13 \$12
2659		XORI	\$13 1	2725		XOR	\$12 \$11
2660		ADD	\$11 \$3	2726		EXCH	\$11 \$10
2661		ADDI	\$11 14	2727		ADD	\$11 \$3
2662		EXCH	\$12 \$11	2728		ADDI	\$11 2
2663		ADDI	\$11 -14	2729		EXCH	\$12 \$11
2664		SUB	\$11 \$3	2730		ADDI	\$11 -2
2665		EXCH	\$11 \$10	2731		SUB	\$11 \$3

		<b></b>	<b>610</b>	÷.c	a=aa1			61 1
2732		EXCH	\$13		2798		ADDI	\$1 1
2733	swap_83:	XOR	\$12	\$13	2799		EXCH	\$13 \$1
2734		XOR	\$13	\$12	2800		ADDI	\$1 1
2735		XOR	\$12		2801		EXCH	\$6 \$1
2736		EXCH	\$13		2802		ADDI	\$1 1
2737		ADD	\$11		2803		EXCH	\$7 \$1
2738		ADDI	\$11	2	2804		ADDI	\$1 1
2739		EXCH	\$12	\$11	2805		EXCH	\$8 \$1
2740		ADDI	\$11		2806		ADDI	\$1 1
		SUB	\$11				EXCH	
2741					2807			\$9 \$1
2742		EXCH	\$11		2808		ADDI	\$1 1
2743		ADD	\$12	\$3	2809		EXCH	\$10 \$1
2744		ADDI	\$12	2	2810		ADDI	\$1 1
2745		EXCH	\$13		2811		EXCH	\$3 \$1
1								
2746		ADDI	\$12		2812		ADD	\$11 \$3
2747		SUB	\$12	\$3	2813		ADDI	\$11 2
2748	copy_84:	XOR	\$11	\$13	2814		EXCH	\$12 \$11
2749		ADDI	\$13	1	2815		ADDI	\$11 -2
2750		EXCH	\$14		2816		SUB	\$11 \$3
1								
2751		ADDI	\$14		2817		EXCH	\$14 \$13
2752		EXCH	\$14	\$13	2818		ADDI	\$14 2
2753		ADDI	\$13	-1	2819		EXCH	\$15 \$14
2754		ADD	\$12	\$3	2820		XOR	\$16 \$15
		ADDI	\$12				EXCH	
2755					2821			\$15 \$14
2756		EXCH	\$13		2822		ADDI	\$14 -2
2757		ADDI	\$12	-2	2823	loadMetAdd_85_i:	EXCH	\$14 \$13
2758		SUB	\$12	\$3	2824		XOR	\$13 \$12
2759		EXCH	\$11		2825		ADD	\$11 \$3
							ADDI	
2760		ADD	\$11		2826			\$11 2
2761		ADDI	\$11	2	2827		EXCH	\$12 \$11
2762		EXCH	\$12	\$11	2828		ADDI	\$11 -2
2763		ADDI	\$11	-2	2829		SUB	\$11 \$3
2764		SUB	\$11		2830		EXCH	\$11 \$6
2765		XOR	\$13		2831		EXCH	\$12 \$7
2766	loadMetAdd_85:	EXCH	\$14	\$13	2832	copy_89:	XOR	\$11 \$12
2767		ADDI	\$14	2	2833		ADDI	\$12 1
2768		EXCH	\$15	\$14	2834		EXCH	\$13 \$12
2769		XOR	\$16		2835		ADDI	\$13 1
1								
2770		EXCH	\$15		2836		EXCH	\$13 \$12
2771		ADDI	\$14	-2	2837		ADDI	\$12 -1
2772		EXCH	\$14	\$13	2838		EXCH	\$12 \$7
2773		ADD	\$11	\$3	2839		EXCH	\$11 \$6
2774		ADDI	\$11		2840		EXCH	\$11 \$7
1								
2775		EXCH	\$12		2841		XOR	\$12 \$11
2776		ADDI	\$11	-2	2842	loadMetAdd_90:	EXCH	\$13 \$12
2777		SUB	\$11	\$3	2843		ADDI	\$13 2
2778		EXCH	\$3 \$	\$1	2844		EXCH	\$14 \$13
2779		ADDI	\$1 -		2845		XOR	\$15 \$14
		EXCH	\$10				EXCH	\$14 \$13
2780					2846			
2781		ADDI	\$1 -		2847		ADDI	\$13 -2
2782		EXCH	\$9 \$	51	2848		EXCH	\$13 \$12
2783		ADDI	\$1 -	-1	2849		EXCH	\$11 \$7
2784		EXCH	\$8 \$	\$1	2850		EXCH	\$3 \$1
2785		ADDI	\$1 -		2851		ADDI	\$1 -1
1								
2786		EXCH	\$7 \$		2852		EXCH	\$10 \$1
2787		ADDI	\$1 -		2853		ADDI	\$1 -1
2788		EXCH	\$6 \$	\$1	2854		EXCH	\$9 \$1
2789		ADDI	\$1 -	-1	2855		ADDI	\$1 -1
2790		EXCH	\$13		2856		EXCH	\$8 \$1
1								
2791		ADDI	\$1 -		2857		ADDI	\$1 -1
2792		ADDI	\$16	-2792	2858		EXCH	\$7 \$1
2793	l_rjmp_top_87:	RBRA	1_r	jmp_bot_8	8859		ADDI	\$1 -1
2794	l_jmp_86:	SWAPBR	\$16		2860		EXCH	\$6 \$1
2795	- <del>-</del>	NEG	\$16		2861		ADDI	\$1 -1
	1 rimp bot 00.			imp + 0				
2796	l_rjmp_bot_88:	BRA		jmp_top_8			EXCH	\$12 \$1
2797		ADDI	\$16	2792	2863		ADDI	\$1 -1

2864		ADDI	\$15 -2864	2930	1_jmp_96:	SWAPBR	\$15
2865	l_rjmp_top_92:	RBRA	l_rjmp_bot_		_3 1_***	NEG	\$15
2866	l_jmp_91:	SWAPBR		2932	l_rjmp_bot_98:	BRA	l_rjmp_top_97
2867	=3 1=	NEG	\$15	2933	_ 3 1	ADDI	\$15 2928
2868	l_rjmp_bot_93:	BRA	l_rjmp_top_	92934		ADDI	\$1 1
2869	_ 3 1	ADDI	\$15 2864	2935		EXCH	\$12 \$1
2870		ADDI	\$1 1	2936		ADDI	\$1 1
2871		EXCH	\$12 \$1	2937		EXCH	\$6 \$1
2872		ADDI	\$1 1	2938		ADDI	\$1 1
2873		EXCH	\$6 \$1	2939		EXCH	\$7 \$1
2874		ADDI	\$1 1	2940		ADDI	\$1 1
2875		EXCH	\$7 \$1	2941		EXCH	\$8 \$1
2876		ADDI	\$1 1	2942		ADDI	\$1 1
2877		EXCH	\$8 \$1	2943		EXCH	\$9 \$1
2878		ADDI	\$1 1	2944		ADDI	\$1 1
2879		EXCH	\$9 \$1	2945		EXCH	\$10 \$1
2880		ADDI	\$1 1	2946		ADDI	\$1 1
2881		EXCH	\$10 \$1	2947		EXCH	\$3 \$1
2882		ADDI	\$1 1	2948		EXCH	\$11 \$8
2883		EXCH	\$3 \$1	2949		EXCH	\$13 \$12
2884		EXCH	\$11 \$7	2950		ADDI	\$13 2
2885		EXCH	\$13 \$12	2951		EXCH	\$14 \$13
2886		ADDI	\$13 2	2952		XOR	\$15 \$14
2887		EXCH	\$14 \$13	2953		EXCH	\$14 \$13
2888		XOR	\$15 \$14	2954		ADDI	\$13 -2
2889		EXCH	\$14 \$13	2955	loadMetAdd_95_i:	EXCH	\$13 \$12
2890		ADDI	\$13 -2	2956	TOAUMECAUU_JJ_I.	XOR	\$12 \$11
2891	loadMetAdd_90_i:	EXCH	\$13 \$12	2957		EXCH	\$11 \$8
2892	10admetAdd_90_1.	XOR				EXCH	
2892		EXCH	\$12 \$11 \$11 \$7	2958		EXCH	\$11 \$6
1				2959	gopy 99.		\$12 \$9
2894		EXCH	\$11 \$6	2960	сору_99:	XOR	\$11 \$12
2895		EXCH	\$12 \$8	2961		ADDI	\$12 1
2896	copy_94:	XOR	\$11 \$12	2962		EXCH	\$13 \$12
2897		ADDI	\$12 1	2963		ADDI	\$13 1
2898		EXCH	\$13 \$12	2964		EXCH	\$13 \$12
2899		ADDI	\$13 1	2965		ADDI	\$12 -1
2900		EXCH	\$13 \$12	2966		EXCH	\$12 \$9
2901		ADDI	\$12 -1	2967		EXCH	\$11 \$6
2902		EXCH	\$12 \$8	2968		EXCH	\$11 \$9
2903		EXCH	\$11 \$6	2969	7 77 17 100	XOR	\$12 \$11
2904		EXCH	\$11 \$8	2970	loadMetAdd_100:	EXCH	\$13 \$12
2905		XOR	\$12 \$11	2971		ADDI	\$13 2
2906	loadMetAdd_95:	EXCH	\$13 \$12	2972		EXCH	\$14 \$13
2907		ADDI	\$13 2	2973		XOR	\$15 \$14
2908		EXCH	\$14 \$13	2974		EXCH	\$14 \$13
2909		XOR	\$15 \$14	2975		ADDI	\$13 -2
2910		EXCH	\$14 \$13	2976		EXCH	\$13 \$12
2911		ADDI	\$13 -2	2977		EXCH	\$11 \$9
2912		EXCH	\$13 \$12	2978		EXCH	\$3 \$1
2913		EXCH	\$11 \$8	2979		ADDI	\$1 -1
2914		EXCH	\$3 \$1	2980		EXCH	\$10 \$1
2915		ADDI	\$1 -1	2981		ADDI	\$1 -1
2916		EXCH	\$10 \$1	2982		EXCH	\$9 \$1
2917		ADDI	\$1 -1	2983		ADDI	\$1 -1
2918		EXCH	\$9 \$1	2984		EXCH	\$8 \$1
2919		ADDI	\$1 -1	2985		ADDI	\$1 -1
2920		EXCH	\$8 \$1	2986		EXCH	\$7 \$1
2921		ADDI	\$1 -1	2987		ADDI	\$1 -1
2922		EXCH	\$7 \$1	2988		EXCH	\$6 \$1
2923		ADDI	\$1 -1	2989		ADDI	\$1 -1
2924		EXCH	\$6 \$1	2990		EXCH	\$12 \$1
2925		ADDI	\$1 -1	2991		ADDI	\$1 -1
2926		EXCH	\$12 \$1	2992		ADDI	\$15 -2992
2927		ADDI	\$1 -1	2993		RBRA	l_rjmp_bot_103
2928		ADDI	\$15 -2928	2994	l_jmp_101:	SWAPBR	
2929	l_rjmp_top_97:	RBRA	l_rjmp_bot_	9 2995		NEG	\$15

2996	l rimp bot 103.	BRA	l_rjmp_top_1	1 മഹദാ		ADDI	\$1 1
2996	l_rjmp_bot_103:	ADDI	\$15 2992	3063		EXCH	\$12 \$1
2998		ADDI	\$1 1	3064		ADDI	\$1 1
2999		EXCH	\$12 \$1	3065		EXCH	\$6 \$1
3000		ADDI	\$1 1	3066		ADDI	\$1 1
3001		EXCH	\$6 \$1	3067		EXCH	\$7 \$1
3001		ADDI	\$1 1	3068		ADDI	\$1 1
3002		EXCH	\$7 \$1	3069		EXCH	\$8 \$1
3003		ADDI	\$1 1	3070		ADDI	\$1 1
3004		EXCH	\$8 \$1	3071		EXCH	\$9 \$1
3006		ADDI	\$1 1	3072		ADDI	\$1 1
3007		EXCH	\$9 \$1	3073		EXCH	\$10 \$1
3007		ADDI	\$1 1	3074		ADDI	\$1 1
3009		EXCH	\$10 \$1	3074		EXCH	\$3 \$1
3010		ADDI	\$1 1	3076		EXCH	\$11 \$10
3011		EXCH	\$3 \$1	3077		EXCH	\$13 \$12
3011		EXCH	\$11 \$9	3078		ADDI	\$13 2
3012		EXCH	\$13 \$12	3079		EXCH	\$14 \$13
3013		ADDI	\$13 2	3080		XOR	\$15 \$14
3014		EXCH	\$14 \$13	3081		EXCH	\$14 \$13
3016		XOR	\$15 \$14	3082		ADDI	\$13 -2
3017		EXCH	\$14 \$13	3083	loadMetAdd_105_i:	EXCH	\$13 \$12
3017		ADDI	\$13 -2	3084	TOAUMECAUU_TUU_T.	XOR	\$12 \$11
3019	loadMetAdd_100_i:	EXCH	\$13 \$12	3085		EXCH	\$11 \$10
3020	TOAUNECAUU_TOO_T:	XOR	\$12 \$11	3086		EXCH	\$11 \$6
3020		EXCH	\$12 \$11 \$11	3086		EXCH	\$12 \$9
3021		EXCH			copy_109:	XOR	
			\$11 \$6	3088	copy_109:		\$11 \$12
3023	104.	EXCH	\$12 \$10	3089		ADDI	\$12 1
3024	copy_104:	XOR	\$11 \$12	3090		EXCH	\$13 \$12
3025		ADDI EXCH	\$12 1	3091		ADDI EXCH	\$13 1 \$13 \$12
3026		ADDI	\$13 \$12	3092		ADDI	
3027			\$13 1	3093			\$12 -1
3028		EXCH	\$13 \$12	3094		EXCH	\$12 \$9
3029		ADDI	\$12 -1	3095		EXCH	\$11 \$6
3030		EXCH	\$12 \$10	3096		EXCH	\$11 \$10
3031		EXCH	\$11 \$6	3097	1	XOR	\$12 \$11
3032		EXCH XOR	\$11 \$10	3098	loadMetAdd_110:	EXCH ADDI	\$13 \$12
3033	loadMetAdd_105:	EXCH	\$12 \$11 \$13 \$12	3099 3100		EXCH	\$13 0
3034 3035	TOAUMECAGG_TOS.	ADDI				XOR	\$14 \$13
			\$13 2	3101			\$15 \$14
3036		EXCH XOR	\$14 \$13	3102 3103		EXCH ADDI	\$14 \$13 \$13 0
3037 3038		EXCH	\$15 \$14			EXCH	
3038		ADDI	\$14 \$13	3104 3105		EXCH	\$13 \$12
			\$13 -2				\$11 \$10 \$3 \$1
3040		EXCH EXCH	\$13 \$12	3106		EXCH	
3041 3042		EXCH	\$11 \$10 \$3 \$1	3107 3108		ADDI EXCH	\$1 -1 \$10 \$1
3043		ADDI	\$1 -1	3109		ADDI	\$1 -1
3043		EXCH	\$10 \$1	3110		EXCH	\$9 \$1
3044		ADDI	\$10 \$1	3110		ADDI	\$1 -1
3045		EXCH	\$9 \$1	3111		EXCH	\$8 \$1
3046		ADDI	\$1 -1	3112		ADDI	\$1 -1
3047		EXCH	\$8 \$1	3114		EXCH	\$7 \$1
3048		ADDI	\$1 -1	3114		ADDI	\$1 -1
3050		EXCH		3116		EXCH	\$6 \$1
3050		ADDI	\$7 \$1 \$1 -1	3116		ADDI	\$1 -1
3051		EXCH	\$6 \$1	3117		EXCH	\$12 \$1
3052		ADDI	\$1 -1	3118		ADDI	\$1 -1
3054		EXCH	\$12 \$1	3119		ADDI	\$15 -3120
3054		ADDI	\$12 \$1 \$1 -1		l rimp top 112.	RBRA	
					l_rjmp_top_112:		l_rjmp_bot_113
3056	l rimp top 107.	ADDI RBRA	\$15 -3056	3122	l_jmp_111:	SWAPBR	\$15 \$15
3057	l_rjmp_top_107:		l_rjmp_bot_1		1 rimp bot 112.	NEG BBA	
3058	l_jmp_106:	SWAPBR		3124	l_rjmp_bot_113:	BRA	l_rjmp_top_112
3059	l rimp bot 100.	NEG BBA	\$15	3125		ADDI	\$15 3120
3060	l_rjmp_bot_108:	BRA	l_rjmp_top_1			ADDI	\$1 1
3061		ADDI	\$15 3056	3127		EXCH	\$12 \$1

1			** *				***
3128		ADDI	\$1 1	3194		EXCH	\$11 \$9
3129		EXCH	\$6 \$1	3195		EXCH	\$13 \$12
3130		ADDI	\$1 1	3196		ADDI	\$13 1
3131		EXCH	\$7 \$1	3197		EXCH	\$14 \$13
3132		ADDI	\$1 1	3198		XOR	\$15 \$14
		EXCH				EXCH	\$14 \$13
3133			\$8 \$1	3199			
3134		ADDI	\$1 1	3200		ADDI	\$13 -1
3135		EXCH	\$9 \$1	3201	loadMetAdd_114_i:	EXCH	\$13 \$12
3136		ADDI	\$1 1	3202		XOR	\$12 \$11
3137		EXCH	\$10 \$1	3203		EXCH	\$11 \$9
3138		ADDI	\$1 1	3204		EXCH	\$11 \$6
3139		EXCH	\$3 \$1	3205		EXCH	\$12 \$8
					110		
3140		EXCH	\$11 \$10	3206	copy_118:	XOR	\$11 \$12
3141		EXCH	\$13 \$12	3207		ADDI	\$12 1
3142		ADDI	\$13 0	3208		EXCH	\$13 \$12
3143		EXCH	\$14 \$13	3209		ADDI	\$13 1
3144		XOR	\$15 \$14	3210		EXCH	\$13 \$12
3145		EXCH		3211		ADDI	\$12 -1
			\$14 \$13				
3146		ADDI	\$13 0	3212		EXCH	\$12 \$8
3147	loadMetAdd_110_i:	EXCH	\$13 \$12	3213		EXCH	\$11 \$6
3148		XOR	\$12 \$11	3214		EXCH	\$11 \$9
3149		EXCH	\$11 \$10	3215		XOR	\$12 \$11
3150		EXCH	\$11 \$9	3216	loadMetAdd_119:	EXCH	\$13 \$12
3151		XOR	\$12 \$11	3217	iodane chaa_iij.	ADDI	\$13 0
	3 26 1 3 1 1 1 1 1						
3152	loadMetAdd_114:	EXCH	\$13 \$12	3218		EXCH	\$14 \$13
3153		ADDI	\$13 1	3219		XOR	\$15 \$14
3154		EXCH	\$14 \$13	3220		EXCH	\$14 \$13
3155		XOR	\$15 \$14	3221		ADDI	\$13 0
3156		EXCH	\$14 \$13	3222		EXCH	\$13 \$12
3157		ADDI	\$13 -1	3223		EXCH	\$11 \$9
3158		EXCH	\$13 \$12	3224		EXCH	\$3 \$1
3159		EXCH	\$11 \$9	3225		ADDI	\$1 -1
3160		EXCH	\$3 \$1	3226		EXCH	\$10 \$1
3161		ADDI	\$1 -1	3227		ADDI	\$1 -1
3162		EXCH	\$9 \$1	3228		EXCH	\$9 \$1
3163		ADDI	\$1 -1	3229		ADDI	\$1 -1
3164		EXCH	\$8 \$1	3230		EXCH	\$8 \$1
3165		ADDI	\$1 -1	3231		ADDI	\$1 -1
3166		EXCH	\$7 \$1	3232		EXCH	\$7 \$1
3167		ADDI	\$1 -1	3233		ADDI	\$1 -1
3168		EXCH	\$6 \$1	3234		EXCH	\$6 \$1
3169		ADDI	\$1 -1	3235		ADDI	\$1 -1
3170		EXCH	\$10 \$1	3236		EXCH	\$12 \$1
3171		ADDI	\$1 -1	3237		ADDI	\$1 -1
3172		EXCH	\$12 \$1	3238		ADDI	\$15 -3238
3173		ADDI	\$1 -1	3239	l_rjmp_top_121:	RBRA	l_rjmp_bot_122
3174		ADDI	\$15 -3174	3240	l_jmp_120:	SWAPBR	\$15
	l_rjmp_top_116:	RBRA	l_rjmp_bot_1		- <del></del>	NEG	\$15
3176	l_jmp_115:	SWAPBR		3242	l_rjmp_bot_122:	BRA	l_rjmp_top_121
	Jb+				Jmp_xvc_122.		
3177	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	NEG	\$15	3243		ADDI	\$15 3238
3178	l_rjmp_bot_117:	BRA	l_rjmp_top_1			ADDI	\$1 1
3179		ADDI	\$15 3174	3245		EXCH	\$12 \$1
3180		ADDI	\$1 1	3246		ADDI	\$1 1
3181		EXCH	\$12 \$1	3247		EXCH	\$6 \$1
3182		ADDI	\$1 1	3248		ADDI	\$1 1
3183		EXCH	\$10 \$1	3249		EXCH	\$7 \$1
3184		ADDI	\$1 1	3250		ADDI	\$1 1
3185		EXCH	\$6 \$1	3251		EXCH	\$8 \$1
3186		ADDI	\$1 1	3252		ADDI	\$1 1
3187		EXCH	\$7 \$1	3253		EXCH	\$9 \$1
3188		ADDI	\$1 1	3254		ADDI	\$1 1
3189		EXCH	\$8 \$1	3255		EXCH	\$10 \$1
		ADDI				ADDI	\$1 1
3190			\$1 1	3256			
3191		EXCH	\$9 \$1	3257		EXCH	\$3 \$1
3192		ADDI	\$1 1	3258		EXCH	\$11 \$9
3193		EXCH	\$3 \$1	3259		EXCH	\$13 \$12

