UNI-corn

A hardware description language for the software programmer

1. Introduction

Hardware description languages (HDLs) are powerful tools used by computer engineers and architects to abstract increasingly complex digital logic systems. Since the 1980's HDLs have allowed engineers to produce digital circuit designs in more efficient ways both to test outputs of large, interdependent designs and to visualize them using computer aided design (CAD). **UNI-corn** hopes to add to that body of work, focusing on the computer science student.

Many academic institutions offering bachelor's degrees in computer science often initiate students with an introduction to the field along with some focus on software programming. For many computer science students, especially those with little to no prior engineering experience, the study of computer architecture can be one of the more challenging aspects of a curriculum. Great tools such as the graphical digital design application—Logisim—and HDLs like System Verilog and VHDL exist to aid students in constructing the digital logic and microarchitectures learned in a typical introductory computer architecture class. However, one of the key challenges that a student might experience with learning something like Verilog or VHDL is the simple problem of unfamiliarity. With many introductory programming courses focusing on languages like Java, C++, and Python, learning the (admittedly necessarily) unique syntax of Verilog or VHDL can add to the mounting concepts for a student to master. UNI-corn aims to solve for that by using Java-like syntax. Emphasis on *aims.*

A key caveat to this experiment is that the topic of "how to make intrinsically challenging science, technology, engineering, and mathematics (STEM) concepts easier" is fraught with conflicting data and viewpoints. UNI-corn does not presume to solve for this issue. It's simply in a syntax with which a student who's taken a previous programming course may already be familiar.

That said, UNI-corn functions like other HDLs, allows programmers to build digital circuits to test their outputs on a given clock cycle. Using the basic building building blocks of **logic gates**, UNI-corn can be used to build combinational and sequential logic designs such as different types of counters, adders, finite state machines, etc. These can be organized into modules that contain input(s) and output(s) to produce more complex designs.

2. Language Tutorial

2.1. Compiler Setup

The language compiler is developed in OCaml, which needs to be installed on the machine used to run the UNI-corn program. You can install OCaml through OPAM (OCaml Package Manager). The easiest way to install OCaml on most systems is through a package management system such as Homebrew, `apt-get`, or `dpkg`. On Homebrew, for example, you can run the following commands:

```
$ brew install ocaml
$ brew install opam
$ opam init
```

You'll also need to install the OCaml LLVM library, which can also be installed via OPAM.

```
$ opam install llvm.3.6
```

Here you must verify that the version of LLVM that OPAM installs is the same as the version that's installed via `brew`. You'll also need to take note of where brew places its executables. The PATH should match something like the one below, depending on your machine's setup:

```
$ /usr/local/Cellar/llvm/7.0.0/bin/llc
```

2.2. Compiler File Structure

The UNI-corn compiler comes in the `unicorn.tar.gz` tarball. Decompressing it gives the compiler directory structure below. The compiler files are each explained in Section 6 : Architectural Design. For now, it suffices to know two things:

1. Programs should be run from the top-level `/unicorn/` directory
2. There are test files under `/unicorn/tests` that can be used for testing

```
/unicorn/
|-- compiler/
|   |-- scanner.ml
|   |-- parser.mly
|   |-- ast.ml
|   |-- modfill.ml
|   |-- harden.ml
|   |-- semant.ml
|   |-- elaborate.ml
|   |-- indexing.ml
|   |-- noloop.ml
|   |-- simplelines.ml
|   |-- topsort.ml
|   |-- unic.ml
|   |-- codegen.ml
|-- tests/
|   |-- hello_world.uni
|   |-- comments/
|   |   |-- test1.uni
|   |
|   |-- buses/
|   |   |-- test1.uni
|   |
< other test cases continue >
|-- make.sh
|-- run.sh
|-- README.txt
```

2.3. Using the Compiler

UNI-corn files have a `.uni` extension as their file names. In this example, we will be compiling a file called `half_adder.uni`, assuming you're running this from the `/unicorn/` directory. The code snippet is below, just for reference. Do not worry about understanding the code right now. There is a run-through of the program in Subsection 2.4 and there is an in-depth description of the program in Section 3. It suffices to know that the expected value to print out is the $101 \oplus 000 = 101$.

```
// Code for a half adder
halfAdder(a<3>, b<3>) {
    sum = a xor b;
    carry = a and b;

    out: sum<3>, carry<3>;
}

main() {
    print ha: halfAdder(100b, 001b)[0];
    out:;
}