			ć12 O			<b></b>	610 610
3260		ADDI	\$13 0	3326		EXCH	\$13 \$12
3261		EXCH	\$14 \$13	3327		ADDI	\$13 1
3262		XOR	\$15 \$14	3328		EXCH	\$13 \$12
3263		EXCH	\$14 \$13	3329		ADDI	\$12 -1
3264		ADDI	\$13 0	3330		EXCH	\$12 \$7
3265	loadMetAdd_119_i:	EXCH	\$13 \$12	3331		EXCH	\$11 \$6
3266		XOR	\$12 \$11	3332		EXCH	\$11 \$8
3267		EXCH	\$11 \$9	3333		XOR	\$12 \$11
3268		EXCH	\$11 \$8	3334	loadMetAdd_128:	EXCH	\$13 \$12
3269		XOR	\$12 \$11	3335		ADDI	\$13 0
3270	loadMetAdd_123:	EXCH	\$13 \$12	3336		EXCH	\$14 \$13
3271		ADDI	\$13 1	3337		XOR	\$15 \$14
3272		EXCH	\$14 \$13	3338		EXCH	\$14 \$13
3273		XOR	\$15 \$14	3339		ADDI	\$13 0
3274		EXCH	\$14 \$13	3340		EXCH	\$13 \$12
3275		ADDI	\$13 -1	3341		EXCH	\$11 \$8
3276		EXCH	\$13 \$12	3342		EXCH	\$3 \$1
3277		EXCH	\$11 \$8	3343		ADDI	\$1 -1
3278		EXCH	\$3 \$1	3344		EXCH	\$10 \$1
3279		ADDI	\$1 -1	3345		ADDI	\$1 -1
3280		EXCH	\$10 \$1	3346		EXCH	\$9 \$1
3281		ADDI	\$1 -1	3347		ADDI	\$1 -1
3282		EXCH	\$8 \$1	3348		EXCH	\$8 \$1
3283		ADDI	\$1 -1	3349		ADDI	\$1 -1
3284		EXCH	\$7 \$1	3350		EXCH	\$7 \$1
3285		ADDI	\$1 -1	3351		ADDI	\$1 -1
3286		EXCH	\$6 \$1	3352		EXCH	\$6 \$1
		ADDI					
3287			\$1 -1	3353		ADDI	\$1 -1
3288		EXCH	\$9 \$1	3354		EXCH	\$12 \$1
3289		ADDI	\$1 -1	3355		ADDI	\$1 -1
3290		EXCH	\$12 \$1	3356		ADDI	\$15 -3356
3291		ADDI	\$1 -1	3357	l_rjmp_top_130:	RBRA	l_rjmp_bot_131
3292		ADDI	\$15 -3292	3358	l_jmp_129:	SWAPBR	\$15
3293	l_rjmp_top_125:	RBRA	l_rjmp_bot_1	L <b>23</b> 559		NEG	\$15
3294	l_jmp_124:	SWAPBR	\$15	3360	l_rjmp_bot_131:	BRA	l_rjmp_top_130
3295		NEG	\$15	3361		ADDI	\$15 3356
3296	l_rjmp_bot_126:	BRA	l_rjmp_top_1	L 23562		ADDI	\$1 1
3297	_ 3 1_***_ **	ADDI	\$15 3292	3363		EXCH	\$12 \$1
3298		ADDI	\$1 1	3364		ADDI	\$1 1
3299		EXCH	\$12 \$1	3365		EXCH	\$6 \$1
3300		ADDI	\$1 1	3366		ADDI	\$1 1
3301		EXCH	\$9 \$1	3367		EXCH	\$7 \$1
3302		ADDI	\$1 1	3368		ADDI	\$1 1
3303		EXCH	\$6 \$1	3369		EXCH	\$8 \$1
3304		ADDI	\$1 1	3370		ADDI	\$1 1
3305		EXCH	\$7 \$1	3371		EXCH	\$9 \$1
3306		ADDI	\$1 1	3372		ADDI	\$1 1
3307		EXCH	\$8 \$1	3373		EXCH	\$10 \$1
3308		ADDI	\$1 1	3374		ADDI	\$1 1
3309		EXCH	\$10 \$1	3375		EXCH	\$3 \$1
3310		ADDI	\$1 1	3376		EXCH	\$11 \$8
3311		EXCH	\$3 \$1	3377		EXCH	\$13 \$12
3312		EXCH	\$11 \$8	3378		ADDI	\$13 0
3313		EXCH	\$13 \$12	3379		EXCH	\$14 \$13
3314		ADDI	\$13 1	3380		XOR	\$15 \$14
3315		EXCH	\$14 \$13	3381		EXCH	\$14 \$13
3316		XOR	\$15 \$14	3382	1	ADDI	\$13 0
3317		EXCH	\$14 \$13	3383	loadMetAdd_128_i:	EXCH	\$13 \$12
3318	1 22 22 200	ADDI	\$13 -1	3384		XOR	\$12 \$11
3319	loadMetAdd_123_i:	EXCH	\$13 \$12	3385		EXCH	\$11 \$8
3320		XOR	\$12 \$11	3386		EXCH	\$11 \$7
3321		EXCH	\$11 \$8	3387		XOR	\$12 \$11
3322		EXCH	\$11 \$6	3388	loadMetAdd_132:	EXCH	\$13 \$12
3323		EXCH	\$12 \$7	3389		ADDI	\$13 1
3324	copy_127:	XOR	\$11 \$12	3390		EXCH	\$14 \$13
3325		ADDI	\$12 1	3391		XOR	\$15 \$14
ı							

3392		EXCH	\$14 \$13	3458		EXCH	\$11 \$7
3393		ADDI	\$13 -1	3459		XOR	\$12 \$11
3394		EXCH	\$13 \$12	3460	loadMetAdd_137:	EXCH	\$13 \$12
3395		EXCH	\$11 \$7	3461		ADDI	\$13 0
3396		EXCH	\$3 \$1	3462		EXCH	\$14 \$13
3397		ADDI	\$1 -1	3463		XOR	\$15 \$14
3398		EXCH	\$10 \$1	3464		EXCH	\$14 \$13
3399		ADDI	\$1 -1	3465		ADDI	\$13 0
3400		EXCH	\$9 \$1	3466		EXCH	\$13 \$12
3401		ADDI	\$1 -1	3467		EXCH	\$11 \$7
3402		EXCH	\$7 \$1	3468		EXCH	\$3 \$1
3403		ADDI	\$1 -1	3469		ADDI	\$1 -1
3404		EXCH	\$6 \$1	3470		EXCH	\$10 \$1
3405		ADDI	\$1 -1	3471		ADDI	\$1 -1
3406		EXCH	\$8 \$1	3472		EXCH	\$9 \$1
3407		ADDI	\$1 -1	3473		ADDI	\$1 -1
3408		EXCH	\$12 \$1	3474		EXCH	\$8 \$1
3409		ADDI	\$1 -1	3475		ADDI	\$1 -1
3410		ADDI	\$15 -3410	3476		EXCH	\$7 \$1
3411	l_rjmp_top_134:	RBRA	l_rjmp_bot_1			ADDI	\$1 -1
3412	l_jmp_133:	SWAPBR		3478		EXCH	\$6 \$1
3413	<u></u>	NEG	\$15	3479		ADDI	\$1 -1
3414	l_rjmp_bot_135:	BRA	l_rjmp_top_1			EXCH	\$12 \$1
3415	<u></u>	ADDI	\$15 3410	3481		ADDI	\$1 -1
3416		ADDI	\$1 1	3482		ADDI	\$15 -3482
3417		EXCH	\$12 \$1	3483	l_rjmp_top_139:	RBRA	l_rjmp_bot_140
3418		ADDI	\$1 1	3484		SWAPBR	
3419		EXCH	\$8 \$1	3485	1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 - 1 -	NEG	\$15
3420		ADDI	\$1 1	3486	l_rjmp_bot_140:	BRA	1_rjmp_top_139
3421		EXCH	\$6 \$1	3487	1_1_1	ADDI	\$15 3482
3422		ADDI	\$1 1	3488		ADDI	\$1 1
3423		EXCH	\$7 \$1	3489		EXCH	\$12 \$1
3424		ADDI	\$1 1	3490		ADDI	\$1 1
3425		EXCH	\$9 \$1	3491		EXCH	\$6 \$1
3426		ADDI	\$1 1	3492		ADDI	\$1 1
3427		EXCH	\$10 \$1	3493		EXCH	\$7 \$1
3428		ADDI	\$1 1	3494		ADDI	\$1 1
3429		EXCH	\$3 \$1	3495		EXCH	\$8 \$1
3430		EXCH	\$11 \$7	3496		ADDI	\$1 1
3431		EXCH	\$13 \$12	3497		EXCH	\$9 \$1
3432		ADDI	\$13 1	3498		ADDI	\$1 1
3433		EXCH	\$14 \$13	3499		EXCH	\$10 \$1
3434		XOR	\$15 \$14	3500		ADDI	\$1 1
3435		EXCH	\$14 \$13	3501		EXCH	\$3 \$1
3436		ADDI	\$13 -1	3502		EXCH	\$11 \$7
3437	loadMetAdd_132_i:	EXCH	\$13 \$12	3503		EXCH	\$13 \$12
3438		XOR	\$12 \$11	3504		ADDI	\$13 0
3439		EXCH	\$11 \$7	3505		EXCH	\$14 \$13
3440		EXCH	\$11 \$6	3506		XOR	\$15 \$14
3441		ADD	\$12 \$3	3507		EXCH	\$14 \$13
3442		ADDI	\$12 2	3508		ADDI	\$13 0
3443		EXCH	\$13 \$12	3509	loadMetAdd_137_i:	EXCH	\$13 \$12
3444		ADDI	\$12 -2	3510		XOR	\$12 \$11
3445	105	SUB	\$12 \$3	3511		EXCH	\$11 \$7
3446	copy_136:	XOR	\$11 \$13	3512		ADD	\$11 \$3
3447		ADDI	\$13 1	3513		ADDI	\$11 2
3448		EXCH	\$14 \$13	3514		EXCH	\$12 \$11
3449		ADDI	\$14 1	3515		ADDI	\$11 -2
3450		EXCH	\$14 \$13	3516		SUB	\$11 \$3
3451		ADDI	\$13 -1	3517		XOR	\$13 \$12
3452		ADD	\$12 \$3	3518	loadMetAdd_141:	EXCH	\$14 \$13
3453		ADDI	\$12 2	3519		ADDI	\$14 1
3454		EXCH	\$13 \$12	3520		EXCH	\$15 \$14
3455		ADDI	\$12 -2	3521		XOR	\$16 \$15
3456 $3457$		SUB EXCH	\$12 \$3 \$11 \$6	3522 $3523$		EXCH ADDI	\$15 \$14 \$14 -1
3407		EACH	ATT A0	JJ <b>∠</b> J	I	דחחד	A T . T

3524		EXCH	\$14 \$13	3590		ADDI	\$1 1
3525		ADD	\$11 \$3	3591		EXCH	\$9 \$1
		ADDI		3592		XOR	\$9 \$0
3526			\$11 2				
3527		EXCH	\$12 \$11	3593	localBlock_147_i:	XOR	\$8 \$1
3528		ADDI	\$11 -2	3594		ADDI	\$1 1
3529		SUB	\$11 \$3	3595		EXCH	\$8 \$1
3530		EXCH	\$3 \$1	3596		XOR	\$8 \$0
					11511 140 :		
3531		ADDI	\$1 -1	3597	localBlock_148_i:	XOR	\$7 \$1
3532		EXCH	\$10 \$1	3598		ADDI	\$1 1
3533		ADDI	\$1 -1	3599		EXCH	\$7 \$1
3534		EXCH	\$9 \$1	3600		XOR	\$7 \$0
		ADDI			logalDlogk 140 i.	XOR	
3535			\$1 -1	3601	localBlock_149_i:		\$6 \$1
3536		EXCH	\$8 \$1	3602	l_initTape_3_bot:	BRA	
3537		ADDI	\$1 -1		l_initTape_3_top		
3538		EXCH	\$6 \$1	3603	l_init_4_top:	BRA	l_init_4_bot
3539		ADDI	\$1 -1	3604		ADDI	\$1 1
3540		EXCH	\$7 \$1	3605		EXCH	\$2 \$1
3541		ADDI	\$1 -1	3606		EXCH	\$3 \$1
3542		EXCH	\$13 \$1	3607		ADDI	\$1 -1
3543		ADDI	\$1 -1	3608	l_init_4:	SWAPBR	\$2.
		ADDI	\$16 -3544	3609		NEG	\$2
3544							
3545	l_rjmp_top_143:	RBRA	l_rjmp_bot_1	. <b>36</b> 10		ADDI	\$1 1
3546	l_jmp_142:	SWAPBR	\$16	3611		EXCH	\$3 \$1
3547		NEG	\$16	3612		EXCH	\$2 \$1
3548	l_rjmp_bot_144:	BRA	l_rjmp_top_1	46313		ADDI	\$1 -1
				- 1			
3549		ADDI	\$16 3544	3614		EXCH	\$3 \$1
3550		ADDI	\$1 1	3615		ADDI	\$1 -1
3551		EXCH	\$13 \$1	3616		BRA	
3552		ADDI	\$1 1			1 :	initLiterals_1
3553		EXCH	\$7 \$1	3617		ADDI	\$1 1
3554		ADDI	\$1 1	3618		EXCH	\$3 \$1
3555		EXCH	\$6 \$1	3619		EXCH	\$3 \$1
3556		ADDI	\$1 1	3620		ADDI	\$1 -1
3557		EXCH	\$8 \$1	3621		BRA	l_initRules_2
3558		ADDI	\$1 1	3622		ADDI	\$1 1
3559		EXCH	\$9 \$1	3623		EXCH	\$3 \$1
3560		ADDI	\$1 1	3624		EXCH	\$3 \$1
3561		EXCH	\$10 \$1	3625		ADDI	\$1 -1
3562		ADDI	\$1 1	3626		BRA	l_initTape_3
3563		EXCH	\$3 \$1	3627		ADDI	\$1 1
3564		ADD	\$11 \$3	3628		EXCH	\$3 \$1
3565		ADDI	\$11 2	3629		ADD	\$6 \$3
3566		EXCH	\$12 \$11	3630		ADDI	\$6 11
3567		ADDI	\$11 -2	3631		EXCH	\$7 \$6
3568		SUB	\$11 \$3	3632		ADDI	\$6 -11
3569		EXCH	\$14 \$13	3633		SUB	\$6 \$3
3570		ADDI	\$14 1	3634		XORI	\$8 1
3571		EXCH	\$15 \$14	3635		ADD	\$7 \$8
3572		XOR	\$16 \$15	3636		XORI	\$8 1
3573		EXCH	\$15 \$14	3637		ADD	\$6 \$3
3574	1 12 12 12 12 12	ADDI	\$14 -1	3638		ADDI	\$6 11
3575	loadMetAdd_141_i:	EXCH	\$14 \$13	3639		EXCH	\$7 \$6
3576		XOR	\$13 \$12	3640		ADDI	\$6 -11
3577		ADD	\$11 \$3	3641		SUB	\$6 \$3
3578		ADDI	\$11 2	3642		ADD	\$6 \$3
3579		EXCH	\$12 \$11	3643		ADDI	\$6 12
3580		ADDI	\$11 -2	3644		EXCH	\$7 \$6
3581		SUB	\$11 \$3	3645		ADDI	\$6 -12
3582		ADDI	\$1 1	3646		SUB	\$6 \$3
3583		EXCH	\$11 \$1	3647		XORI	\$8 1
3584	1 151 1 115 1	XOR	\$11 \$0	3648		ADD	\$7 \$8
3585	localBlock_145_i:	XOR	\$10 \$1	3649		XORI	\$8 1
3586		ADDI	\$1 1	3650		ADD	\$6 \$3
3587		EXCH	\$10 \$1	3651		ADDI	\$6 12
3588		XOR	\$10 \$0	3652		EXCH	\$7 \$6
	logalPlogh 146 :						\$6 -12
3089	localBlock_146_i:	XOR	\$9 \$1	3653		ADDI	40 -TZ

0054		CIID	ċ.c	ĊO	0510		*DD	¢0 ¢2
3654		SUB	\$6		3710		ADD	\$9 \$3
3655		ADD	\$6		3711		ADDI	\$9 12
3656		ADDI	\$6	13	3712		EXCH	\$10 \$9
3657		EXCH	\$7	\$6	3713		ADDI	\$9 -12
3658		ADDI		-13	3714		SUB	\$9 \$3
3659		SUB	\$6		3715		ADD	\$7 \$3
3660	:	XORI	\$8	6	3716		ADDI	\$7 11
3661		ADD	\$7	\$8	3717		EXCH	\$8 \$7
3662		XORI	\$8	6	3718		ADDI	\$7 -11
3663		ADD	\$6		3719		SUB	\$7 \$3
3664		ADDI	\$6		3720		ADD	\$7 \$3
3665		EXCH	\$7		3721		ADDI	\$7 2
3666		ADDI	\$6	-13	3722		EXCH	\$8 \$7
3667		SUB	\$6	\$3	3723		ADDI	\$7 -2
3668	:	EXCH	\$3	\$1	3724		SUB	\$7 \$3
3669		ADDI	\$1	-1	3725		XOR	\$9 \$8
		BRA			ate_53726	loadMo+1/dd 150.	EXCH	\$10 \$9
3670						loadMetAdd_158:		
3671		ADDI	\$1		3727		ADDI	\$10 3
3672	:	EXCH	\$3	\$1	3728		EXCH	\$11 \$10
3673	l_init_4_bot:	BRA	1_i	.nit_	4_top3729		XOR	\$12 \$11
3674	<pre>l_simulate_5_top:</pre>	BRA			3730		EXCH	\$11 \$10
	l_simulate_5_bot				3731		ADDI	\$10 -3
3675		ADDI	\$1	1	3732		EXCH	\$10 \$9
3676		EXCH	\$2		3733		ADD	\$7 \$3
		EXCH	\$3				ADDI	\$7 2
3677					3734			
3678		ADDI	\$1	-1	3735		EXCH	\$8 \$7
3679		SWAPBR			3736		ADDI	\$7 -2
3680		NEG	\$2		3737		SUB	\$7 \$3
3681		ADDI	\$1	1	3738		ADD	\$13 \$3
3682		EXCH	\$3	\$1	3739		ADDI	\$13 14
3683	:	EXCH	\$2	\$1	3740		EXCH	\$3 \$1
3684		ADDI	\$1		3741		ADDI	\$1 -1
3685		XORI	\$6		3742		EXCH	\$13 \$1
		BEQ	\$6		3743		ADDI	\$1 -1
3686	<del></del>	PFŐ	90	ŞÜ				
	assert_152				3744		EXCH	\$9 \$1
3687		ADD	\$7	\$3	3745		ADDI	\$1 -1
3688		ADDI	\$7	11	3746		ADDI	\$12 -3745
3689	:	EXCH	\$8	\$7	3747	1_jmp_159:	SWAPBR	\$12
3690		ADDI	\$7	-11	3748		NEG	\$12
3691		SUB	\$7	\$3	3749		ADDI	\$12 3745
3692		ADD	\$9		3750		ADDI	\$1 1
		ADDI	\$9				EXCH	\$9 \$1
3693					3751			
3694		EXCH		) \$9	3752		ADDI	\$1 1
3695		ADDI		-12	3753		EXCH	\$13 \$1
3696		SUB	\$9	\$3	3754		ADDI	\$1 1
3697	cmp_top_154:	BNE	\$8	\$10	3755		EXCH	\$3 \$1
	cmp_bot_155				3756		ADDI	\$13 -14
3698	<del>-</del>	XORI	\$11	. 1	3757		SUB	\$13 \$3
3699		BNE		\$10	3758		ADD	\$7 \$3
2300	cmp_top_154		, 0	0	3759		ADDI	\$7 2
3700		BEQ	¢11	\$0	3760		EXCH	\$8 \$7
3700	_ = -	PEŽ	ΥLI	. YU				
	f_bot_157		A		3761		ADDI	\$7 -2
3701		XORI	\$12		3762		SUB	\$7 \$3
3702		BEQ	\$11	. \$0	3763		EXCH	\$10 \$9
	f_top_156				3764		ADDI	\$10 3
3703		XOR	\$6	\$12	3765		EXCH	\$11 \$10
3704	f_bot_157_i:	BEQ	\$11	\$0	3766		XOR	\$12 \$11
	 f_top_156_i				3767		EXCH	\$11 \$10
3705	<del>-</del>	XORI	\$12	2 1	3768		ADDI	\$10 -3
3706		BEQ		\$0	3769	loadMetAdd_158_i:	EXCH	\$10 \$9
3100	_	-52	Y 1 1	. 70		10441101144_100_1.		
0505	f_bot_157_i	DME	÷ 0	610	3770		XOR	\$9 \$8
3707	i =     =     =	BNE	Şδ	\$10	3771		ADD	\$7 \$3
	cmp_top_154_i				3772		ADDI	\$7 2
3708		XORI	\$11		3773		EXCH	\$8 \$7
3709	cmp_top_154_i:	BNE	\$8	\$10	3774		ADDI	\$7 -2
	cmp_bot_155_i				3775		SUB	\$7 \$3
					·			

	_		60 61		I		67 16
3776		XCH	\$3 \$1	3842		ADDI	\$7 -16
3777		DDI RA	\$1 -1	3843		SUB XORI	\$7 \$3
3778			l_inst_6	3844			\$9 1 \$8 \$9
3779		DDI	\$1 1	3845		ADD	
3780		XCH	\$3 \$1	3846		XORI	\$9 1
3781		DD DD T	\$7 \$3	3847		ADD	\$7 \$3
3782		DDI	\$7 2	3848		ADDI	\$7 16
3783		XCH	\$8 \$7	3849		EXCH	\$8 \$7
3784		DDI	\$7 -2	3850		ADDI	\$7 -16
3785		UB	\$7 \$3	3851		SUB	\$7 \$3
3786		OR	\$9 \$8	3852		ADD	\$8 \$3
3787	_	XCH	\$10 \$9	3853		ADDI	\$8 16
3788		DDI	\$10 3	3854		EXCH	\$9 \$8
3789		XCH	\$11 \$10	3855		ADDI	\$8 -16
3790		OR	\$12 \$11	3856		SUB	\$8 \$3
3791		XCH	\$11 \$10	3857		ADD	\$10 \$3
3792		DDI	\$10 -3	3858		ADDI	\$10 15
3793		XCH	\$10 \$9	3859		EXCH	\$11 \$10
3794		DD	\$7 \$3	3860		ADDI	\$10 -15
3795		DDI	\$7 2	3861		SUB	\$10 \$3
3796		XCH	\$8 \$7	3862	cmp_top_168:	BNE	\$9 \$11
3797		DDI	\$7 -2		cmp_bot_169		
3798		UB	\$7 \$3	3863		XORI	\$12 1
3799		DD	\$13 \$3	3864	cmp_bot_169:	BNE	\$9 \$11
3800		DDI	\$13 14		cmp_top_168		
3801		XCH	\$3 \$1	3865	f_top_170:	BEQ	\$12 \$0
3802	A	DDI	\$1 -1		f_bot_171		
3803	E	XCH	\$13 \$1	3866		XORI	\$13 1
3804	A	DDI	\$1 -1	3867	f_bot_171:	BEQ	\$12 \$0
3805	E	XCH	\$9 \$1		f_top_170		
3806	A	DDI	\$1 -1	3868		XOR	\$7 \$13
3807	A	DDI	\$12 -3807	3869	f_bot_171_i:	BEQ	\$12 \$0
3808	l_rjmp_top_162: R	BRA	l_rjmp_bot_1	.63	f_top_170_i		
3809	l_jmp_161: S	WAPBR	\$12	3870		XORI	\$13 1
3810	N	EG	\$12	3871	f_top_170_i:	BEQ	\$12 \$0
3811	l_rjmp_bot_163: B	RA	l_rjmp_top_1	.62	f_bot_171_i		
3812	A	DDI	\$12 3807	3872	cmp_bot_169_i:	BNE	\$9 \$11
3813		DDI	\$1 1		cmp_top_168_i		
3814		XCH	\$9 \$1	3873		XORI	\$12 1
3815	A	DDI	\$1 1	3874	cmp_top_168_i:	BNE	\$9 \$11
3816	E	XCH	\$13 \$1		cmp_bot_169_i		
3817		DDI	\$1 1	3875		ADD	\$10 \$3
3818		XCH	\$3 \$1	3876		ADDI	\$10 15
3819		DDI	\$13 -14	3877		EXCH	\$11 \$10
3820		UB	\$13 \$3	3878		ADDI	\$10 -15
3821		DD	\$7 \$3	3879		SUB	\$10 \$3
3822		DDI	\$7 2	3880		ADD	\$8 \$3
3823		XCH	\$8 \$7	3881		ADDI	\$8 16
3824		DDI	\$7 -2	3882		EXCH	\$9 \$8
3825		UB	\$7 \$3	3883		ADDI	\$8 -16
3826		XCH	\$10 \$9	3884		SUB	\$8 \$3
3827		DDI	\$10 3	3885	test_164:	BEQ	\$7 \$0
3828		XCH	\$11 \$10		test_false_166		
3829		OR	\$12 \$11	3886		XORI	\$7 1
3830	E	XCH	\$11 \$10	3887		ADD	\$8 \$3
3831	A	DDI	\$10 -3	3888		ADDI	\$8 16
3832		XCH	\$10 \$9	3889		EXCH	\$9 \$8
3833		OR	\$9 \$8	3890		ADDI	\$8 -16
3834	A	DD	\$7 \$3	3891		SUB	\$8 \$3
3835	A	DDI	\$7 2	3892		ADD	\$10 \$3
3836	E	XCH	\$8 \$7	3893		ADDI	\$10 15
3837	A	DDI	\$7 -2	3894		EXCH	\$11 \$10
3838	S	UB	\$7 \$3	3895		ADDI	\$10 -15
3839	A	DD	\$7 \$3	3896		SUB	\$10 \$3
3840	A	DDI	\$7 16	3897		XOR	\$9 \$11
3841	E	XCH	\$8 \$7	3898		ADD	\$10 \$3

3899		ADDI	\$10 15	3952	f_bot_179_i:	BEQ	\$11 \$0
3900		EXCH	\$11 \$10		f_top_178_i		
3901		ADDI	\$10 -15	3953		XORI	\$12 1
3902		SUB	\$10 \$3		f_top_178_i:	BEQ	\$11 \$0
				3934	_	DEQ	711 70
3903		ADD	\$8 \$3		f_bot_179_i		
3904		ADDI	\$8 16	3955	cmp_bot_177_i:	BNE	\$8 \$10
3905		EXCH	\$9 \$8		cmp_top_176_i		
3906		ADDI	\$8 -16	3956		XORI	\$11 1
3907		SUB	\$8 \$3	3957	cmp_top_176_i:	BNE	\$8 \$10
3908		XORI	\$7 1		cmp bot 177 i		
3909	assert_true_165:	BRA	assert_167	3958		ADD	\$9 \$3
3910	test_false_166:	BRA	test_164	3959		ADDI	\$9 13
3911	assert_167:	BNE	\$7 \$0	3960		EXCH	\$10 \$9
	assert_true_165			3961		ADDI	\$9 -13
3912		ADD	\$8 \$3	3962		SUB	\$9 \$3
3913		ADDI	\$8 16	3963		ADD	\$7 \$3
3914		EXCH	\$9 \$8	3964		ADDI	\$7 11
3915		ADDI	\$8 -16	3965		EXCH	\$8 \$7
		SUB				ADDI	\$7 -11
3916	. 170		\$8 \$3	3966			
3917	cmp_top_172:	BNE	\$9 \$0	3967		SUB	\$7 \$3
	cmp_bot_173			3968	test_151:	BNE	\$6 \$0 exit_153
3918		XORI	\$10 1	3969	assert_152:	BRA	entry_150
3919	cmp_bot_173:	BNE	\$9 \$0	3970	exit_153:	BRA	test_151
	cmp_top_172			3971	_	XORI	\$6 1
2020	f_top_174:	BEQ	\$10 \$0	3972	l_simulate_5_bot:	BRA	
3920	_	DEQ	ATO AO	3912		DKA	
	f_bot_175		611 1		l_simulate_5_top		3
3921		XORI	\$11 1	3973	l_inst_6_top:	BRA	l_inst_6_bot
3922	f_bot_175:	BEQ	\$10 \$0	3974		ADDI	\$1 1
	f_top_174			3975		EXCH	\$2 \$1
3923		XOR	\$7 \$11	3976		EXCH	\$3 \$1
3924	f_bot_175_i:	BEQ	\$10 \$0	3977		ADDI	\$1 -1
	f_top_174_i	_		3978	l_inst_6:	SWAPBR	\$2
3925	1_00P_1/1_1	XORI	\$11 1	3979	<u></u>	NEG	\$2
	£ + 17/ ÷ .						
3926	f_top_174_i:	BEQ	\$10 \$0	3980		ADDI	\$1 1
	f_bot_175_i			3981		EXCH	\$3 \$1
3927	cmp_bot_173_i:	BNE	\$9 \$0	3982		EXCH	\$2 \$1
	cmp_top_172_i			3983		ADDI	\$1 -1
3928		XORI	\$10 1	3984		ADD	\$7 \$3
3929	cmp_top_172_i:	BNE	\$9 \$0	3985		ADDI	\$7 11
	cmp_bot_173_i		1 - 1 -	3986		EXCH	\$8 \$7
3930	Cmp_80e_173_1	ADD	¢0 ¢2			ADDI	\$7 -11
			\$8 \$3	3987			
3931		ADDI	\$8 16	3988		SUB	\$7 \$3
3932		EXCH	\$9 \$8	3989		ADD	\$9 \$3
3933		ADDI	\$8 -16	3990		ADDI	\$9 3
3934		SUB	\$8 \$3	3991		ADD	\$10 \$3
3935		ADD	\$7 \$3	3992		ADDI	\$10 16
3936		ADDI	\$7 11	3993		EXCH	\$11 \$10
3937		EXCH	\$8 \$7	3994		ADDI	\$10 -16
3938		ADDI	\$7 -11	3995		SUB	\$10 \$3
3939		SUB	\$7 \$3	3996		EXCH	\$13 \$9
3940		ADD	\$9 \$3	3997		XOR	\$12 \$13
3941		ADDI	\$9 13	3998		EXCH	\$13 \$9
3942		EXCH	\$10 \$9	3999		ADDI	\$12 2
3943		ADDI	\$9 -13	4000		ADD	\$12 \$11
3944		SUB	\$9 \$3	4001		ADD	\$10 \$3
3945	cmp_top_176:	BNE	\$8 \$10	4002		ADDI	\$10 16
-5.0	cmp_bot_177			4003		EXCH	\$11 \$10
20.40	Cmp_D0C_1 / /	VORT	¢11 1				
3946		XORI	\$11 1	4004		ADDI	\$10 -16
3947	cmp_bot_177:	BNE	\$8 \$10	4005		SUB	\$10 \$3
	cmp_top_176			4006		ADDI	\$9 -3
3948	f_top_178:	BEQ	\$11 \$0	4007		SUB	\$9 \$3
	f_bot_179			4008		EXCH	\$14 \$12
3949		XORI	\$12 1	4009		ADD	\$9 \$3
	f_bot_179:	BEQ	\$11 \$0	4010		ADDI	\$9 3
	f_top_178	- 2		4011		ADD	\$10 \$3
3951		XOR	\$6 \$12	4011		ADDI	\$10 16
5951	I	AUR	YU Y14	4012	l	TUUL	4 T O T O

4013		EXCH	\$11 \$10	4076		XORI	\$24 1
4014		ADDI	\$10 -16	4077	cmp_bot_187:	BNE	\$17 \$23
4015		SUB	\$10 \$3		cmp_top_186		
4016		SUB	\$12 \$11	4078		ANDX	\$25 \$15 \$24
4017		ADDI	\$12 -2	4079	f_top_188:	BEQ	\$25 \$0
4018		EXCH	\$13 \$9		f_bot_189	_	
4019		XOR	\$12 \$13	4080		XORI	\$26 1
4020		EXCH	\$13 \$9	4081	f_bot_189:	BEQ	\$25 \$0
4021		ADD	\$10 \$3		f_top_188	~	
4022		ADDI	\$10 16	4082	1_00P_100	XOR	\$6 \$26
4023		EXCH	\$11 \$10	4083	f bot 189 i:	BEQ	\$25 \$0
4024		ADDI	\$10 -16	1000	f_top_188_i	z	120 10
4025		SUB	\$10 \$3	4084	1_00P_100_1	XORI	\$26 1
4026		ADDI	\$9 -3	4085	f_top_188_i:	BEQ	\$25 \$0
4027		SUB	\$9 \$3	4000	f_bot_189_i	DEQ	V23 V0
4028	cmp_top_184:	BNE	\$8 \$14	4086	1_500_105_1	ANDX	\$25 \$15 \$24
4020		DNE	A0 A14		amp bot 197 i.	BNE	\$17 \$23
4000	cmp_bot_185	XORI	\$15 1	4087	cmp_bot_187_i:	DNE	71/ 723
4029	amp ba+ 10E.			4000	cmp_top_186_i	VODT	¢24 1
4030	cmp_bot_185:	BNE	\$8 \$14	4088	t 10C :	XORI	\$24 1
	cmp_top_184		416 42	4089	cmp_top_186_i:	BNE	\$17 \$23
4031		ADD	\$16 \$3		cmp_bot_187_i		410 40
4032		ADDI	\$16 14	4090		ADD	\$18 \$3
4033		EXCH	\$17 \$16	4091		ADDI	\$18 5
4034		ADDI	\$16 -14	4092		ADD	\$19 \$3
4035		SUB	\$16 \$3	4093		ADDI	\$19 16
4036		ADD	\$18 \$3	4094		EXCH	\$20 \$19
4037		ADDI	\$18 5	4095		ADDI	\$19 -16
4038		ADD	\$19 \$3	4096		SUB	\$19 \$3
4039		ADDI	\$19 16	4097		EXCH	\$22 \$18
4040		EXCH	\$20 \$19	4098		XOR	\$21 \$22
4041		ADDI	\$19 -16	4099		EXCH	\$22 \$18
4042		SUB	\$19 \$3	4100		ADDI	\$21 2
4043		EXCH	\$22 \$18	4101		ADD	\$21 \$20
4044		XOR	\$21 \$22	4102		ADD	\$19 \$3
4045		EXCH	\$22 \$18	4103		ADDI	\$19 16
4046		ADDI	\$21 2	4104		EXCH	\$20 \$19
4047		ADD	\$21 \$20	4105		ADDI	\$19 -16
4048		ADD	\$19 \$3	4106		SUB	\$19 \$3
4049		ADDI	\$19 16	4107		ADDI	\$18 -5
4050		EXCH	\$20 \$19	4108		SUB	\$18 \$3
4051		ADDI	\$19 -16	4109		EXCH	\$23 \$21
4052		SUB	\$19 \$3	4110		ADD	\$18 \$3
4053		ADDI	\$18 -5	4111		ADDI	\$18 5
4054		SUB	\$18 \$3	4112		ADD	\$19 \$3
4055		EXCH	\$23 \$21	4113		ADDI	\$19 16
4056		ADD	\$18 \$3	4114		EXCH	\$20 \$19
4057		ADDI	\$18 5	4115		ADDI	\$19 -16
4058		ADD	\$19 \$3	4116		SUB	\$19 \$3
4059		ADDI	\$19 16	4117		SUB	\$21 \$20
4060		EXCH	\$20 \$19	4118		ADDI	\$21 -2
4061		ADDI	\$19 -16	4119		EXCH	\$22 \$18
4062		SUB	\$19 \$3	4120		XOR	\$21 \$22
4063		SUB	\$21 \$20	4121		EXCH	\$22 \$18
4064		ADDI	\$21 -2	4122		ADD	\$19 \$3
4065		EXCH	\$22 \$18	4123		ADDI	\$19 16
4066		XOR	\$21 \$22	4124		EXCH	\$20 \$19
4067		EXCH	\$22 \$18	4125		ADDI	\$19 -16
4068		ADD	\$19 \$3	4126		SUB	\$19 \$3
4069		ADDI	\$19 16	4127		ADDI	\$18 -5
4070		EXCH	\$20 \$19	4128		SUB	\$18 \$3
4071		ADDI	\$19 -16	4129		ADD	\$16 \$3
4072		SUB	\$19 \$3	4130		ADDI	\$16 14
4073		ADDI	\$18 -5	4131		EXCH	\$17 \$16
4074		SUB	\$18 \$3	4132		ADDI	\$16 -14
4075	cmp_top_186:	BNE	\$17 \$23	4133		SUB	\$16 \$3
-3.3	cmp_bot_187		. = : +20		cmp_bot_185_i:	BNE	\$8 \$14
	1			1101			. = 1 = *

	cmp_top_184_i			4198	ADDI	\$12 2
4135		XORI	\$15 1	4199	ADD	\$12 \$11
4136	cmp_top_184_i:	BNE	\$8 \$14	4200	ADD	\$10 \$3
	cmp_bot_185_i			4201	ADDI	\$10 16
4137		ADD	\$9 \$3	4202	EXCH	\$11 \$10
4138		ADDI	\$9 3	4203	ADDI	\$10 -16
4139 4140		ADD ADDI	\$10 \$3 \$10 16	4204 4205	SUB ADDI	\$10 \$3 \$9 -4
4141		EXCH	\$10 10	4206	SUB	\$9 \$3
4142		ADDI	\$10 -16	4207	EXCH	\$14 \$12
4143		SUB	\$10 \$3	4208	ADD	\$9 \$3
4144		EXCH	\$13 \$9	4209	ADDI	\$9 4
4145		XOR	\$12 \$13	4210	ADD	\$10 \$3
4146		EXCH	\$13 \$9	4211	ADDI	\$10 16
4147		ADDI	\$12 2	4212	EXCH	\$11 \$10
4148		ADD	\$12 \$11	4213	ADDI	\$10 -16
4149		ADD	\$10 \$3	4214	SUB	\$10 \$3
4150		ADDI	\$10 16	4215	SUB	\$12 \$11
4151 $4152$		EXCH ADDI	\$11 \$10 \$10 -16	4216 4217	ADDI EXCH	\$12 -2 \$13 \$9
4153		SUB	\$10 \$3	4218	XOR	\$12 \$13
4154		ADDI	\$9 -3	4219	EXCH	\$13 \$9
4155		SUB	\$9 \$3	4220	ADD	\$10 \$3
4156		EXCH	\$14 \$12	4221	ADDI	\$10 16
4157		ADD	\$9 \$3	4222	EXCH	\$11 \$10
4158		ADDI	\$9 3	4223	ADDI	\$10 -16
4159		ADD	\$10 \$3	4224	SUB	\$10 \$3
4160		ADDI	\$10 16	4225	ADDI	\$9 -4
4161		EXCH	\$11 \$10	4226	SUB	\$9 \$3
4162 4163		ADDI SUB	\$10 -16 \$10 \$3	4227 4228	ADD ADDI	\$15 \$3 \$15 3
4164		SUB	\$10 \$3	4228	ADDI	\$16 \$3
4165		ADDI	\$12 -2	4230	ADDI	\$16 16
4166		EXCH	\$13 \$9	4231	EXCH	\$17 \$16
4167		XOR	\$12 \$13	4232	ADDI	\$16 -16
4168		EXCH	\$13 \$9	4233	SUB	\$16 \$3
4169		ADD	\$10 \$3	4234	EXCH	\$19 \$15
4170		ADDI	\$10 16	4235	XOR	\$18 \$19
4171		EXCH	\$11 \$10	4236	EXCH	\$19 \$15
4172 $4173$		ADDI SUB	\$10 -16 \$10 \$3	4237 4238	ADDI ADD	\$18 2 \$18 \$17
4173		ADDI	\$9 -3	4238	ADD	\$16 \$3
4175		SUB	\$9 \$3	4240	ADDI	\$16 16
4176		ADD	\$7 \$3	4241	EXCH	\$17 \$16
4177		ADDI	\$7 11	4242	ADDI	\$16 -16
4178		EXCH	\$8 \$7	4243	SUB	\$16 \$3
4179		ADDI	\$7 -11	4244	ADDI	\$15 -3
4180		SUB	\$7 \$3	4245	SUB	\$15 \$3
4181	test_180:	BEQ	\$6 \$0	4246	EXCH	\$20 \$18
4182	test_false_182	XORI	\$6 1	4247 4248	ADD ADDI	\$15 \$3 \$15 3
4182		ADD	\$6 I \$7 \$3	4248	ADD1 ADD	\$15 3 \$16 \$3
4184		ADDI	\$7 11	4250	ADDI	\$16 16
4185		EXCH	\$8 \$7	4251	EXCH	\$17 \$16
4186		ADDI	\$7 -11	4252	ADDI	\$16 -16
4187		SUB	\$7 \$3	4253	SUB	\$16 \$3
4188		ADD	\$9 \$3	4254	SUB	\$18 \$17
4189		ADDI	\$9 4	4255	ADDI	\$18 -2
4190		ADD	\$10 \$3 \$10 16	4256	EXCH	\$19 \$15
4191 4192		ADDI EXCH	\$10 16 \$11 \$10	4257 4258	XOR EXCH	\$18 \$19 \$19 \$15
4192		ADDI	\$10 -16	4258 4259	ADD	\$19 \$15
4194		SUB	\$10 \$3	4260	ADDI	\$16 16
4195		EXCH	\$13 \$9	4261	EXCH	\$17 \$16
4196		XOR	\$12 \$13	4262	ADDI	\$16 -16
4197		EXCH	\$13 \$9	4263	SUB	\$16 \$3