```

```
// example c file
void tick();

int main (){
    tick();
}
```

1. First run a shell script to compile all the OCaml files that will then be used to compile UNI-corn:

   ```
   $ ./make.sh
   ```

2. To compile the .uni file, run the run.sh shell script like so:

   ```
   $ ./run.sh half_adder cfile.c
   >> 101
   ```

   Note that you should omit the file extensions when using this script

2.3.1. Running with flags

You can access information about each stage of the compilation process by running the following command, replacing `-flag` with one of the options for the table below. Remember to take note of where the `unic.native` file is placed during compilation.

```
$ ./unic.native -flag filename.uni
```

- `-a` **ast**: prints the abstract syntax tree (AST) for the program
- `-m` **modfilled ast**: prints a AST with modules expanded
- `-h` **hardened ast**: AST with variables replaced by values
- `-s` **semantic ast**: semantically-checked AST (same as hast)
- `-n` **netlist:** prints netlist (i.e. unordered list of gates)
- `-sl` **netlist** (**simplified**): prints netlist with simplified code
- `-i` **indices:** prints netlist with expanded indices (i.e. values)
- `-n2` **netlist** (**more simplified**): netlist with simplified
- `-t` **netlist** (**topsorted**): topologically sorted netlist
- `-io` **netlist** (**IO**): topologically sorted netlist with IO values
- `-l` **llvm ir:** prints the LLVM IR code
- `-c` **default**: checks and prints the generated LLVM IR

2.4. Your first UNI-corn program: `half_adder.uni`

Let's revisit the small simple half adder program from above. A "Hello World" program is typically used to introduce the very basic components of a programming language. Since the purpose of an HDL is to create a programming abstraction of digital circuit logic, the small, simple half adder seems like an appropriate equivalent. Hopefully, as is one of the primary intents of UNI-corn, the syntax will look familiar to anyone who has coded in a program like Java or C++.

2.4.1. UNI-corn program building blocks

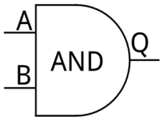
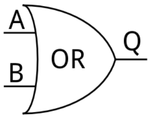
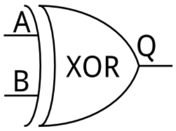
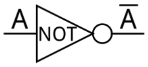
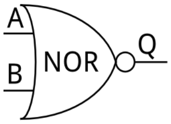
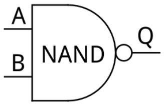
At the highest level, a UNI-corn program consists of the following key components. For the Java or C++ programmer, respective analogies have been drawn.

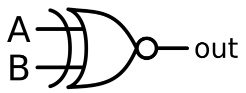
- **Comments, Expressions, and Blocks**: Comments are set the same way as in Java or C using two backslashes (//). Like these two languages, expressions end in semicolons.

- **Data Types**: As with the underpinnings of computer hardware—the very thing this language is used to describe—UNI-corn has but a single data type (get the name now?)—binary numbers. These are typed as any sequence of ones and zeros followed immediately by a `b`. The language of this type can be described by the following regular expression $(0 \mid 1)^{*}b$ and the following binary strings are in this language: `1b`, `0b`, `10b`, `11b`, `100b`, `101b`, `111b`, etc.

  Integers do exists but their use is restricted as magic numbers for use in defining the size of a bus, dereferencing a bus, or to specify the iterations of a loop.

- **Operators**: The most intrinsic building blocks of digital circuit logic and sole operators behind UNI-corn are the digital logic gates below:

Gate	Syntax
	`q = a and b;`
	`q = a or b;`
	`q = a xor b`
	`a0 = not a;`
	`q = a nor b;`
	`q = a nand b;`

	`out = a xnor b;`

Any other gates and logic can be built using a combination of the gates above.