4264	ADI	οI	\$15	-3	4330			ADD	\$9 \$3
4265	SUE	3	\$15	\$3	433			ADDI	\$9 4
4266	XOI			\$14	4333	:		ADD	\$10 \$3
4267	SUI			\$20	4333			ADDI	\$10 16
4268	ADI		\$8 :		433	1		EXCH	\$11 \$10
4269	ADI			\$20	433	i		ADDI	\$10 -16
4270 $4271$	XOI ADI		\$15	\$14 \$3	4330 433'	1		SUB	\$10 \$3 \$12 \$11
4271	ADI		\$15	3	433			ADDI	\$12 -2
4273	ADI		\$16		4339	1		EXCH	\$13 \$9
4274	ADI		\$16		4340	1		XOR	\$12 \$13
4275	EXC			\$16	434	1		EXCH	\$13 \$9
4276	ADI	ΟI	\$16	-16	4345	:		ADD	\$10 \$3
4277	SUE	3	\$16	\$3	434	:		ADDI	\$10 16
4278	EXC		\$19	\$15	434			EXCH	\$11 \$10
4279	XOI		\$18	\$19	434			ADDI	\$10 -16
4280	EXC			\$15	4340	1		SUB	\$10 \$3
4281	ADI				434	i		ADDI	\$9 -4
4282 4283	ADI ADI		\$16	\$17	4348 4349	1		SUB ADD	\$9 \$3 \$7 \$3
4284	ADI		\$16		434			ADDI	\$7 11
4285	EXC			\$16	435	1		EXCH	\$8 \$7
4286	ADI			-16	435	i		ADDI	\$7 -11
4287	SUI	3	\$16		435	1		SUB	\$7 \$3
4288	ADI	ΟI	\$15	-3	435	L		ADD	\$7 \$3
4289	SUE	3	\$15	\$3	435	i		ADDI	\$7 14
4290	EXC			\$18	4350	s		EXCH	\$8 \$7
4291	ADI		\$15		435'			ADDI	\$7 -14
4292	ADI		\$15		4358	1		SUB	\$7 \$3
4293	ADI				4359	1		ADD	\$9 \$3
4294 4295	ADI EXC		\$16	\$16	4360 4360	1		ADDI ADD	\$9 6 \$10 \$3
4296	ADI			-16	436			ADDI	\$10 16
4297	SUE		\$16		436	1		EXCH	\$11 \$10
4298	SUI		\$18	\$17	436	i		ADDI	\$10 -16
4299	ADI	ΟI	\$18	-2	436	s		SUB	\$10 \$3
4300	EXC	CH	\$19	\$15	4360	;		EXCH	\$13 \$9
4301	XOI			\$19	436'	1		XOR	\$12 \$13
4302	EXC			\$15	4368	1		EXCH	\$13 \$9
4303	ADI		\$16		4369			ADDI	\$12 2
4304	ADI EXC		\$16	16 \$16	4370	1		ADD ADD	\$12 \$11 \$10 \$3
4305 4306	ADI			-16	437 437	1		ADDI	\$10 \$3 \$10 16
4307	SUE		\$16		437	1		EXCH	\$11 \$10
4308	ADI		\$15		437			ADDI	\$10 -16
4309	SUE	3	\$15		437	s		SUB	\$10 \$3
4310	ADI	)	\$9 5	\$3	4370	;		ADDI	\$9 -6
4311	ADI		\$9 4		437'			SUB	\$9 \$3
4312	ADI		\$10		4378	1		EXCH	\$14 \$12
4313	ADI		\$10		4379	1		ADD	\$9 \$3
4314 4315	EXC ADI			\$10 -16	4380	i .		ADDI ADD	\$9 6 \$10 \$3
4316	SUI		\$10		438			ADDI	\$10 35
4317	EXC		\$13		4383			EXCH	\$11 \$10
4318	XOI			\$13	438	i		ADDI	\$10 -16
4319	EXC		\$13		438	1		SUB	\$10 \$3
4320	ADI	ΟI	\$12		4386	5		SUB	\$12 \$11
4321	ADI			\$11	438'	1		ADDI	\$12 -2
4322	ADI		\$10		4388	1		EXCH	\$13 \$9
4323	ADI		\$10		4389			XOR	\$12 \$13
4324	EXC			\$10 -16	4390	1		EXCH	\$13 \$9
4325 4326	ADI SUI		\$10	-16 \$3	439	!		ADD ADDI	\$10 \$3 \$10 16
4327	ADI		\$9 -		439	i .		EXCH	\$11 \$10
4328	SUE		\$9 :		439			ADDI	\$10 -16
4329	EXC			\$12	439	1		SUB	\$10 \$3
,						•			

4396	ADDI	\$9 -6	4462		ADD	\$15 \$3
4397	SUB	\$9 \$3	4463		ADDI	\$15 5
4398	ADD	\$15 \$3	4464		ADD	\$16 \$3
4399	ADDI	\$15 5	4465		ADDI	\$16 16
4400	ADD	\$16 \$3	4466		EXCH	\$17 \$16
4401	ADDI	\$16 16	4467		ADDI	\$16 -16
4402	EXCH	\$17 \$16	4468		SUB	\$16 \$3
4403	ADDI	\$16 -16	4469		SUB	\$18 \$17
4404	SUB	\$16 \$3	4470		ADDI	\$18 -2
4405	EXCH	\$19 \$15	4471		EXCH	\$19 \$15
4406	XOR	\$18 \$19	4472		XOR	\$18 \$19
4407	EXCH	\$19 \$15	4473		EXCH	\$19 \$15
4408	ADDI	\$18 2	4474		ADD	\$16 \$3
4409	ADD	\$18 \$17	4475		ADDI	\$16 16
4410	ADD ADDI	\$16 \$3	4476		EXCH ADDI	\$17 \$16
4411	EXCH	\$16 16	4477		SUB	\$16 -16
4412 4413	ADDI	\$17 \$16 \$16 -16	4478 $4479$		ADDI	\$16 \$3 \$15 -5
4414	SUB	\$16 \$3	4480		SUB	\$15 \$3
4415	ADDI	\$15 -5	4481		ADD	\$9 \$3
4416	SUB	\$15 \$3	4482		ADDI	\$9 6
4417	EXCH	\$20 \$18	4483		ADD	\$10 \$3
4418	ADD	\$15 \$3	4484		ADDI	\$10 16
4419	ADDI	\$15 5	4485		EXCH	\$11 \$10
4420	ADD	\$16 \$3	4486		ADDI	\$10 -16
4421	ADDI	\$16 16	4487		SUB	\$10 \$3
4422	EXCH	\$17 \$16	4488		EXCH	\$13 \$9
4423	ADDI	\$16 -16	4489		XOR	\$12 \$13
4424	SUB	\$16 \$3	4490		EXCH	\$13 \$9
4425	SUB	\$18 \$17	4491		ADDI	\$12 2
4426	ADDI	\$18 -2	4492		ADD	\$12 \$11
4427	EXCH	\$19 \$15	4493		ADD	\$10 \$3
4428	XOR	\$18 \$19	4494		ADDI	\$10 16
4429 4430	EXCH ADD	\$19 \$15 \$16 \$3	4495 4496		EXCH ADDI	\$11 \$10 \$10 -16
4431	ADDI	\$16 16	4497		SUB	\$10 \$3
4432	EXCH	\$17 \$16	4498		ADDI	\$9 -6
4433	ADDI	\$16 -16	4499		SUB	\$9 \$3
4434	SUB	\$16 \$3	4500		EXCH	\$14 \$12
4435	ADDI	\$15 -5	4501		ADD	\$9 \$3
4436	SUB	\$15 \$3	4502		ADDI	\$9 6
4437	XOR	\$21 \$14	4503		ADD	\$10 \$3
4438	SUB	\$21 \$20	4504		ADDI	\$10 16
4439	ADD	\$8 \$21	4505		EXCH	\$11 \$10
4440	ADD	\$21 \$20	4506		ADDI	\$10 -16
4441	XOR	\$21 \$14	4507		SUB	\$10 \$3
4442	ADD ADDT	\$15 \$3 \$15 5	4508		SUB	\$12 \$11 \$12 =2
4443 4444	ADDI ADD	\$15 5 \$16 \$3	4509 4510		ADDI EXCH	\$12 -2 \$13 \$9
4444	ADDI	\$16 16	4510		XOR	\$12 \$13
4446	EXCH	\$17 \$16	4512		EXCH	\$13 \$9
4447	ADDI	\$16 -16	4513		ADD	\$10 \$3
4448	SUB	\$16 \$3	4514		ADDI	\$10 16
4449	EXCH	\$19 \$15	4515		EXCH	\$11 \$10
4450	XOR	\$18 \$19	4516		ADDI	\$10 -16
4451	EXCH	\$19 \$15	4517		SUB	\$10 \$3
4452	ADDI	\$18 2	4518		ADDI	\$9 -6
4453	ADD	\$18 \$17	4519		SUB	\$9 \$3
4454	ADD	\$16 \$3	4520		ADD	\$7 \$3
4455	ADDI	\$16 16	4521		ADDI	\$7 14
4456	EXCH	\$17 \$16 \$16 -16	4522		EXCH	\$8 \$7 \$7 _14
4457 4458	ADDI SUB	\$16 -16 \$16 \$3	4523 $4524$		ADDI SUB	\$7 -14 \$7 \$3
4458	ADDI	\$15 -5	4524		XORI	\$6 1
4460	SUB	\$15 \$3	4526	assert_true_181:	BRA	assert_183
4461	EXCH	\$20 \$18		test_false_182:	BRA	test_180
- "						

1				1		
4528 assert_183:	BNE	\$6 \$0	4591		ADDI	\$21 2
assert_true_181		+= +o	4592		ADD	\$21 \$20
4529	ADD	\$7 \$3	4593		ADD	\$19 \$3
4530	ADDI	\$7 11	4594		ADDI	\$19 16
4531	EXCH	\$8 \$7	4595		EXCH	\$20 \$19
4532	ADDI	\$7 -11	4596		ADDI	\$19 -16
4533	SUB	\$7 \$3	4597		SUB	\$19 \$3
4534	ADD	\$9 \$3	4598		ADDI	\$18 -6
4535	ADDI ADD	\$9 4	4599		SUB	\$18 \$3
4536	ADDI	\$10 \$3 \$10 16	4600		EXCH ADD	\$23 \$21 \$18 \$3
4537 4538	EXCH	\$10 16	4601 4602		ADDI	\$18 6
	ADDI	\$10 -16			ADDI	\$10 0
4539 4540	SUB	\$10 -16	4603 4604		ADDI	\$19 33
4541	EXCH	\$10 \$3	4604		EXCH	\$20 \$19
4541	XOR	\$12 \$13	4606		ADDI	\$19 -16
4542	EXCH	\$13 \$9	4607		SUB	\$19 \$3
4544	ADDI	\$12 2	4608		SUB	\$21 \$20
4545	ADD	\$12 \$11	4609		ADDI	\$21 -2
4546	ADD	\$10 \$3	4610		EXCH	\$22 \$18
4547	ADDI	\$10 16	4611		XOR	\$21 \$22
4548	EXCH	\$11 \$10	4612		EXCH	\$22 \$18
4549	ADDI	\$10 -16	4613		ADD	\$19 \$3
4550	SUB	\$10 \$3	4614		ADDI	\$19 16
4551	ADDI	\$9 -4	4615		EXCH	\$20 \$19
4552	SUB	\$9 \$3	4616		ADDI	\$19 -16
4553	EXCH	\$14 \$12	4617		SUB	\$19 \$3
4554	ADD	\$9 \$3	4618		ADDI	\$18 -6
4555	ADDI	\$9 4	4619		SUB	\$18 \$3
4556	ADD	\$10 \$3	4620	cmp_top_192:	BNE	\$17 \$23
4557	ADDI	\$10 16		cmp_bot_193		
4558	EXCH	\$11 \$10	4621		XORI	\$24 1
4559	ADDI	\$10 -16	4622	cmp_bot_193:	BNE	\$17 \$23
4560	SUB	\$10 \$3		cmp_top_192		
4561	SUB	\$12 \$11	4623		ANDX	\$25 \$15 \$24
4562	ADDI	\$12 -2	4624	f_top_194:	BEQ	\$25 \$0
4563	EXCH	\$13 \$9		f_bot_195		
4564	XOR	\$12 \$13	4625		XORI	\$26 1
4565	EXCH	\$13 \$9	4626	f_bot_195:	BEQ	\$25 \$0
4566	ADD	\$10 \$3		f_top_194		
4567	ADDI	\$10 16	4627		XOR	\$6 \$26
4568	EXCH	\$11 \$10	4628		BEQ	\$25 \$0
4569	ADDI	\$10 -16	4000	f_top_194_i	VODT	¢0.6 1
4570	SUB	\$10 \$3	4629	£ + 104 ÷ .	XORI	\$26 1
4571	ADDI SUB	\$9 -4 \$9 \$3	4630		BEQ	\$25 \$0
4572 4573 cmp_top_190:	BNE	\$8 \$14	4631	f_bot_195_i	ANDX	\$25 \$15 \$24
cmp_bot_191		70 717	4632	  cmp_bot_193_i:	BNE	\$17 \$23
4574	XORI	\$15 1	1002	cmp_top_192_i		, 1, 120
4575 cmp_bot_191:	BNE	\$8 \$14	4633		XORI	\$24 1
cmp_top_190			4634	cmp_top_192_i:	BNE	\$17 \$23
4576	ADD	\$16 \$3		cmp_bot_193_i	•	
4577	ADDI	\$16 14	4635		ADD	\$18 \$3
4578	EXCH	\$17 \$16	4636		ADDI	\$18 6
4579	ADDI	\$16 -14	4637		ADD	\$19 \$3
4580	SUB	\$16 \$3	4638		ADDI	\$19 16
4581	ADD	\$18 \$3	4639		EXCH	\$20 \$19
4582	ADDI	\$18 6	4640		ADDI	\$19 -16
4583	ADD	\$19 \$3	4641		SUB	\$19 \$3
4584	ADDI	\$19 16	4642		EXCH	\$22 \$18
4585	EXCH	\$20 \$19	4643		XOR	\$21 \$22
4586	ADDI	\$19 -16	4644		EXCH	\$22 \$18
4587	SUB	\$19 \$3	4645		ADDI	\$21 2
4588	EXCH	\$22 \$18	4646		ADD	\$21 \$20
4589	XOR	\$21 \$22	4647		ADD	\$19 \$3
4590	EXCH	\$22 \$18	4648		ADDI	\$19 16

1			400 4	10			412 40
4649		EXCH	\$20 \$			EXCH	\$13 \$9
4650		ADDI	\$19 -			ADD	\$10 \$3
4651		SUB	\$19 \$	3 4715		ADDI	\$10 16
4652		ADDI	\$18 -	6 4716		EXCH	\$11 \$10
4653		SUB	\$18 \$	3 4717		ADDI	\$10 -16
4654		EXCH	\$23 \$3	21 4718		SUB	\$10 \$3
4655		ADD	\$18 \$	3 4719		ADDI	\$9 -4
4656		ADDI	\$18 6	4720		SUB	\$9 \$3
4657		ADD	\$19 \$			ADD	\$7 \$3
4658		ADDI	\$19 1			ADDI	\$7 11
4659		EXCH	\$20 \$			EXCH	\$8 \$7
4660		ADDI	\$19 -			ADDI	\$7 -11
							\$7 \$3
4661		SUB	\$19 \$			SUB	
4662		SUB	\$21 \$			ADD	\$7 \$3
4663		ADDI	\$21 -			ADDI	\$7 11
4664		EXCH	\$22 \$			EXCH	\$8 \$7
4665		XOR	\$21 \$			ADDI	\$7 -11
4666		EXCH	\$22 \$			SUB	\$7 \$3
4667		ADD	\$19 \$	3 4731		ADD	\$9 \$3
4668		ADDI	\$19 1	6 4732		ADDI	\$9 3
4669		EXCH	\$20 \$	19 4733		ADD	\$10 \$3
4670		ADDI	\$19 -	16 4734		ADDI	\$10 16
4671		SUB	\$19 \$	3 4735		EXCH	\$11 \$10
4672		ADDI	\$18 -			ADDI	\$10 -16
4673		SUB	\$18 \$			SUB	\$10 \$3
4674		ADD	\$16 \$			EXCH	\$13 \$9
4675		ADDI	\$16 1			XOR	\$12 \$13
4676		EXCH	\$17 \$			EXCH	\$13 \$9
		ADDI	\$16 -			ADDI	\$12 2
4677							
4678	101 : .	SUB	\$16 \$			ADD	\$12 \$11
4679	cmp_bot_191_i:	BNE	\$8 \$1			ADD	\$10 \$3
	cmp_top_190_i			4744		ADDI	\$10 16
4680		XORI	\$15 1	4745		EXCH	\$11 \$10
4681	cmp_top_190_i:	BNE	\$8 \$1			ADDI	\$10 -16
	cmp_bot_191_i			4747		SUB	\$10 \$3
4682		ADD	\$9 \$3	4748		ADDI	\$9 -3
4683		ADDI	\$9 4	4749		SUB	\$9 \$3
4684		ADD	\$10 \$	3 4750		EXCH	\$14 \$12
4685		ADDI	\$10 1	6 4751		ADD	\$9 \$3
4686		EXCH	\$11 \$	10 4752		ADDI	\$9 3
4687		ADDI	\$10 -	16 4753		ADD	\$10 \$3
4688		SUB	\$10 \$	3 4754		ADDI	\$10 16
4689		EXCH	\$13 \$			EXCH	\$11 \$10
4690		XOR	\$12 \$			ADDI	\$10 -16
4691		EXCH	\$13 \$			SUB	\$10 \$3
4692		ADDI	\$12 2	4758		SUB	\$12 \$11
4693		ADDI	\$12 \$			ADDI	\$12 -2
4694		ADD	\$10 \$			EXCH	\$13 \$9
		ADDI				XOR	
4695			\$10 1				\$12 \$13
4696		EXCH	\$11 \$			EXCH	\$13 \$9
4697		ADDI	\$10 -			ADD	\$10 \$3
4698		SUB	\$10 \$			ADDI	\$10 16
4699		ADDI	\$9 -4	4765		EXCH	\$11 \$10
4700		SUB	\$9 \$3	4766		ADDI	\$10 -16
4701		EXCH	\$14 \$			SUB	\$10 \$3
4702		ADD	\$9 \$3	4768		ADDI	\$9 -3
4703		ADDI	\$9 4	4769		SUB	\$9 \$3
4704		ADD	\$10 \$	3 4770	cmp_top_200:	BNE	\$8 \$14
4705		ADDI	\$10 1	6	cmp_bot_201		
4706		EXCH	\$11 \$	10 4771		XORI	\$15 1
4707		ADDI	\$10 -		cmp_bot_201:	BNE	\$8 \$14
4708		SUB	\$10 \$		cmp_top_200		•
4709		SUB	\$12 \$		1	ADD	\$16 \$3
4710		ADDI	\$12 -			ADDI	\$16 5
4711		EXCH	\$13 \$			ADD	\$17 \$3
4711		XOR	\$12 \$			ADDI	\$17 16
4112		AUA	Y ± 4	+110		ADDI	Y 1 1 1 1 1 1

ı			+						+00 =
4777		EXCH	\$18			4835		ADDI	\$22 -7
4778		ADDI	\$17	-16		4836		SUB	\$22 \$3
4779		SUB	\$17	\$3		4837		ADD	\$16 \$3
4780		EXCH	\$20	\$16		4838		ADDI	\$16 5
4781		XOR	\$19	\$20		4839		ADD	\$17 \$3
4782		EXCH	\$20	\$16		4840		ADDI	\$17 16
4783		ADDI	\$19	2		4841		EXCH	\$18 \$17
4784		ADD	\$19			4842		ADDI	\$17 -16
4785		ADD	\$17			4843		SUB	\$17 \$3
4786		ADDI	\$17			4844		EXCH	\$20 \$16
4787		EXCH	\$18			4845		XOR	\$19 \$20
4788		ADDI	\$17			4846		EXCH	\$20 \$16
		SUB						ADDI	
4789			\$17			4847			\$19 2
4790		ADDI	\$16			4848		ADD	\$19 \$18
4791		SUB	\$16			4849		ADD	\$17 \$3
4792		EXCH	\$21			4850		ADDI	\$17 16
4793		ADD	\$16			4851		EXCH	\$18 \$17
4794		ADDI	\$16	5		4852		ADDI	\$17 -16
4795		ADD	\$17	\$3		4853		SUB	\$17 \$3
4796		ADDI	\$17	16		4854		ADDI	\$16 -5
4797		EXCH	\$18	\$17		4855		SUB	\$16 \$3
4798		ADDI	\$17	-16		4856		EXCH	\$21 \$19
4799		SUB	\$17	\$3		4857		ADD	\$16 \$3
4800		SUB	\$19	\$18		4858		ADDI	\$16 5
4801		ADDI	\$19	-2		4859		ADD	\$17 \$3
4802		EXCH	\$20			4860		ADDI	\$17 16
4803		XOR	\$19			4861		EXCH	\$18 \$17
4804		EXCH	\$20			4862		ADDI	\$17 -16
4805		ADD	\$17			4863		SUB	\$17 \$3
4806		ADDI	\$17			4864		SUB	\$19 \$18
4807		EXCH	\$18			4865		ADDI	\$19 -2
4808		ADDI	\$17			4866		EXCH	\$20 \$16
4809		SUB	\$17			4867		XOR	\$19 \$20
4810		ADDI	\$16			4868		EXCH	\$20 \$16
4811		SUB	\$16			4869		ADD	\$17 \$3
4812		ADD	\$22			4870		ADDI	\$17 16
4813		ADDI	\$22			4871		EXCH	\$18 \$17
4814		EXCH	\$23			4872		ADDI	\$17 -16
4815		ADDI	\$22			4873		SUB	\$17 \$3
4816		SUB	\$22	\$3		4874		ADDI	\$16 -5
4817	cmp_top_202:	BNE	\$21	\$23		4875		SUB	\$16 \$3
	cmp_bot_203					4876	cmp_bot_201_i:	BNE	\$8 \$14
4818		XORI	\$24	1			cmp_top_200_i		
4819	cmp_bot_203:	BNE	\$21	\$23		4877		XORI	\$15 1
	cmp_top_202					4878	cmp_top_200_i:	BNE	\$8 \$14
4820		ANDX	\$25	\$15	\$24		cmp_bot_201_i		
4821	f_top_204:	BEQ	\$25	\$0		4879		ADD	\$9 \$3
	f_bot_205					4880		ADDI	\$9 3
4822		XORI	\$26	1		4881		ADD	\$10 \$3
4823	f_bot_205:	BEQ	\$25			4882		ADDI	\$10 16
	f_top_204	~		•		4883		EXCH	\$11 \$10
4824		XOR	\$6 \$	26		4884		ADDI	\$10 -16
4825	f_bot_205_i:	BEQ	\$25			4885		SUB	\$10 \$3
1020	f_top_204_i	2	720	T -		4886		EXCH	\$13 \$9
4826	1_000_204_1	XORI	\$26	1		4887		XOR	\$12 \$13
4827	f_top_204_i:	BEQ	\$25			4888		EXCH	\$13 \$9
4021	_	היים	4 L J	γU		4888		ADDI	
4000	f_bot_205_i	A VILVA	¢ O E	¢1 E	¢21			ADDI	\$12 2 \$12 \$11
4828	amp bot 203 :	ANDX			\$24				\$12 \$11
4829	cmp_bot_203_i:	BNE	\$21	<b>4</b> ∠3		4891		ADD	\$10 \$3
4000	cmp_top_202_i	VODT	004	1		4892		ADDI	\$10 16
4830		XORI	\$24			4893		EXCH	\$11 \$10
4831	cmp_top_202_i:	BNE	\$21	\$23		4894		ADDI	\$10 -16
	cmp_bot_203_i		÷	4.0		4895		SUB	\$10 \$3
4832		ADD	\$22			4896		ADDI	\$9 -3
4833		ADDI	\$22			4897		SUB	\$9 \$3
4834		EXCH	\$23	\$22		4898		EXCH	\$14 \$12

4899		ADD	\$9 \$3	4964		EXCH	\$11 \$10
4900		ADDI	\$9 3	4965		ADDI	\$10 -16
4901		ADD	\$10 \$3	4966		SUB	\$10 \$3
4902 4903		ADDI EXCH	\$10 16 \$11 \$10	4967 4968		ADDI SUB	\$9 -4 \$9 \$3
4904		ADDI	\$10 -16	4969		ADD	\$15 \$3
4905		SUB	\$10 \$3	4970		ADDI	\$15 3
4906		SUB	\$12 \$11	4971	24	ADD	\$16 \$3
4907		ADDI	\$12 -2	4972	<b>.</b>	ADDI	\$16 16
4908		EXCH	\$13 \$9	4973		EXCH	\$17 \$16
4909		XOR	\$12 \$13	4974		ADDI	\$16 -16
4910 4911		EXCH ADD	\$13 \$9 \$10 \$3	4975 4976		SUB EXCH	\$16 \$3 \$19 \$15
4911		ADDI	\$10 \$3	4976		KOR	\$18 \$19
4913		EXCH	\$11 \$10	4978		EXCH	\$19 \$15
4914		ADDI	\$10 -16	4979	24	ADDI	\$18 2
4915		SUB	\$10 \$3	4980		ADD	\$18 \$17
4916		ADDI	\$9 -3	4981		ADD	\$16 \$3
4917		SUB ADD	\$9 \$3	4982 4983		ADDI EXCH	\$16 16 \$17 \$16
4918 4919		ADDI	\$7 \$3 \$7 11	4984		ADDI	\$16 -16
4920		EXCH	\$8 \$7	4985		SUB	\$16 \$3
4921		ADDI	\$7 -11	4986		ADDI	\$15 -3
4922		SUB	\$7 \$3	4987	s	SUB	\$15 \$3
4923	test_196:	BEQ	\$6 \$0	4988		EXCH	\$20 \$18
100.1	test_false_198	VODT	¢ ( 1	4989		ADD	\$15 \$3
4924 4925		XORI ADD	\$6 1 \$7 \$3	4990 4991		ADD ADDI	\$15 3 \$16 \$3
4926		ADDI	\$7 11	4992		ADDI	\$16 16
4927		EXCH	\$8 \$7	4993		EXCH	\$17 \$16
4928		ADDI	\$7 -11	4994	24	ADDI	\$16 -16
4929		SUB	\$7 \$3	4995		SUB	\$16 \$3
4930		ADD	\$9 \$3	4996		SUB	\$18 \$17
4931		ADDI ADD	\$9 4	4997		ADDI EXCH	\$18 -2
4932 4933		ADDI	\$10 \$3 \$10 16	4998 4999		KOR	\$19 \$15 \$18 \$19
4934		EXCH	\$11 \$10	5000		EXCH	\$19 \$15
4935		ADDI	\$10 -16	5001		ADD	\$16 \$3
4936		SUB	\$10 \$3	5002		ADDI	\$16 16
4937		EXCH	\$13 \$9	5003		EXCH	\$17 \$16
4938		XOR EXCH	\$12 \$13 \$13 \$9	5004		ADDI SUB	\$16 -16
4939 4940		ADDI	\$12 2	5005 5006		ADDI	\$16 \$3 \$15 -3
4941		ADD	\$12 \$11	5007		SUB	\$15 \$3
4942		ADD	\$10 \$3	5008	х	KOR	\$21 \$14
4943		ADDI	\$10 16	5009		SUB	\$21 \$20
4944		EXCH	\$11 \$10	5010		ADD	\$8 \$21
4945		ADDI SUB	\$10 -16 \$10 \$3	5011		ADD COP	\$21 \$20
4946 4947		ADDI	\$10 \$3 \$9 -4	5012 5013		KOR ADD	\$21 \$14 \$15 \$3
4948		SUB	\$9 \$3	5014		ADDI	\$15 3
4949		EXCH	\$14 \$12	5015		ADD	\$16 \$3
4950		ADD	\$9 \$3	5016		ADDI	\$16 16
4951		ADDI	\$9 4	5017		EXCH	\$17 \$16
4952 4953		ADD ADDI	\$10 \$3 \$10 16	5018 5019		ADDI SUB	\$16 -16 \$16 \$3
4954		EXCH	\$10 10	5020		EXCH	\$19 \$15
4955		ADDI	\$10 -16	5021		KOR	\$18 \$19
4956		SUB	\$10 \$3	5022		EXCH	\$19 \$15
4957		SUB	\$12 \$11	5023		ADDI	\$18 2
4958		ADDI	\$12 -2	5024		ADD	\$18 \$17
4959 4960		EXCH XOR	\$13 \$9 \$12 \$13	5025 5026		ADDI ADD	\$16 \$3 \$16 16
4960		EXCH	\$13 \$9	5026		EXCH	\$17 \$16
4962		ADD	\$10 \$3	5028		ADDI	\$16 -16
4963		ADDI	\$10 16	5029		SUB	\$16 \$3
				'			

F020	ADDT	¢1 E	2	500cl		ADD.	co co
5030	ADDI	\$15		5096		ADD	\$8 \$3
5031	SUB		\$3	5097		ADDI	\$8 6
5032	EXCH	\$20		5098		ADD	\$9 \$3
5033	ADD	\$15	\$3	5099		ADDI	\$9 16
5034	ADDI	\$15	3	5100		EXCH	\$10 \$9
5035	ADD	\$16	\$3	5101		ADDI	\$9 -16
5036	ADDI	\$16	16	5102		SUB	\$9 \$3
5037	EXCH	\$17	\$16	5103		EXCH	\$12 \$8
5038	ADDI	\$16	-16	5104		XOR	\$11 \$12
5039	SUB	\$16		5105		EXCH	\$12 \$8
5040	SUB	\$18		5106		ADDI	\$11 2
5041	ADDI	\$18		5107		ADDI	\$11 \$10
5042	EXCH	\$19		5108		ADD	\$9 \$3
5043	XOR	\$18		5109		ADDI	\$9 16
5044	EXCH	\$19		5110		EXCH	\$10 \$9
5045	ADD	\$16		5111		ADDI	\$9 -16
5046	ADDI	\$16	16	5112		SUB	\$9 \$3
5047	EXCH	\$17	\$16	5113		ADDI	\$8 -6
5048	ADDI	\$16	-16	5114		SUB	\$8 \$3
5049	SUB	\$16	\$3	5115		EXCH	\$13 \$11
5050	ADDI	\$15	-3	5116		ADD	\$8 \$3
5051	SUB	\$15	\$3	5117		ADDI	\$8 6
5052	ADD	\$9 \$		5118		ADD	\$9 \$3
5053	ADDI	\$9 4		5119		ADDI	\$9 16
	ADD	\$10		5120		EXCH	\$10 \$9
5054							
5055	ADDI	\$10		5121		ADDI	\$9 -16
5056	EXCH	\$11		5122		SUB	\$9 \$3
5057	ADDI	\$10		5123		SUB	\$11 \$10
5058	SUB	\$10		5124		ADDI	\$11 -2
5059	EXCH	\$13		5125		EXCH	\$12 \$8
5060	XOR	\$12	\$13	5126		XOR	\$11 \$12
5061	EXCH	\$13	\$9	5127		EXCH	\$12 \$8
5062	ADDI	\$12	2	5128		ADD	\$9 \$3
5063	ADD	\$12	\$11	5129		ADDI	\$9 16
5064	ADD	\$10	\$3	5130		EXCH	\$10 \$9
5065	ADDI	\$10		5131		ADDI	\$9 -16
5066	EXCH	\$11		5132		SUB	\$9 \$3
5067	ADDI	\$10		5133		ADDI	\$8 -6
5068	SUB	\$10		5134		SUB	\$8 \$3
5069	ADDI	\$9 -		5135		ADD	\$14 \$3
5070	SUB	\$9 \$		5136		ADDI	\$14 9
5071	EXCH	\$14		5137		EXCH	\$15 \$14
5072	ADD	\$9 \$		5138		ADDI	\$14 -9
5073	ADDI	\$9 4		5139		SUB	\$14 \$3
5074	ADD	\$10		5140	cmp_top_210:	BNE	\$13 \$15
5075	ADDI	\$10	16		cmp_bot_211		
5076	EXCH	\$11	\$10	5141		XORI	\$16 1
5077	ADDI	\$10	-16	5142	cmp_bot_211:	BNE	\$13 \$15
5078	SUB	\$10	\$3		cmp_top_210		
5079	SUB	\$12	\$11	5143	f_top_212:	BEQ	\$16 \$0
5080	ADDI	\$12			f_bot_213		
5081	EXCH	\$13		5144		XORI	\$17 1
5082	XOR	\$12		5145	f_bot_213:	BEQ	\$16 \$0
5083	EXCH	\$13		0110	f_top_212		720 70
5084	ADD	\$10		5146	1_00p_212	XOR	\$7 \$17
	ADDI			- 1	f_bot_213_i:	BEQ	
5085		\$10		5147		PFÕ	\$16 \$0
5086	EXCH	\$11			f_top_212_i		610 1
5087	ADDI	\$10		5148		XORI	\$17 1
5088	SUB	\$10		5149	f_top_212_i:	BEQ	\$16 \$0
5089	ADDI	\$9 -			f_bot_213_i		
5090	SUB	\$9 \$	3	5150	cmp_bot_211_i:	BNE	\$13 \$15
5091	ADD	\$7 \$	33		cmp_top_210_i		
5092	ADDI	\$7 1	11	5151		XORI	\$16 1
5093	EXCH	\$8 \$	37	5152	cmp_top_210_i:	BNE	\$13 \$15
5094	ADDI	\$7 -	-11		cmp_bot_211_i		
5095	SUB	\$7 \$		5153		ADD	\$14 \$3
T .							•

5154		ADDI	\$14 9	5218		ADDI	\$11 2
5155		EXCH	\$15 \$14	5219		ADD	\$11 \$10
5156		ADDI	\$14 -9	5220		ADD	\$9 \$3
5157		SUB	\$14 \$3	5221		ADDI	\$9 16
5158		ADD	\$8 \$3	5222		EXCH	\$10 \$9
5159		ADDI	\$8 6	5223		ADDI	\$9 -16
5160		ADD	\$9 \$3	5224		SUB	\$9 \$3
		ADDI				ADDI	
5161			\$9 16	5225			\$8 -6
5162		EXCH	\$10 \$9	5226		SUB	\$8 \$3
5163		ADDI	\$9 -16	5227		EXCH	\$13 \$11
5164		SUB	\$9 \$3	5228		ADD	\$8 \$3
5165		EXCH	\$12 \$8	5229		ADDI	\$8 6
5166		XOR	\$11 \$12	5230		ADD	\$9 \$3
5167		EXCH	\$12 \$8	5231		ADDI	\$9 16
5168		ADDI	\$11 2	5232		EXCH	\$10 \$9
5169		ADD	\$11 \$10	5233		ADDI	\$9 -16
5170		ADD	\$9 \$3	5234		SUB	\$9 \$3
5171		ADDI	\$9 16	5235		SUB	\$11 \$10
5172		EXCH	\$10 \$9	5236		ADDI	\$11 -2
5173		ADDI	\$9 -16	5237		EXCH	\$12 \$8
5174		SUB	\$9 \$3	5238		XOR	\$11 \$12
5175		ADDI	\$8 -6	5239		EXCH	\$12 \$8
5176		SUB	\$8 \$3	5240		ADD	\$9 \$3
5177		EXCH	\$13 \$11	5241		ADDI	\$9 16
5178		ADD	\$8 \$3	5242		EXCH	\$10 \$9
5179		ADDI	\$8 6	5243		ADDI	\$9 -16
5180		ADD	\$9 \$3	5244		SUB	\$9 \$3
5181		ADDI	\$9 16	5245		ADDI	\$8 -6
5182		EXCH	\$10 \$9	5246		SUB	\$8 \$3
5183		ADDI	\$9 -16	5247		ADD	\$14 \$3
5184		SUB	\$9 \$3	5248		ADDI	\$14 9
5185		SUB	\$11 \$10	5249		EXCH	\$15 \$14
		ADDI	\$11 -2				
5186				5250		ADDI	\$14 -9
5187		EXCH	\$12 \$8	5251		SUB	\$14 \$3
5188		XOR	\$11 \$12	5252	cmp_top_214:	BNE	\$13 \$15
5189		EXCH	\$12 \$8		cmp_bot_215		
5190		ADD	\$9 \$3	5253		XORI	\$16 1
5191		ADDI	\$9 16	5254	cmp_bot_215:	BNE	\$13 \$15
5192		EXCH	\$10 \$9		cmp_top_214		
5193		ADDI	\$9 -16	5255	f_top_216:	BEQ	\$16 \$0
5194		SUB	\$9 \$3		f_bot_217		
5195		ADDI	\$8 -6	5256		XORI	\$17 1
5196		SUB	\$8 \$3	5257	f_bot_217:	BEQ	\$16 \$0
5197	test_206:	BEQ	\$7 \$0		 f_top_216	_	
	test_false_208	_		5258		XOR	\$7 \$17
5198	0000_14100_200	XORI	\$7 1	5259	f_bot_217_i:	BEQ	\$16 \$0
5199		EXCH	\$3 \$1	0200	f_top_216_i		720 70
5200		ADDI	\$1 -1	5260		XORI	\$17 1
		BRA	l moveRight		f top 216 ;	BEQ	
5201				_9201	f_top_216_i:	₽₽₽₽	\$16 \$0
5202		ADDI	\$1 1		f_bot_217_i	DNE	610 615
5203		EXCH	\$3 \$1	5262	cmp_bot_215_i:	BNE	\$13 \$15
5204		XORI	\$7 1		cmp_top_214_i		
5205	assert_true_207:	BRA	assert_209	5263		XORI	\$16 1
5206	test_false_208:	BRA	test_206	5264	cmp_top_214_i:	BNE	\$13 \$15
5207	assert_209:	BNE	\$7 \$0		cmp_bot_215_i		
	assert_true_207			5265		ADD	\$14 \$3
5208		ADD	\$8 \$3	5266		ADDI	\$14 9
5209		ADDI	\$8 6	5267		EXCH	\$15 \$14
5210		ADD	\$9 \$3	5268		ADDI	\$14 -9
5211		ADDI	\$9 16	5269		SUB	\$14 \$3
5212		EXCH	\$10 \$9	5270		ADD	\$8 \$3
5213		ADDI	\$9 -16	5271		ADDI	\$8 6
5214		SUB	\$9 \$3	5272		ADD	\$9 \$3
5214		EXCH	\$12 \$8	5273		ADDI	\$9 16
		XOR				EXCH	
5216			\$11 \$12	5274			\$10 \$9
5217		EXCH	\$12 \$8	5275		ADDI	\$9 -16

5276	SUB	\$9 \$3	5342	I	ADDI	\$9 16
5277	EXCH	\$12 \$8	5343		EXCH	\$10 \$9
	XOR	\$11 \$12			ADDI	\$9 -16
5278	EXCH		5344		SUB	\$9 \$3
5279	ADDI	\$12 \$8	5345		ADDI	
5280		\$11 2	5346			\$8 -6
5281	ADD ADD	\$11 \$10 \$9 \$3	5347		SUB ADD	\$8 \$3 \$14 \$3
5282	ADDI	\$9 16	5348		ADDI	\$14 8
5283	EXCH	\$10 \$9	5349		EXCH	\$15 \$14
5284 5285	ADDI	\$9 -16	5350		ADDI	\$13 \$14
5286	SUB	\$9 \$3	5351 5352		SUB	\$14 \$3
5287	ADDI	\$8 -6	5353	  cmp_top_222:	BNE	\$13 \$15
5288	SUB	\$8 \$3	0000	cmp_bot_223	DILL	V13 V13
5289	EXCH	\$13 \$11	5354	Cmp_boc_223	XORI	\$16 1
5290	ADD	\$8 \$3	5355	cmp bot 223:	BNE	\$13 \$15
5291	ADDI	\$8 6	0000	cmp_top_222		710 710
5292	ADD	\$9 \$3	5356	f_top_224:	BEQ	\$16 \$0
5293	ADDI	\$9 16	0000	f_bot_225	2	720 70
5294	EXCH	\$10 \$9	5357		XORI	\$17 1
5295	ADDI	\$9 -16	5358	f_bot_225:	BEQ	\$16 \$0
5296	SUB	\$9 \$3		f_top_224	~	
5297	SUB	\$11 \$10	5359		XOR	\$7 \$17
5298	ADDI	\$11 -2	5360	f_bot_225_i:	BEQ	\$16 \$0
5299	EXCH	\$12 \$8		f_top_224_i	_	
5300	XOR	\$11 \$12	5361		XORI	\$17 1
5301	EXCH	\$12 \$8	5362	f_top_224_i:	BEQ	\$16 \$0
5302	ADD	\$9 \$3		f_bot_225_i		
5303	ADDI	\$9 16	5363	cmp_bot_223_i:	BNE	\$13 \$15
5304	EXCH	\$10 \$9		cmp_top_222_i		
5305	ADDI	\$9 -16	5364		XORI	\$16 1
5306	SUB	\$9 \$3	5365	cmp_top_222_i:	BNE	\$13 \$15
5307	ADDI	\$8 -6		cmp_bot_223_i		
5308	SUB	\$8 \$3	5366		ADD	\$14 \$3
5309	ADD	\$8 \$3	5367		ADDI	\$14 8
5310	ADDI	\$8 6	5368		EXCH	\$15 \$14
5311	ADD	\$9 \$3	5369		ADDI	\$14 -8
5312	ADDI	\$9 16	5370		SUB	\$14 \$3
5313	EXCH	\$10 \$9	5371		ADD	\$8 \$3
5314	ADDI	\$9 -16	5372		ADDI	\$8 6
5315	SUB	\$9 \$3	5373		ADD	\$9 \$3
5316	EXCH	\$12 \$8	5374		ADDI	\$9 16
5317	XOR	\$11 \$12	5375		EXCH	\$10 \$9
5318	EXCH	\$12 \$8	5376		ADDI	\$9 -16
5319	ADDI	\$11 2	5377		SUB	\$9 \$3
5320	ADD ADD	\$11 \$10 \$9 \$3	5378		EXCH	\$12 \$8
5321 5322	ADDI	\$9 \$3 \$9 16	5379 5380		XOR EXCH	\$11 \$12 \$12 \$8
5322	EXCH	\$10 \$9	5380		ADDI	\$12 \$8
5324	ADDI	\$9 -16	5382		ADDI	\$11 \$10
5325	SUB	\$9 \$3	5383		ADD	\$9 \$3
5326	ADDI	\$8 -6	5384		ADDI	\$9 16
5327	SUB	\$8 \$3	5385		EXCH	\$10 \$9
5328	EXCH	\$13 \$11	5386		ADDI	\$9 -16
5329	ADD	\$8 \$3	5387		SUB	\$9 \$3
5330	ADDI	\$8 6	5388		ADDI	\$8 -6
5331	ADD	\$9 \$3	5389		SUB	\$8 \$3
5332	ADDI	\$9 16	5390		EXCH	\$13 \$11
5333	EXCH	\$10 \$9	5391		ADD	\$8 \$3
5334	ADDI	\$9 -16	5392		ADDI	\$8 6
5335	SUB	\$9 \$3	5393		ADD	\$9 \$3
5336	SUB	\$11 \$10	5394		ADDI	\$9 16
5337	ADDI	\$11 -2	5395		EXCH	\$10 \$9
5338	EXCH	\$12 \$8	5396		ADDI	\$9 -16
5339	XOR	\$11 \$12	5397		SUB	\$9 \$3
5340	EXCH	\$12 \$8	5398		SUB	\$11 \$10
5341	ADD	\$9 \$3	5399		ADDI	\$11 -2