- **Bus:** A bus is the only kind of variable that can be created. Since there is only one data type, all bus variables are of type binary string. These can be set using the equals sign (=) to a binary string literal, another bus, or an expression that evaluates to a binary string. The names are alphanumeric (further naming rules can be found section X of the language reference manual below). In the half adder code snippet above, `a` and `b` are buses that are the inputs to the `halfAdder()` module and `sum` and `carry` are buses that are its outputs.

  Because all buses are of type binary string, there's no need to specify the type like in other languages. However, you do need to specify the size of the bus when assigning a bus as input or output. This is done using angle brackets (<**`int`**>). More on this in section X of the language reference manual.

- **Modules**: These are akin to functions or methods. In digital circuit terms, this is a self-contained subcircuit, potentially with inputs and outputs, that can be reused throughout a larger circuit. In the half adder program above, a module named `halfAdder()` is defined in lines 1 through 7 and then invoked in line 10. This subcircuit has two inputs—`a` and `b`—that it then manipulates with logic gates to produce two outputs—`sum` and `carry`. The code of a module is enclosed in curly brackets.

  - **Inputs**: Inputs to a module are optional. Any inputs needed can be specified in the same way that you specify parameters to a function in Java or C. Like in these just-mentioned languages, the values of inputs are set during the module invocation.

  - **Outputs:** Outputs are **not** optional. Each and every module must have an output, even if that output is **`null`**. Outputs can be either 1 (which can **`null`**) or more. Outputs are specified using the **out** keyword followed by a colon and then either nothing, a binary string, or a bus. In the half adder example, the `halfAdder()` module has two outputs and the main() module has one output—**null**—which is set by leaving the space between the colon and the end of the expression (the semicolon) empty.

    - **main**(): A special kind of module that is required in the program. All other modules must be invoked here for them to be evaluated. The `main()` module also received ticks from the clock (more on this later).

  The final structure of a module ends up looking something like:

```
name(in0<K>,..inL<K>) { expr0;...;exprN; out:M; }
```

  where $K,M \in \mathbb{Z}^+$ and $N,L \in \mathbb{Z}$.

- **Print**: A special function used to print to stdout. This is one aspect of UNI-corn that has no Java or C equivalent and whose syntax is slightly more like a functional programming language. The function `print` is followed by a parameter (separated by a space) to which the value of the expression on the right hand side of the colon is assigned. `print` then outputs the value of the parameter to `stdout`. In the half adder program above, the parameter is printed is named `ha` and its value is the first output of the `halfAdder()` module (as denoted by the dereferencing operation `[0]`...more on this later).

- **neigh!**: Because the developers of UNI-corn are a silly bunch, all UNI-corn programs require a completely arbitrary end of file terminator (EOFT). This can either be a unicorn emoji (   ) that comes with the latest Unicode standards or simple the keyword **`neigh!`**.

That's it! With perhaps the exception of dereferencing one or more of the outputs of a module, this quick run through should provide you with everything you need to understand and run the `halfAdder()` example or create a small program of your own. Again, a more in-depth rule of the aspects of a language can be found in the Language Reference Manual below.

3. Language Reference Manual

UNI-corn is a simple hardware description language. Using the basic building blocks of combinational logic, UNI-corn can be used to build more complex digital circuits such as different types of counters, adders, mealy state machines, and many other common circuits. As is customary with digital circuits, modules—a collection of one or more gates with input and output—lie at the core of UNI-corn. Furthermore, like with a digital circuit board, sequence of operations are not determined by the code's sequence in a text file but rather by the direction and flow of the circuit's buses (either within a module or from module to module). UNI-corn's sole data types are the rise (1) and the fall (0) of electricity flowing through each bus. As discussed further in the the next chapter, these binary values can be represented and introduced into your program via a couple of different forms. As an introduction, below is a simple program that will perform all the basic logic operations on a pair of buses with constant values. A file need only have the extension .uni and, because UNI-corn has no classes, every file must have a main() method (more on this later) wherein the modules can run based on the system's clock.