1					I		
5400		EXCH	\$12 \$8	5464		SUB	\$14 \$3
5401		XOR	\$11 \$12	5465	cmp_top_226:	BNE	\$13 \$15
5402		EXCH	\$12 \$8		cmp_bot_227		
5403		ADD	\$9 \$3	5466		XORI	\$16 1
5404		ADDI	\$9 16	5467	cmp_bot_227:	BNE	\$13 \$15
5405		EXCH	\$10 \$9		cmp_top_226		
5406		ADDI	\$9 -16	5468	f_top_228:	BEQ	\$16 \$0
5407		SUB	\$9 \$3		f_bot_229		
5408		ADDI	\$8 -6	5469		XORI	\$17 1
5409		SUB	\$8 \$3	5470	f_bot_229:	BEQ	\$16 \$0
5410 test_2	18:	BEQ	\$7 \$0		f_top_228	_	
	st false 220	_		5471		XOR	\$7 \$17
5411		XORI	\$7 1	5472	f_bot_229_i:	BEQ	\$16 \$0
5412		EXCH	\$3 \$1		f_top_228_i	_	
5413		ADDI	\$1 -1	5473		XORI	\$17 1
5414		RBRA	l_moveRight		f_top_228_i:	BEQ	\$16 \$0
5415		ADDI	\$1 1		f_bot_229_i	2	710 70
5416		EXCH	\$3 \$1	5475	cmp_bot_227_i:	BNE	\$13 \$15
5417		XORI	\$7 1	3413	cmp_top_226_i	DNE	V13 V13
	+ ruo 210.	BRA		E 476		XORI	\$16 1
	_true_219:		assert_221	5476	+ 226 :-		
	alse_220:	BRA	test_218	5477	cmp_top_226_i:	BNE	\$13 \$15
5420 assert		BNE	\$7 \$0		cmp_bot_227_i	300	614 62
ł	sert_true_219		40.40	5478		ADD	\$14 \$3
5421		ADD	\$8 \$3	5479		ADDI	\$14 8
5422		ADDI	\$8 6	5480		EXCH	\$15 \$14
5423		ADD	\$9 \$3	5481		ADDI	\$14 -8
5424		ADDI	\$9 16	5482		SUB	\$14 \$3
5425		EXCH	\$10 \$9	5483		ADD	\$8 \$3
5426		ADDI	\$9 -16	5484		ADDI	\$8 6
5427		SUB	\$9 \$3	5485		ADD	\$9 \$3
5428		EXCH	\$12 \$8	5486		ADDI	\$9 16
5429		XOR	\$11 \$12	5487		EXCH	\$10 \$9
5430		EXCH	\$12 \$8	5488		ADDI	\$9 -16
5431		ADDI	\$11 2	5489		SUB	\$9 \$3
5432		ADD	\$11 \$10	5490		EXCH	\$12 \$8
5433		ADD	\$9 \$3	5491		XOR	\$11 \$12
5434		ADDI	\$9 16	5492		EXCH	\$12 \$8
5435		EXCH	\$10 \$9	5493		ADDI	\$11 2
5436		ADDI	\$9 -16	5494		ADD	\$11 \$10
5437		SUB	\$9 \$3	5495		ADD	\$9 \$3
5438		ADDI	\$8 -6	5496		ADDI	\$9 16
5439		SUB	\$8 \$3	5497		EXCH	\$10 \$9
5440		EXCH	\$13 \$11	5498		ADDI	\$9 -16
5441		ADD	\$8 \$3	5499		SUB	\$9 \$3
5442		ADDI	\$8 6	5500		ADDI	\$8 -6
5443		ADD	\$9 \$3	5501		SUB	\$8 \$3
5444		ADDI	\$9 16	5502		EXCH	\$13 \$11
5445		EXCH	\$10 \$9	5503		ADD	\$8 \$3
5446		ADDI	\$9 -16	5504		ADDI	\$8 6
5447		SUB	\$9 \$3	5505		ADD	\$9 \$3
5448		SUB	\$11 \$10	5506		ADDI	\$9 16
5449		ADDI	\$11 -2	5507		EXCH	\$10 \$9
5450		EXCH	\$12 \$8	5508		ADDI	\$9 -16
5451		XOR	\$11 \$12	5509		SUB	\$9 \$3
5452		EXCH	\$12 \$8	5510		SUB	\$11 \$10
5453		ADD	\$9 \$3	5511		ADDI	\$11 -2
5454		ADDI	\$9 16	5512		EXCH	\$12 \$8
5455		EXCH	\$10 \$9	5513		XOR	\$11 \$12
5456		ADDI	\$9 -16	5514		EXCH	\$12 \$8
5457		SUB	\$9 \$3	5515		ADD	\$9 \$3
5458		ADDI	\$8 -6	5516		ADDI	\$9 16
5459		SUB	\$8 \$3	5517		EXCH	\$10 \$9
5460		ADD	\$14 \$3	5518		ADDI	\$9 -16
5461		ADDI	\$14 8	5519		SUB	\$9 \$3
5462		EXCH	\$15 \$14	5520		ADDI	\$8 -6
5463		ADDI	\$14 -8	5521		SUB	\$8 \$3
3400			711 0	0021	I	202	40 YJ

5522	XORI	\$6 1	5585		ADD	\$17 \$3
5523 assert_true_197:	BRA	assert_199	5586		ADDI	\$17 16
5524 test_false_198:	BRA	test_196	5587		EXCH	\$18 \$17
5525 assert_199:	BNE	\$6 \$0	5588		ADDI	\$17 -16
assert_true_197			5589		SUB	\$17 \$3
5526	ADD	\$7 \$3	5590		ADDI	\$16 -5
5527	ADDI	\$7 11	5591		SUB	\$16 \$3
5528	EXCH	\$8 \$7	5592		EXCH	\$21 \$19
5529	ADDI	\$7 -11	5593		ADD	\$16 \$3
5530	SUB	\$7 \$3	5594		ADDI	\$16 5
5531	ADD	\$9 \$3	5595		ADD	\$17 \$3
5532	ADDI	\$9 4	5596		ADDI	\$17 16
5533	ADD	\$10 \$3	5597		EXCH	\$18 \$17
5534	ADDI	\$10 16	5598		ADDI	\$17 -16
5535 5536	EXCH ADDI	\$11 \$10 \$10 -16	5599 5600		SUB SUB	\$17 \$3 \$19 \$18
5537	SUB	\$10 \$3	5601		ADDI	\$19 -2
5538	EXCH	\$13 \$9	5602		EXCH	\$20 \$16
5539	XOR	\$12 \$13	5603		XOR	\$19 \$20
5540	EXCH	\$13 \$9	5604		EXCH	\$20 \$16
5541	ADDI	\$12 2	5605		ADD	\$17 \$3
5542	ADD	\$12 \$11	5606		ADDI	\$17 16
5543	ADD	\$10 \$3	5607		EXCH	\$18 \$17
5544	ADDI	\$10 16	5608		ADDI	\$17 -16
5545	EXCH	\$11 \$10	5609		SUB	\$17 \$3
5546	ADDI	\$10 -16	5610		ADDI	\$16 -5
5547	SUB	\$10 \$3	5611		SUB	\$16 \$3
5548	ADDI	\$9 -4	5612		ADD	\$22 \$3
5549	SUB	\$9 \$3	5613		ADDI	\$22 7
5550	EXCH ADD	\$14 \$12 \$9 \$3	5614 5615		EXCH ADDI	\$23 \$22 \$22 -7
5551 5552	ADDI	\$9 4	5616		SUB	\$22 \$3
5553	ADD	\$10 \$3	5617	cmp_top_232:	BNE	\$21 \$23
5554	ADDI	\$10 16	5017	cmp_bot_233	DIVE	YZ1 YZ3
5555	EXCH	\$11 \$10	5618	op_200_200	XORI	\$24 1
5556	ADDI	\$10 -16	5619	cmp_bot_233:	BNE	\$21 \$23
5557	SUB	\$10 \$3		cmp_top_232		
5558	SUB	\$12 \$11	5620		ANDX	\$25 \$15 \$24
5559	ADDI	\$12 -2	5621	f_top_234:	BEQ	\$25 \$0
5560	EXCH	\$13 \$9		f_bot_235		
5561	XOR	\$12 \$13	5622		XORI	\$26 1
5562	EXCH	\$13 \$9	5623	f_bot_235:	BEQ	\$25 \$0
5563	ADD	\$10 \$3	F 00 4	f_top_234	VOD	¢C ¢2C
5564 5565	ADDI EXCH	\$10 16 \$11 \$10	5624 5625	f_bot_235_i:	XOR BEQ	\$6 \$26 \$25 \$0
5566	ADDI	\$10 -16	3023	f_top_234_i	PFÕ	723 70
5567	SUB	\$10 \$3	5626	1_00p_201_1	XORI	\$26 1
5568	ADDI	\$9 -4	5627	f_top_234_i:	BEQ	\$25 \$0
5569	SUB	\$9 \$3		f_bot_235_i	-	
5570 cmp_top_230:	BNE	\$8 \$14	5628		ANDX	\$25 \$15 \$24
cmp_bot_231			5629	cmp_bot_233_i:	BNE	\$21 \$23
5571	XORI	\$15 1		cmp_top_232_i		
5572 cmp_bot_231:	BNE	\$8 \$14	5630		XORI	\$24 1
cmp_top_230		¢16 ¢2	5631	cmp_top_232_i:	BNE	\$21 \$23
5573	ADD	\$16 \$3	F. C. C.	cmp_bot_233_i	300	600 60
5574	ADDI	\$16 5	5632		ADD	\$22 \$3
5575 5576	ADD ADDI	\$17 \$3 \$17 16	5633 5634		ADDI EXCH	\$22 7 \$23 \$22
5576 5577	EXCH	\$17 16	5635		ADDI	\$23 \$22 \$22 -7
5578	ADDI	\$17 -16	5636		SUB	\$22 \$3
5579	SUB	\$17 \$3	5637		ADD	\$16 \$3
5580	EXCH	\$20 \$16	5638		ADDI	\$16 5
5581	XOR	\$19 \$20	5639		ADD	\$17 \$3
5582	EXCH	\$20 \$16	5640		ADDI	\$17 16
5583	ADDI	\$19 2	5641		EXCH	\$18 \$17
5584	ADD	\$19 \$18	5642		ADDI	\$17 -16

5643		SUB	\$17 \$3	5707		ADDI	\$12 -2
		EXCH	\$20 \$16	5708		EXCH	\$13 \$9
5644							
5645		XOR	\$19 \$20	5709		XOR	\$12 \$13
5646		EXCH	\$20 \$16	5710		EXCH	\$13 \$9
5647		ADDI	\$19 2	5711		ADD	\$10 \$3
5648		ADD	\$19 \$18	5712		ADDI	\$10 16
5649		ADD	\$17 \$3	5713		EXCH	\$11 \$10
5650		ADDI	\$17 16	5714		ADDI	\$10 -16
5651		EXCH	\$18 \$17	5715		SUB	\$10 \$3
5652		ADDI	\$17 -16	5716		ADDI	\$9 -4
5653		SUB	\$17 \$3	5717		SUB	\$9 \$3
5654		ADDI	\$16 -5	5718		ADD	\$7 \$3
5655		SUB	\$16 \$3	5719		ADDI	\$7 11
5656		EXCH	\$21 \$19	5720		EXCH	\$8 \$7
5657		ADD	\$16 \$3	5721		ADDI	\$7 -11
5658		ADDI	\$16 5	5722		SUB	\$7 \$3
5659		ADD	\$17 \$3	5723	l_inst_6_bot:	BRA	l_inst_6_top
5660		ADDI	\$17 16	5724	l_moveRight_7_top:	BRA	
5661		EXCH	\$18 \$17		l_moveRight_7_bot		
		ADDI		E79E		ADDI	¢1 1
5662			\$17 -16	5725		ADDI	\$1 1
5663		SUB	\$17 \$3	5726		EXCH	\$2 \$1
5664		SUB	\$19 \$18	5727		EXCH	\$3 \$1
5665		ADDI	\$19 -2	5728		ADDI	\$1 -1
5666		EXCH	\$20 \$16	5729	l_moveRight_7:	SWAPBR	
5667		XOR	\$19 \$20	5730		NEG	\$2
5668		EXCH	\$20 \$16	5731		ADDI	\$1 1
5669		ADD	\$17 \$3	5732		EXCH	\$3 \$1
5670		ADDI	\$17 16	5733		EXCH	\$2 \$1
5671		EXCH	\$18 \$17	5734		ADDI	\$1 -1
					11511 200		
5672		ADDI	\$17 -16	5735	localBlock_302:	XOR	\$6 \$1
5673		SUB	\$17 \$3	5736		XOR	\$7 \$0
5674		ADDI	\$16 -5	5737		EXCH	\$7 \$1
5675		SUB	\$16 \$3	5738		ADDI	\$1 -1
	amp bot 231 i.	BNE			localBlock_301:	XOR	\$7 \$1
5676	cmp_bot_231_i:	DINE	\$8 \$14	5739	10Calblock_301:		
	cmp_top_230_i			5740		XOR	\$8 \$0
5677		XORI	\$15 1	5741		EXCH	\$8 \$1
5678	cmp_top_230_i:	BNE	\$8 \$14	5742		ADDI	\$1 -1
	cmp_bot_231_i			5743		ADD	\$8 \$3
5070	cmp_boc_231_1	NDD.	60 63				
5679		ADD	\$9 \$3	5744		ADDI	\$8 2
5680		ADDI	\$9 4	5745		EXCH	\$9 \$8
5681		ADD	\$10 \$3	5746		ADDI	\$8 -2
5682		ADDI	\$10 16	5747		SUB	\$8 \$3
5683		EXCH	\$11 \$10	5748		XOR	\$10 \$9
					loadMa+Add 226.		
5684		ADDI	\$10 -16	5749	loadMetAdd_236:	EXCH	\$11 \$10
5685		SUB	\$10 \$3	5750		ADDI	\$11 3
5686		EXCH	\$13 \$9	5751		EXCH	\$12 \$11
5687		XOR	\$12 \$13	5752		XOR	\$13 \$12
5688		EXCH	\$13 \$9	5753		EXCH	\$12 \$11
5689		ADDI	\$12 2	5754		ADDI	\$11 -3
5690		ADD	\$12 \$11	5755		EXCH	\$11 \$10
5691		ADD	\$10 \$3	5756		ADD	\$8 \$3
5692		ADDI	\$10 16	5757		ADDI	\$8 2
5693		EXCH	\$11 \$10	5758		EXCH	\$9 \$8
5694		ADDI	\$10 -16	5759		ADDI	\$8 -2
5695		SUB	\$10 \$3	5760		SUB	\$8 \$3
5696		ADDI	\$9 -4	5761		ADD	\$14 \$3
5697		SUB	\$9 \$3	5762		ADDI	\$14 14
5698		EXCH	\$14 \$12	5763		EXCH	\$3 \$1
5699		ADD	\$9 \$3	5764		ADDI	\$1 -1
5700		ADDI	\$9 4	5765		EXCH	\$7 \$1
5701		ADD	\$10 \$3	5766		ADDI	\$1 -1
5702		ADDI	\$10 16	5767		EXCH	\$6 \$1
5703		EXCH	\$11 \$10	5768		ADDI	\$1 -1
5704		ADDI	\$10 -16	5769		EXCH	\$14 \$1
5705		SUB	\$10 \$3	5770		ADDI	\$1 -1
5706		SUB	\$12 \$11	5771		EXCH	\$10 \$1

5772		ADDI	\$1 -1	5838		ADDI	\$13 5834
5773		ADDI	\$13 -5773	5839		ADDI	\$1 1
5774	l_rjmp_top_238:	RBRA	l_rjmp_bot_2	2 <b>38</b> 40		EXCH	\$10 \$1
5775	l_jmp_237:	SWAPBR		5841		ADDI	\$1 1
5776		NEG	\$13	5842		EXCH	\$6 \$1
5777	l_rjmp_bot_239:	BRA	l_rjmp_top_2			ADDI	\$1 1
5778		ADDI ADDI	\$13 5773	5844		EXCH ADDI	\$7 \$1
5779 5780		EXCH	\$1 1 \$10 \$1	5845 5846		EXCH	\$1 1 \$3 \$1
5781		ADDI	\$1 1	5847		ADD	\$8 \$3
5782		EXCH	\$14 \$1	5848		ADDI	\$8 2
5783		ADDI	\$1 1	5849		EXCH	\$9 \$8
5784		EXCH	\$6 \$1	5850		ADDI	\$8 -2
5785		ADDI	\$1 1	5851		SUB	\$8 \$3
5786		EXCH	\$7 \$1	5852		EXCH	\$11 \$10
5787		ADDI	\$1 1	5853		ADDI	\$11 1
5788		EXCH	\$3 \$1	5854		EXCH	\$12 \$11
5789		ADDI	\$14 -14	5855		XOR	\$13 \$12
5790		SUB	\$14 \$3	5856		EXCH	\$12 \$11
5791		ADD	\$8 \$3	5857	7 77 17 17 17 17	ADDI	\$11 -1
5792		ADDI	\$8 2 \$9 \$8	5858	loadMetAdd_240_i:	EXCH	\$11 \$10
5793 5794		EXCH ADDI	\$8 -2	5859 5860		XOR ADD	\$10 \$9 \$8 \$3
5795		SUB	\$8 \$3	5861		ADDI	\$8 2
5796		EXCH	\$11 \$10	5862		EXCH	\$9 \$8
5797		ADDI	\$11 3	5863		ADDI	\$8 -2
5798		EXCH	\$12 \$11	5864		SUB	\$8 \$3
5799		XOR	\$13 \$12	5865		EXCH	\$9 \$6
5800		EXCH	\$12 \$11	5866	cmp_top_246:	BNE	\$9 \$0
5801		ADDI	\$11 -3		cmp_bot_247		
5802	loadMetAdd_236_i:	EXCH	\$11 \$10	5867		XORI	\$10 1
5803		XOR	\$10 \$9	5868	cmp_bot_247:	BNE	\$9 \$0
5804		ADD	\$8 \$3		cmp_top_246		411 40
5805		ADDI	\$8 2	5869		ADD	\$11 \$3
5806 5807		EXCH ADDI	\$9 \$8 \$8 -2	5870 5871		ADDI EXCH	\$11 14 \$12 \$11
5808		SUB	\$8 \$3	5872		ADDI	\$11 -14
5809		ADD	\$8 \$3	5873		SUB	\$11 \$3
5810		ADDI	\$8 2	5874		ADD	\$13 \$3
5811		EXCH	\$9 \$8	5875		ADDI	\$13 10
5812		ADDI	\$8 -2	5876		EXCH	\$14 \$13
5813		SUB	\$8 \$3	5877		ADDI	\$13 -10
5814		XOR	\$10 \$9	5878		SUB	\$13 \$3
5815	loadMetAdd_240:	EXCH	\$11 \$10	5879	cmp_top_248:	BNE	\$12 \$14
5816		ADDI	\$11 1		cmp_bot_249		** - *
5817		EXCH	\$12 \$11	5880	1 240-	XORI	\$15 1 \$12 \$14
5818 5819		XOR EXCH	\$13 \$12 \$12 \$11	5881	*	BNE	\$1Z \$14
5820		ADDI	\$11 -1	5882	cmp_top_248	ANDX	\$16 \$10 \$15
5821		EXCH	\$11 \$10	5883	f_top_250:	BEQ	\$16 \$0
5822		ADD	\$8 \$3		f_bot_251	<b>z</b>	. =
5823		ADDI	\$8 2	5884		XORI	\$17 1
5824		EXCH	\$9 \$8	5885	f_bot_251:	BEQ	\$16 \$0
5825		ADDI	\$8 -2		f_top_250		
5826		SUB	\$8 \$3	5886		XOR	\$8 \$17
5827		EXCH	\$3 \$1	5887		BEQ	\$16 \$0
5828		ADDI	\$1 -1		f_top_250_i		A1 7 1
5829		EXCH	\$7 \$1	5888	6 + 250 '	XORI	\$17 1
5830		ADDI	\$1 -1	5889		BEQ	\$16 \$0
5831 5832		EXCH ADDI	\$6 \$1 \$1 -1	5890	f_bot_251_i	ZMDV	\$16 \$10 \$15
5832 5833		EXCH	\$10 \$1	5890	cmp_bot_249_i:	ANDX BNE	\$10 \$10 \$15
5834		ADDI	\$1 -1	5551	cmp_bot_245_1.		110 411
5835		ADDI	\$13 -5834	5892	1	XORI	\$15 1
5836	l_jmp_241:	SWAPBR		5893	cmp_top_248_i:	BNE	\$12 \$14
5837	· -	NEG	\$13		cmp_bot_249_i		
					•		