3.1. Program Structure

Program Structure Programs in UNI-corn consist of a main() block and modules. Modules define circuits that can be arbitrarily recreated within other circuits, given some input. The main block will simply contain calls upon a module (or set of modules) that the programmer wants to simulate. The compiler converts the main into a function called tick() that can be linked with .c files. Its inputs and outputs can be accessed through global variables.

3.2. Data Types

UNI-corn is peculiar in that it only has one data type, but it also has arbitrarily many data types, from a different perspective. These are buses. Because there is only one data type, identifiers do not need to be preceded by a type. UNI-corn is a statically inferred language.

3.3. Bus

A bus is conceptually similar to an array of booleans represented in binary, but is represented in code as a string. It is the UNI-corn representation of physical buses. A bus's length acts essentially as a data type; type-checking happens entirely on a bus-length basis. In this sense there are arbitrarily many data types (if we consider each bus-length a distinct type).

3.3.1. **Bus Creation**

There are two main ways of initializing buses: 1) passing a constant and, 2) passing values from other modules. Buses can also be initialized with a binary expression. Once a bus is assigned, it can't be reassigned. Bus names must contain only alphanumeric English characters or the single quote ‘ (no other special symbols like $, #, !,_ etc.). They must begin with an English letter.

bus:

literal
identifier
expression

3.3.2. **Bus Value Definition**

Buses must have a value assigned at creation. The instantiation and assignment begins with the variable name and the assignment to the bus, followed by a lowercase letter ’b’. The bus size is automatically determined by the compiler. Note that leading 0's are factored into the bus's length.

3.3.3. **Bus Assignment**

Buses can have only one series of boolean values, but it has to be followed by the letter b. (i.e. 10001b would be a series of boolean values followed by the letter b)

3.3.4. **Binary Expressions**

All logical UNI-corn expressions are considered binary expressions and eventually evaluated to a binary literal, but can be made up of numerous logic gate operators and identifiers. All expressions are left-right associative and must exist only on the right side of a bus or register definition. NOTE: Assignments, module calls, print function calls, and loops are also considered expressions.

BinExpr:

literal
BinExpr BinaryOp BinExpr
UnaryOp BinExpr
(BinExpr)
identifier
BinExpr[IntExpr]
Assignments
ModuleCalls
Loops

UnaryOp:

not BinExpr
Not literal

3.3.5. Integer Expressions

Integer Expressions and integers exist in UNI-corn but are not allowed to be assigned. They are used solely for indexing and generics purposes. An integer expression can be a numerical value, an identifier, or an infix-style expression with either ‘+’ or ‘-’ as an operator between two operands.

Int Expr:

IntExpr + IntExpr
IntExpr - IntExpr
literal
identifier

3.3.6. Indexing

The square bracket ‘[ ]’ is used to index a bus whose value is a series of binary values. For a n-bit bus (i.e. of length n) the 0th index will be the most significant bit and the n-1th index will be the least significant bit. The user can have infinitely many indices. The user can specify a range while indexing.

Index:

identifier[integer]
identifier[IntExpr]
identifier[IntExpr : IntExpr]

3.3.7. **Indexing Modules**

For modules that return multiple outputs, the value assigned to a bus must be one and only one of those outputs. This output is specified by square brackets [ ] enclosing the variable name of the output bus as defined in the module. If a particular bit in an output bus is desired, double brackets [ ][ ] may be used. In the case where there is only 1 output, brackets may be omitted.

3.4. Register

Registers are built-in types comprised of combinational logic. As such their internal behavior is like modules in that they have input and output buses. However, since they have persistent memory through each cycle, they need to be specified separately. Registers can be assigned a binary string, a bus variable, or an expression of arbitrary size. The binary string value of the initialized value can be of arbitrary size. A register is identified by the assignment symbol := which is followed by the bus name to which the register is assigned. Finally, the user must specify the initial state of the bus. In order to initialize the value of the register, the user must enclose the initial value (either 0 or 1) in between asterisks, *. Use := to initialize given a bus or constant value on the right of the operator.

register:

*BinExpr := BinExpr * BoolVal **

3.5. Lexical Conventions

A UNI-corn program is broken down and parsed into tokens through lexical transformations. Tokens can be keywords, identifiers, operators, whitespace, assignment symbols, indexing symbols, punctuation, and various brackets. There are no constants in UNI-corn with the exception of boolean 1 or 0.