5894		ADD	610 60	5957		XORI	¢10 1
5894		ADDI	\$13 \$3 \$13 10	5958		EXCH	\$10 1 \$10 \$9
5896		EXCH	\$14 \$13	5959	obj_con_252_bot:	ADDI	\$9 -1
5897		ADDI	\$13 -10	5960	obj_con_232_bot.	EXCH	\$9 \$6
5898		SUB	\$13 \$3	5961		EXCH	\$9 \$7
5899		ADD	\$11 \$3	5962		EXCH	\$10 \$6
5900		ADDI	\$11 14	5963	copy_253:	XOR	\$9 \$10
5901		EXCH	\$12 \$11	5964	copy_233.	ADDI	\$10 1
5902		ADDI	\$11 -14	5965		EXCH	\$11 \$10
5902		SUB	\$11 \$3			ADDI	\$11 1
5903	cmp_bot_247_i:	BNE	\$9 \$0	5966 5967		EXCH	\$11 \$10
3904	cmp_bot_247_1.	DINE	79 70	5968		ADDI	\$10 -1
E00E	Cmp_cop_z40_1	XORI	\$10 1	5969		EXCH	\$10 \$6
5905 5906	amp + ap 246 ;	BNE	\$9 \$0			EXCH	\$9 \$7
5906	cmp_top_246_i:	DINE	39 30	5970		EXCH	\$9 \$6
5007	cmp_bot_247_i	EXCH	\$9 \$6	5971 5972		XOR	
5907	+og+ 242.				loadMotadd 254.		\$10 \$9
5908	test_242:	BEQ	\$8 \$0	5973	loadMetAdd_254:	EXCH	\$11 \$10
5000	test_false_244	VODT	ĊO 1	5974		ADDI	\$11 2
5909		XORI	\$8 1	5975		EXCH	\$12 \$11
5910		ADD ADDI	\$9 \$3	5976		XOR	\$13 \$12
5911			\$9 14	5977		EXCH	\$12 \$11
5912		EXCH	\$10 \$9	5978		ADDI	\$11 -2
5913		ADDI	\$9 -14	5979		EXCH	\$11 \$10
5914		SUB	\$9 \$3	5980		EXCH	\$9 \$6
5915		ADD	\$11 \$3	5981		EXCH	\$3 \$1
5916		ADDI	\$11 10	5982		ADDI	\$1 -1
5917		EXCH	\$12 \$11	5983		EXCH	\$6 \$1
5918		ADDI	\$11 -10	5984		ADDI	\$1 -1
5919		SUB	\$11 \$3	5985		EXCH	\$7 \$1
5920		XOR	\$10 \$12	5986		ADDI	\$1 -1
5921		ADD	\$11 \$3	5987		EXCH	\$10 \$1
5922		ADDI	\$11 10	5988		ADDI	\$1 -1
5923		EXCH	\$12 \$11	5989		ADDI	\$13 -5989
5924							
1		ADDI	\$11 -10	5990	1_rjmp_top_256:	RBRA	l_rjmp_bot_257
5925		SUB	\$11 \$3	5991	1_rjmp_top_256: 1_jmp_255:	SWAPBR	\$13
5925 5926		SUB ADD	\$11 \$3 \$9 \$3	5991 5992	1_jmp_255:	SWAPBR NEG	\$13 \$13
5925 5926 5927		SUB ADD ADDI	\$11 \$3 \$9 \$3 \$9 14	5991 5992 5993		SWAPBR NEG BRA	\$13 \$13 l_rjmp_top_256
5925 5926 5927 5928		SUB ADD ADDI EXCH	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9	5991 5992 5993 5994	1_jmp_255:	SWAPBR NEG BRA ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989
5925 5926 5927 5928 5929		SUB ADD ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14	5991 5992 5993 5994 5995	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1
5925 5926 5927 5928 5929 5930		SUB ADD ADDI EXCH ADDI SUB	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3	5991 5992 5993 5994 5995 5996	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1
5925 5926 5927 5928 5929 5930 5931		SUB ADD ADDI EXCH ADDI SUB EXCH	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1	5991 5992 5993 5994 5995 5996 5997	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1
5925 5926 5927 5928 5929 5930 5931 5932		SUB ADD ADDI EXCH ADDI SUB EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1	5991 5992 5993 5994 5995 5996 5997 5998	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1
5925 5926 5927 5928 5929 5930 5931 5932 5933		ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1	5991 5992 5993 5994 5995 5996 5997 5998 5999	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934		SUB ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935		SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000 6001	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936		SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000 6001 6002	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937	obj_con_252:	SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI ADDI ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000 6001 6002 6003	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$6 \$1 \$1 1 \$6 \$1 \$1 1 \$9 \$6
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938	obj_con_252:	SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000 6001 6002 6003 6004	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$1
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938 5939	obj_con_252:	SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI ADDI ADDI ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000 6001 6002 6003 6004 6005	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938 5939 5940	obj_con_252:	SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$9 \$1	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000 6001 6002 6003 6004 6005 6006	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938 5939 5940 5941	obj_con_252:	SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000 6001 6002 6003 6004 6005 6006 6007	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938 5939 5940 5941	obj_con_252:	SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$1 malloc	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000 6001 6002 6003 6004 6005 6006 6007 6008	1_jmp_255:	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938 5939 5940 5941 5942 5943	obj_con_252:	SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$1 \$	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000 6001 6002 6003 6004 6005 6006 6007 6008	<pre>1_jmp_255: 1_rjmp_bot_257:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$11 -2
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938 5939 5940 5941 5942 5943 5944	obj_con_252:	SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1	5991 5992 5993 5994 5995 5996 5997 5998 5999 6000 6001 6002 6003 6004 6005 6006 6007 6008 6009 6010	1_jmp_255:	SWAPBR NEG BRA ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$11 -2 \$11 \$10
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938 5940 5941 5942 5943 5944 5945	obj_con_252:	SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$1  5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6004 6005 6006 6007 6008 6009 6010 6011	<pre>1_jmp_255: 1_rjmp_bot_257:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH EXCH ADDI EXCH EXCH EXCH ADDI EXCH EXCH ADDI EXCH XOR	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$11 -2 \$11 \$10 \$10 \$9	
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938 5940 5941 5942 5943 5944 5945		SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$1	5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6004 6005 6006 6007 6008 6009 6010 6011 6012	<pre>1_jmp_255: 1_rjmp_bot_257:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI EXCH EXCH EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 \$2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$11 -2 \$11 \$10 \$10 \$9 \$9 \$6
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5938 5940 5941 5942 5943 5944 5945 5946 5947	<pre>obj_con_252: obj_con_252_i:</pre>	SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI BRA ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 \$14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1	5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6004 6005 6006 6007 6008 6009 6010 6011 6012 6013	<pre>1_jmp_255: 1_rjmp_bot_257:</pre>	SWAPBR NEG BRA ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH EXCH ADDI EXCH EXCH EXCH EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH EXCH EXCH	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10
5925 5926 5927 5928 5929 5930 5931 5933 5934 5935 5936 5937 5938 5939 5940 5941 5942 5943 5944 5945 5946 5947 5948		SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI BRA ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$1	5991 5992 5993 5994 5995 5996 6000 6001 6002 6003 6004 6005 6006 6007 6008 6009 6010 6011 6012 6013	<pre>1_jmp_255: 1_rjmp_bot_257:  loadMetAdd_254_i:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH XOR EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$11 -2 \$11 \$10 \$10 \$9 \$9 \$6 \$10 \$9
5925 5926 5927 5928 5929 5930 5931 5932 5933 5935 5936 5937 5938 5940 5941 5942 5943 5944 5945 5946 5947 5948		SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 1 \$1 1 \$1 2 \$1 1 \$1 2 \$1 2 \$1 3 \$1 3 \$1 3 \$1 3 \$1 4 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5 \$1 5	5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6004 6005 6006 6007 6008 6009 6010 6011 6012 6013 6014 6015	<pre>1_jmp_255: 1_rjmp_bot_257:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH XOR EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$11 -2 \$11 \$10 \$10 \$9 \$9 \$6 \$10 \$9 \$9 \$6 \$10 \$9 \$10 \$9 \$11 \$10
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5940 5941 5942 5943 5944 5945 5946 5947 5948 5949 5950		SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 \$14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1	5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6006 6007 6008 6009 6010 6011 6012 6013 6014 6015 6016	<pre>1_jmp_255: 1_rjmp_bot_257:  loadMetAdd_254_i:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH XOR EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$11 -2 \$11 \$10 \$10 \$9 \$9 \$6 \$10 \$9 \$11 \$10 \$10 \$10 \$10 \$9 \$11 \$10 \$10 \$9 \$11 \$10 \$10 \$9 \$11 \$10 \$11 \$10 \$11 \$10 \$10 \$10 \$10 \$9 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$10 \$9 \$11 \$10 \$10 \$9 \$11 \$10 \$11 \$9 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$10 \$9 \$11 \$10 \$11 \$10
5925 5926 5927 5928 5929 5930 5931 5932 5935 5936 5937 5938 5939 5940 5941 5942 5943 5944 5945 5946 5947 5948 5949 5950 5951		SUB ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 \$8 \$10 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 1 \$1	5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6006 6007 6008 6009 6010 6011 6012 6013 6014 6015 6016 6017	<pre>1_jmp_255: 1_rjmp_bot_257:  loadMetAdd_254_i:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH XOR EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH EXCH EXCH	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$11 -2 \$11 \$10 \$10 \$9 \$9 \$6 \$10 \$9 \$9 \$6 \$10 \$9 \$1 \$1 \$10 \$10 \$9 \$1 \$1 \$10 \$10 \$9 \$1 \$1 \$10 \$10 \$9 \$11 \$10 \$10 \$9 \$11 \$10 \$10 \$9 \$11 \$10 \$11 \$10 \$10 \$9 \$11 \$10 \$11 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10
5925 5926 5927 5928 5929 5930 5931 5932 5933 5936 5937 5938 5939 5940 5941 5942 5943 5944 5945 5946 5947 5948 5949 5950 5951		SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 \$14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 \$8 \$10 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 1 \$1 -1 \$1 0 \$1 \$1 1 \$1 0 \$1 \$1 1 \$1 1	5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6006 6007 6008 6009 6010 6011 6012 6013 6014 6015 6016 6017 6018	<pre>1_jmp_255: 1_rjmp_bot_257:  loadMetAdd_254_i:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI EXCH EXCH ADDI EXCH XOR EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH XOR	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$10 \$9 \$9 \$6 \$11 \$10 \$10 \$9 \$9 \$6 \$11 \$10 \$10 \$9 \$9 \$6 \$10 \$9 \$11 \$10 \$10 \$9 \$9 \$6 \$10 \$9 \$11 \$10 \$10 \$9 \$11 \$10 \$10 \$9 \$11 \$10 \$10 \$9 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$11 \$10 \$12 \$11 \$13 \$12 \$14 \$10 \$15 \$10 \$11 \$10 \$11 \$10 \$10 \$9 \$11 \$10 \$11 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5936 5937 5940 5941 5942 5943 5944 5945 5946 5947 5948 5949 5950 5951 5952 5953		SUB ADD ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 \$14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$7 \$1 \$1 -1 \$6 \$1 \$1 -1 \$10 8 \$10 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1 1 \$1	5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6004 6005 6006 6007 6008 6009 6010 6011 6012 6013 6014 6015 6016 6017 6018	<pre>1_jmp_255: 1_rjmp_bot_257:  loadMetAdd_254_i:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH ADDI EXCH ADDI EXCH XOR EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH EXCH EXCH	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$10 \$9 \$9 \$6 \$11 \$10 \$11 0 \$10 \$9 \$9 \$6 \$11 \$10 \$11 0 \$10 \$9 \$9 \$6 \$11 \$10 \$10 \$10 \$11 \$10 \$12 \$11 \$13 \$12 \$14 \$10 \$15 \$10 \$11 \$10 \$10 \$9 \$11 \$10 \$11 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10
5925 5926 5927 5928 5929 5930 5931 5932 5933 5936 5936 5937 5938 5940 5941 5942 5943 5944 5945 5946 5947 5948 5949 5950 5951 5952 5953 5954		SUB ADD ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 \$14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$6 \$1 \$1 -1 \$6 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$1 \$1 \$1 \$1	5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6004 6007 6008 6009 6010 6011 6012 6013 6014 6015 6016 6017 6018	<pre>1_jmp_255: 1_rjmp_bot_257:  loadMetAdd_254_i:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH EXCH ADDI EXCH XOR EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$10 \$9 \$9 \$6 \$11 \$10 \$11 9 \$10 \$9 \$9 \$6 \$11 \$10 \$10 \$9 \$9 \$6 \$10 \$9 \$11 \$10 \$11 \$10 \$12 \$11 \$13 \$12 \$14 \$10 \$15 \$10 \$10 \$9 \$11 \$10 \$11 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10
5925 5926 5927 5928 5929 5930 5931 5932 5933 5934 5935 5940 5941 5942 5943 5944 5945 5946 5947 5948 5949 5950 5951 5952 5953		SUB ADD ADDI EXCH ADDI SUB EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH	\$11 \$3 \$9 \$3 \$9 \$14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$6 \$1 \$1 -1 \$6 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 1 \$1 0 8 \$10 \$1 \$1 1 \$1 1 \$1 0 8 \$10 \$1 \$1 1 \$1	5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6004 6005 6006 6007 6008 6009 6010 6011 6012 6013 6014 6015 6016 6017 6018 6019 6020 6021	<pre>1_jmp_255: 1_rjmp_bot_257:  loadMetAdd_254_i:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH ADDI EXCH EXCH EXCH EXCH EXCH EXCH EXCH EXCH	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$6 \$1 \$1 1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$10 \$9 \$9 \$6 \$11 \$10 \$11 0 \$10 \$9 \$9 \$6 \$11 \$10 \$11 0 \$12 \$11 \$11 0 \$12 \$11 \$13 \$12 \$14 \$10 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$15 \$1
5925 5926 5927 5928 5929 5930 5931 5932 5933 5936 5936 5937 5938 5940 5941 5942 5943 5944 5945 5946 5947 5948 5949 5950 5951 5952 5953 5954		SUB ADD ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$11 \$3 \$9 \$3 \$9 \$14 \$10 \$9 \$9 -14 \$9 \$3 \$3 \$1 \$1 -1 \$6 \$1 \$1 -1 \$6 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$9 \$1 \$1 -1 \$1 \$1 \$1 \$1	5991 5992 5993 5994 5995 5996 5997 5998 6000 6001 6002 6003 6004 6007 6008 6009 6010 6011 6012 6013 6014 6015 6016 6017 6018	<pre>1_jmp_255: 1_rjmp_bot_257:  loadMetAdd_254_i:</pre>	SWAPBR NEG BRA ADDI ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH EXCH EXCH EXCH EXCH ADDI EXCH XOR EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI EXCH ADDI	\$13 \$13 1_rjmp_top_256 \$13 5989 \$1 1 \$10 \$1 \$1 1 \$7 \$1 \$1 1 \$6 \$1 \$1 1 \$6 \$1 \$1 1 \$3 \$1 \$9 \$6 \$11 \$10 \$11 2 \$12 \$11 \$13 \$12 \$12 \$11 \$13 \$12 \$12 \$11 \$10 \$9 \$9 \$6 \$11 \$10 \$11 9 \$10 \$9 \$9 \$6 \$11 \$10 \$10 \$9 \$9 \$6 \$10 \$9 \$11 \$10 \$11 \$10 \$12 \$11 \$13 \$12 \$14 \$10 \$15 \$10 \$10 \$9 \$11 \$10 \$11 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10 \$10

1			*** **		1		***
6023		ADD	\$14 \$3	6089		EXCH	\$11 \$10
6024		ADDI	\$14 2	6090		EXCH	\$9 \$6
6025		EXCH	\$3 \$1	6091		EXCH	\$3 \$1
6026		ADDI	\$1 -1	6092		ADDI	\$1 -1
6027		EXCH	\$7 \$1	6093		EXCH	\$6 \$1
6028		ADDI	\$1 -1	6094		ADDI	\$1 -1
6029		EXCH	\$6 \$1	6095		EXCH	\$7 \$1
6030		ADDI	\$1 -1	6096		ADDI	\$1 -1
6031		EXCH	\$14 \$1	6097		EXCH	\$10 \$1
6032		ADDI	\$1 -1	6098		ADDI	\$1 -1
6033		EXCH	\$10 \$1	6099		ADDI	\$13 -6098
6034		ADDI	\$1 -1	6100	1_jmp_264:	SWAPBR	\$13
					1_Jmp_204.		
6035		ADDI	\$13 -6035	6101		NEG	\$13
6036	l_rjmp_top_260:	RBRA	1_rjmp_bot_2	6102		ADDI	\$13 6098
6037	1_jmp_259:	SWAPBR	\$13	6103		ADDI	\$1 1
6038		NEG	\$13	6104		EXCH	\$10 \$1
	1 mimm bot 261.					ADDI	
6039	l_rjmp_bot_261:	BRA	l_rjmp_top_2	c owo			\$1 1
6040		ADDI	\$13 6035	6106		EXCH	\$7 \$1
6041		ADDI	\$1 1	6107		ADDI	\$1 1
1							
6042		EXCH	\$10 \$1	6108		EXCH	\$6 \$1
6043		ADDI	\$1 1	6109		ADDI	\$1 1
6044		EXCH	\$14 \$1	6110		EXCH	\$3 \$1
6045		ADDI	\$1 1	6111		EXCH	\$9 \$6
6046		EXCH	\$6 \$1	6112		EXCH	\$11 \$10
6047		ADDI	\$1 1	6113		ADDI	\$11 0
6048		EXCH	\$7 \$1	6114		EXCH	\$12 \$11
1							
6049		ADDI	\$1 1	6115		XOR	\$13 \$12
6050		EXCH	\$3 \$1	6116		EXCH	\$12 \$11
6051		ADDI	\$14 -2	6117		ADDI	\$11 0
6052		SUB	\$14 \$3	6118	loadMetAdd_263_i:	EXCH	\$11 \$10
1		EXCH			<u>-</u>	XOR	
6053			\$9 \$6	6119			\$10 \$9
6054		EXCH	\$11 \$10	6120		EXCH	\$9 \$6
6055		ADDI	\$11 0	6121		ADD	\$9 \$3
6056		EXCH	\$12 \$11	6122		ADDI	\$9 2
6057		XOR	\$13 \$12	6123		EXCH	\$10 \$9
1							
6058		EXCH	\$12 \$11	6124		ADDI	\$9 -2
6059		ADDI	\$11 0	6125		SUB	\$9 \$3
	1						
6060	loadMetAdd_258_i:	EXCH	\$11 \$10	6126		EXCH	\$11 \$7
6061		XOR	\$10 \$9	6127	uncopy_265:	XOR	\$10 \$11
6062		EXCH	\$9 \$6	6128		ADDI	\$11 1
6063		EXCH	\$9 \$6	6129		EXCH	\$12 \$11
6064		ADD	\$10 \$3	6130		ADDI	\$12 -1
1							
6065		ADDI	\$10 2	6131		EXCH	\$12 \$11
6066		EXCH	\$11 \$10	6132		ADDI	\$11 -1
6067		ADDI	\$10 -2	6122		EXCH	\$11 \$7
6067				6133			\$11 \$7
6068		SUB	\$10 \$3	6134		ADD	\$9 \$3
6069	swap_262:	XOR	\$9 \$11	6135		ADDI	\$9 2
6070		XOR	\$11 \$9	6136		EXCH	\$10 \$9
6071		XOR	\$9 \$11	6137		ADDI	\$9 -2
6072		ADD	\$10 \$3	6138		SUB	\$9 \$3
6073		ADDI	\$10 2	6139		ADD	\$10 \$3
6074		EXCH	\$11 \$10	6140		ADDI	\$10 2
6075		ADDI	\$10 -2	6141		EXCH	\$11 \$10
6076		SUB	\$10 \$3	6142		ADDI	\$10 -2
6077		EXCH	\$9 \$6	6143		SUB	\$10 \$3
6078		XORI	\$8 1	6144	cmp_top_270:	BNE	\$11 \$0
6079	assert_true_243:	BRA	assert_245		cmp_bot_271		
				61.45		VODT	610 1
6080	test_false_244:	BRA	test_242	6145		XORI	\$12 1
6081		EXCH	\$9 \$6	6146	cmp_bot_271:	BNE	\$11 \$0
6082		XOR	\$10 \$9		cmp_top_270		
					Cmp_cop_z / o		*** 0 ***
6083	loadMetAdd_263:	EXCH	\$11 \$10	6147		ADD	\$13 \$3
6084		ADDI	\$11 0	6148		ADDI	\$13 14
6085		EXCH	\$12 \$11	6149		EXCH	\$14 \$13
6086		XOR	\$13 \$12	6150		ADDI	\$13 -14
6087		EXCH	\$12 \$11	6151		SUB	\$13 \$3
6088		ADDI	\$11 0	6152		ADD	\$15 \$3