3.5.1. **Keywords**

In addition to all logic gate names (see 3.5.5), the following are reserved keywords and are not to be used as identifiers or otherwise:

```
from        print
out         main
for         to
make        neigh!
```

3.5.2. **Identifiers**

Identifiers are used to name variables and user-built modules. An identifier is a sequence of letters and digits that must start with a lowercase letter. Uppercase and lowercase letters are considered different characters. Identifiers can only be established during variable or module declarations. Identifiers of different types must have unique names(i.e.there cannot be a bus and a module with the same name).

assign statement:
    identifier = BinExpr;
    identifier = BooleanLiteral;

3.5.3. **Comments**

Comments are characters delineated by a specific combination of symbols and contain characters that are not executed by the program. There are two types of comments, Single-line and Multi-line comments.

- Single-line Comments Single-line comments are introduced with // and everything following the double slashes is ignored by the compiler until a newline

- Multi-line Comments Multi-line comments are introduced with /** and terminated with **/ and everything in

3.5.4. **Whitespace**

Blanks, tabs, and newlines are ignored, except to separate tokens

3.5.5. **Logical Operators**

Logical operators must evaluate to boolean expressions and return true or false in the form of ‘0’ or ‘1’

- **and** is an operator which takes in two single-bit buses, inclusive of the index of an n-bit bus, and performs the boolean logical $\wedge$ operator on them.
- **or** is an operator which takes in two single-bit buses, inclusive of the index of an n-bit bus, and performs the boolean logical $\vee$ operator on them.

- **nor** is an operator which takes in two single-bit buses, inclusive of the index of an n-bit bus, and performs the boolean logical ↓ operator on them.
- **not** is an operator which takes in a single bit and negates it. You can also index a bus to get a single bit then negate it. The **not** operator comes before the variable name.
- **nand** is an operator which takes in two single-bit buses, inclusive of the index of an n-bit bus, and performs the boolean logical ↑ operator on them.
- **xor** is an operator which takes in two single-bit buses, inclusive of the index of an n-bit bus, and performs the boolean logical ⊕ operator on them.
- **xnor** is an operator which takes in two single-bit buses, inclusive of the index of an n-bit bus, and performs the boolean logical ¬⊕ operator on them.

3.6. Syntactic Sugar for Logical Operators

`*=`	`and`	`a *= b;`
`+=`	`or`	`a += b;`
`#=`	`xor`	`a #= b;`
`!+`	`nor`	`a !+ b;`
`!*`	`nand`	`a !* b;`
`!#`	`xnor`	`a *# b;`
`!`	`not`	`! a;`

3.7. Punctuation

UNI-corn code contains three forms of punctuation, the semicolon, the comma, and and EOF terminator.

- **Semicolon:** In UNI-Corn the semicolon is used to denote statement separation. As a statement separator, the semicolon is used to demarcate boundaries between two separate statements. NOTE: putting a semicolon on a separate line does nothing. The compiler will discard a semicolon that is placed on a line by itself without any statement before it.

- **Comma:** The comma is used to denote assignment to multiple variables.
- **EOF:** The unicorn emoji is a file terminator that denotes the end of the program. The user can also type in the word ”neigh!” instead of placing the unicorn emoji at the end of the file in order to denote the end of the program.

3.8. Modules

Modules are named functions that are defined by the user. They may accept a number of parameters, perform a logical operation, and designate a number of outputs using the **out** keyword. Modules can be “called” as long as the call contains the same number of arguments as parameters in its definition.

module declarations:

identifier (params) { statement list }

params:

```
/* nothing */
identifier
identifier<IntExpr>
```

module calls:

identifier (actuals list)

actuals list:

```
/* nothing */
Bus
BoolVal
```

3.8.1. Input

Modules accept only bus types as parameters or Boolean values. Modules should not mutate their input. Parameters must use angled brackets containing the size of the bus.

3.8.2. Output

The keyword **out** is utilized in order to retrieve the most recent call to an instantiated module. There can be multiple outputs from a module. Every module, even main, need to have an out statement at the end even if it's not outputting anything. Parameters specified as output also must specify size using angled brackets.

3.9. Module Automation

3.9.1. Counted Loops

In order to avoid the tedium of copy-paste, modules may contain loops. NOTE: The user must place a semicolon after the for loop's body. Loops take the form of the keyword **for** followed by parens () containing the iterator identifier, the optional starting keyword **from** followed by its value, and the keyword **to** followed by its value. In general, loops execute the contained code, starting with initializing the variable (typically, i) to the from value (defaults to 0 if the from is omitted) creating the buses/modules listed within the block of the loop in each step.

3.9.2. Generics

Generics are used in the parameter declarations when defining a module. They are effectively used as placeholders to allow a buses size to be defined later.

3.10. Scope

A module only has access to its inputs and buses declared within its brackets. The only exception is that the main() method has access to everything within that file. Any module can call any other module (as long as there is no recursive definition), but a module can't access another module's buses, other than its output.

3.11. Standard Library

The standard library contains modules that are included in UNIcorn as built-in modules. Currently the only built-in module is print.

3.11.1. Print

The print module will make use of the reserved word, print, and will print the current value of a bus supplied as argument. Following the **print** keyword, the user defines a variable name to which the printed item is assigned. One print statement is used per value. To print multiple values, you must use multiple print statements.

3.12. Grammar

```
Program:
          module calls

module calls:
          identifier (actuals list)

          actuals list:
                    /* nothing */
                    Bus
                    BoolVal

module declarations:
          identifier (params) { statement list }

          params:
                    /* nothing */
                    identifier
                    identifier<IntExpr>

variable declarations:
          identifier = BinExpr;

assign statement:
          identifier = BinExpr;
          identifier = BooleanLiteral;

Register:
          BinExpr := BinExpr * boolVal *

Index:
          identifier[integer]
          identifier[IntExpr]
          identifier[IntExpr : IntExpr]
```

```
Int Expr:
        IntExpr + IntExpr
        IntExpr - IntExpr
        literal
        identifier

BinExpr:
        literal
        BinExpr BinaryOp BinExpr
        UnaryOp BinExpr
        (BinExpr)
        identifier
        BinExpr[IntExpr]
        Assignments
        module calls
        loops

UnaryOp:
        not BinExpr
        not literal

bus:
        literal
        identifier
        BinExpr

literal:
        stringBoolLiteral
        integer
```

4. Project Plan

4.1. The Plan

Below is the what, who, and when of how the UNI-corn compiler go developed. Descriptions of what each column means as follows:

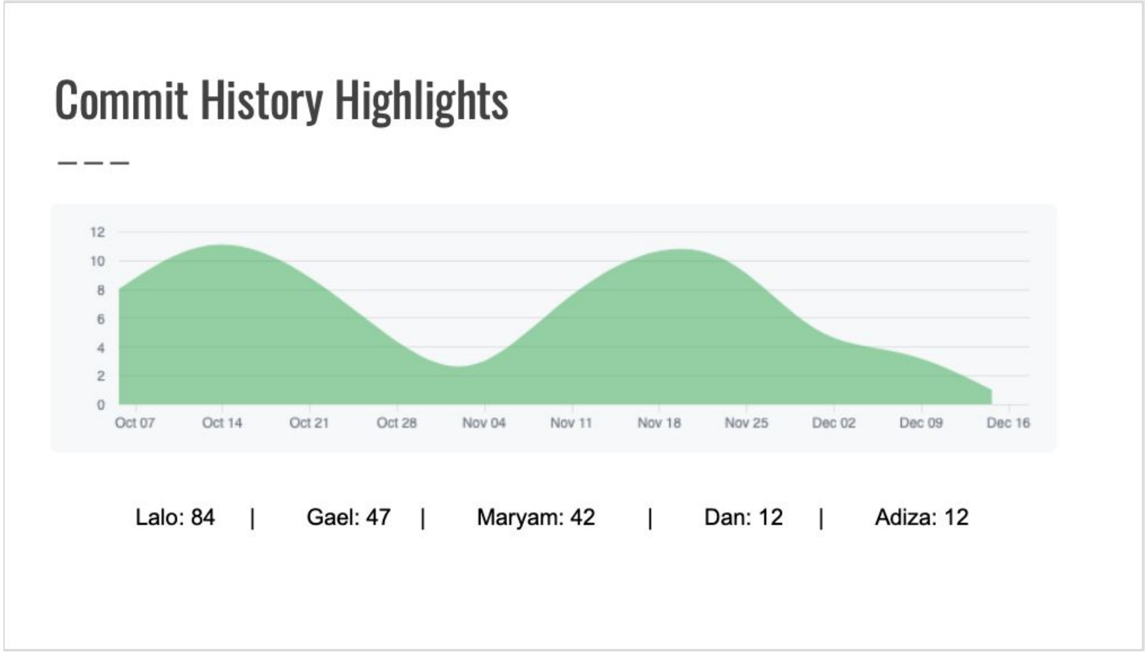
Deliverable: Mainly the actual code file that would enable a feature under the Features column.