1					1		
6153		ADDI	\$15 10	6208		EXCH	\$6 \$1
6154		EXCH	\$16 \$15	6209		ADDI	\$1 -1
6155		ADDI	\$15 -10	6210		EXCH	\$15 \$1
6156		SUB	\$15 \$3	6211		ADDI	\$1 -1
6157	cmp_top_272:	BNE	\$14 \$16	6212		EXCH	\$11 \$1
	cmp_bot_273			6213		ADDI	\$1 -1
6158		XORI	\$17 1	6214		ADDI	\$14 -6213
6159	cmp_bot_273:	BNE	\$14 \$16	6215	l_jmp_277:	SWAPBR	
	cmp_top_272		1 1	6216		NEG	\$14
6160	Cmp_cop_2 / 2	ANDX	\$18 \$12 \$17			ADDI	\$14 6213
1	f + on 274.		\$18 \$0			ADDI	\$1 1
6161	f_top_274:	BEQ	3TO 30	6218			
	f_bot_275		***	6219		EXCH	\$11 \$1
6162		XORI	\$19 1	6220		ADDI	\$1 1
6163	f_bot_275:	BEQ	\$18 \$0	6221		EXCH	\$15 \$1
	f_top_274			6222		ADDI	\$1 1
6164		XOR	\$9 \$19	6223		EXCH	\$6 \$1
6165	f_bot_275_i:	BEQ	\$18 \$0	6224		ADDI	\$1 1
	f_top_274_i			6225		EXCH	\$7 \$1
6166		XORI	\$19 1	6226		ADDI	\$1 1
6167	f_top_274_i:	BEQ	\$18 \$0	6227		EXCH	\$3 \$1
	f_bot_275_i	_		6228		ADDI	\$15 -2
6168	1_200_1 / 0_1	ANDX	\$18 \$12 \$17			SUB	\$15 \$3
6169	cmp_bot_273_i:	BNE	\$14 \$16	6230		EXCH	\$10 \$7
0109	cmp_top_272_i	DNE	AT4 AT0				
	Cmp_cop_z /z_1	WORT	617 1	6231		EXCH	\$12 \$11
6170	. 070	XORI	\$17 1	6232		ADDI	\$12 2
6171	cmp_top_272_i:	BNE	\$14 \$16	6233		EXCH	\$13 \$12
	cmp_bot_273_i			6234		XOR	\$14 \$13
6172		ADD	\$15 \$3	6235		EXCH	\$13 \$12
6173		ADDI	\$15 10	6236		ADDI	\$12 -2
6174		EXCH	\$16 \$15	6237	loadMetAdd_276_i:	EXCH	\$12 \$11
6175		ADDI	\$15 -10	6238		XOR	\$11 \$10
6176		SUB	\$15 \$3	6239		EXCH	\$10 \$7
6177		ADD	\$13 \$3	6240		ADD	\$10 \$3
6178		ADDI	\$13 14	6241		ADDI	\$10 2
6179		EXCH	\$14 \$13	6242		EXCH	\$11 \$10
6180		ADDI	\$13 -14	6243		ADDI	\$10 -2
6181		SUB	\$13 \$3	6244		SUB	\$10 \$3
6182	cmp_bot_271_i:	BNE	\$11 \$0	6245		EXCH	\$12 \$7
0102	_	DINE	AII AO		   un conv. 270.		
	cmp_top_270_i	WODT	A10 1	6246	uncopy_278:	XOR	\$11 \$12
6183		XORI	\$12 1	6247		ADDI	\$12 1
6184	cmp_top_270_i:	BNE	\$11 \$0	6248		EXCH	\$13 \$12
	cmp_bot_271_i			6249		ADDI	\$13 -1
6185		ADD	\$10 \$3	6250		EXCH	\$13 \$12
6186		ADDI	\$10 2	6251		ADDI	\$12 -1
6187		EXCH	\$11 \$10	6252		EXCH	\$12 \$7
6188		ADDI	\$10 -2	6253		ADD	\$10 \$3
6189		SUB	\$10 \$3	6254		ADDI	\$10 2
6190	test_266:	BEQ	\$9 \$0	6255		EXCH	\$11 \$10
	test_false_268			6256		ADDI	\$10 -2
6191		XORI	\$9 1	6257		SUB	\$10 \$3
6192		EXCH	\$10 \$7	6258		EXCH	\$10 \$7
6193		XOR	\$11 \$10	6259	obj_des_279_top:	EXCH	\$11 \$10
6194	loadMetAdd_276:	EXCH	\$12 \$11	6260	2	XORI	\$11 10
6195	ioaaneenaa_z / o:	ADDI	\$12 2	6261		ADDI	\$10 1
6196		EXCH XOR	\$13 \$12	6262		EXCH XORI	\$11 \$10 \$11 1
6197			\$14 \$13	6263			
6198		EXCH	\$13 \$12	6264		ADDI	\$10 -1
6199		ADDI	\$12 -2	6265		EXCH	\$3 \$1
6200		EXCH	\$12 \$11	6266		ADDI	\$1 -1
6201		EXCH	\$10 \$7	6267		EXCH	\$7 \$1
6202		ADD	\$15 \$3	6268		ADDI	\$1 -1
6203		ADDI	\$15 2	6269		EXCH	\$6 \$1
6204		EXCH	\$3 \$1	6270		ADDI	\$1 -1
6205		ADDI	\$1 -1	6271	obj_des_279:	ADDI	\$11 8
6206		EXCH	\$7 \$1	6272		EXCH	\$11 \$1
6207		ADDI	\$1 -1	6273		ADDI	\$1 -1
'					'		

		<b>511011</b>	010 01		1	<b></b>	¢11 ¢10
6274		EXCH	\$10 \$1	6331	loadMetAdd_284:	EXCH	\$11 \$10
6275		ADDI	\$1 -1	6332		ADDI	\$11 0
6276		RBRA	l_malloc	6333		EXCH	\$12 \$11
6277		ADDI	\$1 1	6334		XOR	\$13 \$12
6278		EXCH	\$10 \$1	6335		EXCH	\$12 \$11
6279		ADDI	\$1 1	6336		ADDI	\$11 0
6280		EXCH	\$11 \$1	6337		EXCH	\$11 \$10
	obj_des_279_i:	ADDI	\$11 -8			EXCH	
6281	obj_des_2/9_1.			6338			\$9 \$6
6282		ADDI	\$1 1	6339		EXCH	\$3 \$1
6283		EXCH	\$6 \$1	6340		ADDI	\$1 -1
6284		ADDI	\$1 1	6341		EXCH	\$6 \$1
						ADDI	
6285		EXCH	\$7 \$1	6342			\$1 -1
6286		ADDI	\$1 1	6343		EXCH	\$7 \$1
6287		EXCH	\$3 \$1	6344		ADDI	\$1 -1
6288		EXCH	\$10 \$7	6345		EXCH	\$10 \$1
6289		ADD	\$10 \$3	6346		ADDI	\$1 -1
6290		ADDI	\$10 14	6347		ADDI	\$13 -6347
6291		EXCH	\$11 \$10	6348	l_rjmp_top_286:	RBRA	l_rjmp_bot_287
6292		ADDI	\$10 -14	6349	1_jmp_285:	SWAPBR	
				- 1			
6293		SUB	\$10 \$3	6350		NEG	\$13
6294		ADD	\$12 \$3	6351	l_rjmp_bot_287:	BRA	l_rjmp_top_286
6295		ADDI	\$12 10	6352		ADDI	\$13 6347
1		EXCH				ADDI	\$1 1
6296			\$13 \$12	6353			
6297		ADDI	\$12 -10	6354		EXCH	\$10 \$1
6298		SUB	\$12 \$3	6355		ADDI	\$1 1
6299		XOR	\$11 \$13	6356		EXCH	\$7 \$1
1							
6300		ADD	\$12 \$3	6357		ADDI	\$1 1
6301		ADDI	\$12 10	6358		EXCH	\$6 \$1
6302		EXCH	\$13 \$12	6359		ADDI	\$1 1
6303		ADDI	\$12 -10	6360		EXCH	\$3 \$1
1							
6304		SUB	\$12 \$3	6361		EXCH	\$9 \$6
6305		ADD	\$10 \$3	6362		EXCH	\$11 \$10
6306		ADDI	\$10 14	6363		ADDI	\$11 0
6307		EXCH	\$11 \$10	6364		EXCH	\$12 \$11
6308		ADDI	\$10 -14	6365		XOR	\$13 \$12
6309		SUB	\$10 \$3	6366		EXCH	\$12 \$11
6310		XORI	\$9 1	6367		ADDI	\$11 0
	2000t + mile 267.				londMo+7dd 204 i.	EXCH	\$11 \$10
6311	assert_true_267:	BRA	assert_269	6368	loadMetAdd_284_i:		
6312	test_false_268:	BRA	test_266	6369		XOR	\$10 \$9
6313	assert_269:	BNE	\$9 \$0	6370		EXCH	\$9 \$6
	assert_true_267			6371		ADD	\$9 \$3
0014	dbbcrc_crdc_207	EVOII	¢10 ¢7				
6314		EXCH	\$10 \$7	6372		ADDI	\$9 2
6315	cmp_top_280:	BNE	\$10 \$0	6373		EXCH	\$10 \$9
	cmp_bot_281			6374		ADDI	\$9 -2
6316		XORI	\$11 1	6375		SUB	\$9 \$3
	b-+ 201.						
6317	cmp_bot_281:	BNE	\$10 \$0	6376	0.00	EXCH	\$11 \$6
	cmp_top_280			6377	swap_288:	XOR	\$10 \$11
6318	f_top_282:	BEQ	\$11 \$0	6378		XOR	\$11 \$10
	f_bot_283			6379		XOR	\$10 \$11
6319		XORI	\$12 1	6380		EXCH	\$11 \$6
	£ 1-1 202						
6320	f_bot_283:	BEQ	\$11 \$0	6381		ADD	\$9 \$3
	f_top_282			6382		ADDI	\$9 2
6321		XOR	\$9 \$12	6383		EXCH	\$10 \$9
6322	f_bot_283_i:	BEQ	\$11 \$0	6384		ADDI	\$9 -2
0322		DEG	711 70				
	f_top_282_i			6385		SUB	\$9 \$3
6323		XORI	\$12 1	6386		ADD	\$9 \$3
6324	f_top_282_i:	BEQ	\$11 \$0	6387		ADDI	\$9 2
	f_bot_283_i	~		6388		EXCH	\$10 \$9
			610 60				
6325	cmp_bot_281_i:	BNE	\$10 \$0	6389		ADDI	\$9 -2
	cmp_top_280_i			6390		SUB	\$9 \$3
6326		XORI	\$11 1	6391		XOR	\$11 \$10
6327	cmp_top_280_i:	BNE	\$10 \$0		loadMetAdd_289:	EXCH	\$12 \$11
0327		DRE	4 T O 4 O	6392	TOURING CAUL_209.		
	cmp_bot_281_i			6393		ADDI	\$12 1
6328		EXCH	\$10 \$7	6394		EXCH	\$13 \$12
6329		EXCH	\$9 \$6	6395		XOR	\$14 \$13
6330			\$10 \$9	6396		EXCH	\$13 \$12
0330		XOR	Ψ10 ΨJ	0330			710 712

6207	7	ADDT	610 1	cacal		ADDT	ė1 1
6397		ADDI	\$12 -1	6463		ADDI	\$1 -1
6398		EXCH	\$12 \$11	6464		EXCH	\$7 \$1
6399		ADD	\$9 \$3	6465		ADDI	\$1 -1
6400		ADDI	\$9 2	6466		EXCH	\$6 \$1
6401		EXCH	\$10 \$9	6467		ADDI	\$1 -1
6402		ADDI	\$9 -2	6468		EXCH	\$15 \$1
6403		SUB	\$9 \$3	6469		ADDI	\$1 -1
6404		EXCH	\$3 \$1	6470		EXCH	\$11 \$1
6405		ADDI	\$1 -1	6471		ADDI	\$1 -1
6406		EXCH	\$7 \$1	6472		ADDI	\$14 -6471
6407	1	ADDI	\$1 -1	6473	l_jmp_292:	SWAPBR	\$14
6408		EXCH	\$6 \$1	6474		NEG	\$14
6409	1	ADDI	\$1 -1	6475		ADDI	\$14 6471
6410	I	EXCH	\$11 \$1	6476		ADDI	\$1 1
6411	1	ADDI	\$1 -1	6477		EXCH	\$11 \$1
6412	1	ADDI	\$14 -6411	6478		ADDI	\$1 1
6413	l_jmp_290:	SWAPBR	\$14	6479		EXCH	\$15 \$1
6414	ı	NEG	\$14	6480		ADDI	\$1 1
6415	1	ADDI	\$14 6411	6481		EXCH	\$6 \$1
6416	1	ADDI	\$1 1	6482		ADDI	\$1 1
6417	I	EXCH	\$11 \$1	6483		EXCH	\$7 \$1
6418	1	ADDI	\$1 1	6484		ADDI	\$1 1
6419	I	EXCH	\$6 \$1	6485		EXCH	\$3 \$1
6420	1	ADDI	\$1 1	6486		ADDI	\$15 -14
6421	I	EXCH	\$7 \$1	6487		SUB	\$15 \$3
6422	1	ADDI	\$1 1	6488		ADD	\$9 \$3
6423	I	EXCH	\$3 \$1	6489		ADDI	\$9 2
6424	1	ADD	\$9 \$3	6490		EXCH	\$10 \$9
6425	1	ADDI	\$9 2	6491		ADDI	\$9 -2
6426	I	EXCH	\$10 \$9	6492		SUB	\$9 \$3
6427	1	ADDI	\$9 -2	6493		EXCH	\$12 \$11
6428	5	SUB	\$9 \$3	6494		ADDI	\$12 3
6429	I	EXCH	\$12 \$11	6495		EXCH	\$13 \$12
6430		ADDI	\$12 1	6496		XOR	\$14 \$13
6431		EXCH	\$13 \$12	6497		EXCH	\$13 \$12
6432		KOR	\$14 \$13	6498		ADDI	\$12 -3
6433		EXCH	\$13 \$12	6499	loadMetAdd_291_i:	EXCH	\$12 \$11
6434		ADDI	\$12 -1	6500		XOR	\$11 \$10
6435		EXCH	\$12 \$11	6501		ADD	\$9 \$3
6436		KOR	\$11 \$10	6502		ADDI	\$9 2
6437		ADD	\$9 \$3	6503		EXCH	\$10 \$9
6438		ADDI	\$9 2	6504		ADDI	\$9 -2
6439		EXCH	\$10 \$9	6505		SUB	\$9 \$3
6440		ADDI	\$9 -2	6506	assert_245:	BNE	\$8 \$0
6441		SUB	\$9 \$3		assert_true_243		00.00
6442		ADD	\$9 \$3	6507	202	EXCH	\$9 \$6
6443		ADDI	\$9 2	6508		BNE	\$9 \$0
6444		EXCH	\$10 \$9		cmp_bot_294	WORT	610 1
6445		ADDI	\$9 -2	6509	204	XORI	\$10 1
6446		SUB KOR	\$9 \$3	6510	cmp_bot_294:	BNE	\$9 \$0
6447			\$11 \$10	6511	cmp_top_293	DEO.	¢10 ¢0
6448	_	EXCH ADDI	\$12 \$11	6511	f_top_295: f_bot_296	BEQ	\$10 \$0
6449 6450		EXCH	\$12 3 \$13 \$12	6510	1_000_296	XORI	\$11 1
6451		KOR	\$14 \$13	6512 6513	f_bot_296:	BEQ	\$10 \$0
6452		EXCH	\$13 \$12	0013	f_top_295	DEQ	710 70
6453		ADDI	\$12 -3	6514	1_cop_233	XOR	\$8 \$11
6454		EXCH	\$12 \$11		f_bot_296_i:	BEQ	\$10 \$0
6455		ADD	\$9 \$3	0010	f_top_295_i	2	710 70
6456		ADDI	\$9 2	6516		XORI	\$11 1
6457		EXCH	\$10 \$9	6517	f_top_295_i:	BEQ	\$10 \$0
6458		ADDI	\$9 -2		f_bot_296_i	z	+ -
6459		SUB	\$9 \$3	6518	cmp_bot_294_i:	BNE	\$9 \$0
6460		ADD	\$15 \$3		cmp_top_293_i	•	
6461		ADDI	\$15 14	6519	<b></b>	XORI	\$10 1
6462		EXCH	\$3 \$1		cmp_top_293_i:	BNE	\$9 \$0
1				'			

	cmp_bot_294_i			6586		XOR	\$7 \$0
6521		EXCH	\$9 \$6	6587	localBlock_302_i:	XOR	\$6 \$1
6522		ADD	\$8 \$3	6588	l_moveRight_7_bot:	BRA	
6523		ADDI	\$8 2		l_moveRight_7_top		
6524		EXCH	\$9 \$8	6589	l_main_0_top:	BRA	l_main_0_bot
6525 6526		ADDI SUB	\$8 -2 \$8 \$3	6590 6591		ADDI EXCH	\$1 1 \$2 \$1
6527		XOR	\$10 \$9	6592		EXCH	\$3 \$1
6528	loadMetAdd_297:	EXCH	\$11 \$10	6593		ADDI	\$1 -1
6529	100001001100_23 / •	ADDI	\$11 1	6594	l_main_0:	SWAPBR	
6530		EXCH	\$12 \$11	6595		NEG	\$2
6531		XOR	\$13 \$12	6596		ADDI	\$1 1
6532		EXCH	\$12 \$11	6597		EXCH	\$3 \$1
6533		ADDI	\$11 -1	6598		EXCH	\$2 \$1
6534		EXCH	\$11 \$10	6599		ADDI	\$1 -1
6535		ADD	\$8 \$3	6600		EXCH	\$3 \$1
6536		ADDI	\$8 2	6601	.1	ADDI	\$1 -1
6537 6538		EXCH ADDI	\$9 \$8 \$8 -2	6602 6603	obj_con_303:	ADDI EXCH	\$8 32 \$8 \$1
6539		SUB	\$8 \$3	6604		ADDI	\$1 -1
6540		EXCH	\$3 \$1	6605		EXCH	\$7 \$1
6541		ADDI	\$1 -1	6606		ADDI	\$1 -1
6542		EXCH	\$7 \$1	6607		BRA	l_malloc
6543		ADDI	\$1 -1	6608		ADDI	\$1 1
6544		EXCH	\$6 \$1	6609		EXCH	\$7 \$1
6545		ADDI	\$1 -1	6610		ADDI	\$1 1
6546		EXCH	\$10 \$1	6611		EXCH	\$8 \$1
6547		ADDI	\$1 -1	6612	obj_con_303_i:	ADDI	\$8 -32
6548	1	ADDI	\$13 -6548	6613		ADDI	\$1 1
6549	l_rjmp_top_299:	RBRA SWAPBR	l_rjmp_bot_3			EXCH ADD	\$3 \$1
6550 6551	1_jmp_298:	NEG	\$13	6615 6616		ADDI	\$6 \$3 \$6 2
6552	1_rjmp_bot_300:	BRA	l_rjmp_top_2			XORI	\$8 3
6553	<u></u>	ADDI	\$13 6548	6618		EXCH	\$8 \$7
6554		ADDI	\$1 1	6619		ADDI	\$7 1
6555		EXCH	\$10 \$1	6620		XORI	\$8 1
6556		ADDI	\$1 1	6621		EXCH	\$8 \$7
6557		EXCH	\$6 \$1	6622	obj_con_303_bot:	ADDI	\$7 -1
6558		ADDI	\$1 1	6623		EXCH	\$7 \$6
6559		EXCH	\$7 \$1	6624		ADDI	\$6 -2
6560		ADDI	\$1 1	6625		SUB	\$6 \$3
6561 6562		EXCH ADD	\$3 \$1 \$8 \$3	6626 6627		ADD ADDI	\$6 \$3 \$6 2
6563		ADDI	\$8 2	6628		EXCH	\$7 \$6
6564		EXCH	\$9 \$8	6629		ADDI	\$6 -2
6565		ADDI	\$8 -2	6630		SUB	\$6 \$3
6566		SUB	\$8 \$3	6631		XOR	\$8 \$7
6567		EXCH	\$11 \$10	6632	loadMetAdd_304:	EXCH	\$9 \$8
6568		ADDI	\$11 1	6633		ADDI	\$9 3
6569		EXCH	\$12 \$11	6634		EXCH	\$10 \$9
6570		XOR	\$13 \$12	6635		XOR	\$11 \$10
6571		EXCH ADDI	\$12 \$11	6636		EXCH ADDI	\$10 \$9 \$9 -3
6572 6573	loadMetAdd_297_i:	EXCH	\$11 -1 \$11 \$10	6637 6638		EXCH	\$9 \$8
6574	10dd1ee11dd_257_1.	XOR	\$10 \$9	6639		ADD	\$6 \$3
6575		ADD	\$8 \$3	6640		ADDI	\$6 2
6576		ADDI	\$8 2	6641		EXCH	\$7 \$6
6577		EXCH	\$9 \$8	6642		ADDI	\$6 -2
6578		ADDI	\$8 -2	6643		SUB	\$6 \$3
6579		SUB	\$8 \$3	6644		EXCH	\$3 \$1
6580		ADDI	\$1 1	6645		ADDI	\$1 -1
6581		EXCH	\$8 \$1	6646		EXCH	\$8 \$1
6582	localBlock 301 :.	XOR XOR	\$8 \$0 \$7 \$1	6647 6648		ADDI ADDI	\$1 -1 \$11 -6647
6583 6584	localBlock_301_i:	ADDI	\$7 \$1 \$1 1	6649	1_jmp_305:	SWAPBR	
6585		EXCH	\$7 \$1	6650	<u>P</u> 000 •	NEG	\$11
						-	•

6651	AD	DDI	\$11 6647	6684		ADDI	\$4	1
6652	AD	DDI	\$1 1	6685		ADDI	\$4	-10
6653	EX	KCH	\$8 \$1	6686		XOR	\$1	\$5
6654	AD	DDI	\$1 1	6687		ADDI	\$1	2048
6655	EX	KCH	\$3 \$1	6688		ADDI	\$1	-4
6656	AD	DD	\$6 \$3	6689		XOR	\$3	\$1
6657	AD	DDI	\$6 2	6690		XORI	\$6	2
6658	EX	KCH	\$7 \$6	6691		EXCH	\$6	\$3
6659	AD	DDI	\$6 -2	6692		ADDI	\$1	-1
6660	SU	JB	\$6 \$3	6693		EXCH	\$3	\$1
6661	EX	KCH	\$9 \$8	6694		ADDI	\$1	-1
6662	AD	DDI	\$9 3	6695		BRA	1_r	main_0
6663	EX	KCH	\$10 \$9	6696		ADDI	\$1	
6664	xc	OR	\$11 \$10	6697		EXCH	\$3	\$1
6665	EX	KCH	\$10 \$9	6698		ADDI	\$3	1
6666	AD	DDI	\$9 -3	6699		ADDI	\$3	1
6667	loadMetAdd_304_i: EX	KCH	\$9 \$8	6700		EXCH	\$6	\$3
6668	xc	OR	\$8 \$7	6701		XORI	\$7	1
6669	AD	סס	\$6 \$3	6702		EXCH	\$6	\$7
6670	AD	DDI	\$6 2	6703		XORI	\$7	1
6671	EX	KCH	\$7 \$6	6704		ADDI	\$3	
6672	AD	DDI	\$6 -2	6705		ADDI	\$3	-1
6673	SU	JB	\$6 \$3	6706		ADDI	\$1	1
6674	<pre>l_main_0_bot:</pre> <pre>BR</pre>	RA.	l_main_0_top	6707		EXCH	\$6	
6675	start: BR		top	6708		XORI	\$6	
6676		<b>TART</b>		6709		XOR	\$3	\$1
6677	AD	DDI		6710		ADDI	\$1	4
6678	XC	OR	\$5 \$4	6711		ADDI	\$1	-2048
6679	AD	DDI	\$5 10	6712		XOR	\$1	\$5
6680	xc			6713		ADDI		-10
6681		DDI		6714		XOR	\$5	\$4
6682	AD	DDI		6715		ADDI	\$4	-6715
6683	EX	KCH	\$7 \$4	6716	finish:	FINISH		