Lead: Architect and director for how to code and structure that file. Contributed significantly to that file and thus that feature set.

Contributor: Contributed substantially to the codebase for that file, under direction of the Lead. Assisted Lead in testing, researching solutions to a coding problem, etc.

Collaborator: Was physically (or live virtually) present during the coding of the file. Contributed to the thought process or administrative tasks required to deliver.

4.2. Git Info



Lalo: 84 | Gael: 47 | Maryam: 42 | Dan: 12 | Adiza: 12

4. Language Evolution

UNI-corn was originally conceived as an alternative for the programmatically-inclined to better grasp the concepts of electrical engineering with a focus on creating simulations of circuits using a language designed to emulate Java syntax. Since Java is a relatively beginner friendly programming language and is well-known, this would allow UNI-corn to be an ideal entry point for electrical engineering for those who already have some programming background and learn best using a programmatic approach. Visualization tools for digital circuit design can be cumbersome and require users to learn a new application. With UNI-corn, was designed to resemble popular language syntax while remaining clear and powerful enough to simulate a large number of combinational and sequential logic circuits.

As we set out to materialize our vision of UNI-corn, we began by conceptualizing what our syntax would look like. The first iteration of UNI-corn syntax was roughly based on the Logisim visualization tool (a familiar tool to our team). Because this is a hardware descriptive language, we began by creating simple sequential circuits and translating them into C code. Once they were in C, we adapted and stripped the syntax to best fit with the vocabulary of circuit design. UNI-corn was originally overly verbose with lots of keywords in an attempt to help programmers new to circuit design better understand the details. Currently, UNI-corn features a streamlined syntax, a small set of keywords, and loosely resembles the familiar structure of an Object-Oriented language. At its core, UNI-corn marries the simplicity of a ‘building block’ system (consisting of module definitions, module calls, scoping rules, and a main function) with the non-sequential, input/output control flow of circuit design visualization software.

4.1. The ‘bus’ Data Type

In addition to integer and boolean data types, we included a special data type called ‘wire’ which was an array of boolean values. This ‘wire’ data type went through many iterations, but the boolean array(now known as ‘bus’) remains as one of the core details of UNI-corn. As our team progressed and better understood the project, we decided that the user didn’t need to define integer data types and single boolean values could be rolled into our ‘bus’ data type. In its current state, UNI-corn contains only the bus data type. While integers exist in UNI-corn code, they are used sparingly for indexing and generics and are not valid variable values.

4.2. Registers

Registers existed in the earliest iterations of UNI-corn however, the syntax was drastically different. Faced with challenges in the Scanner and Parser regarding left-right associativity, register syntax was altered from a right-associate to a more familiar and simple left-associative design.

4.3. Input and Output

Inputs and output lines were designed to closely resemble to vocabulary and conceptuality of wires acting as inputs and outputs to logic gates and circuitry in general. Inputs were originally expressed as an explicit line of code within a module, but have since been moved to match Java's syntax of parameter variables defined in the function definition. Therefore, module inputs are defined when the module is defined. Outputs remained functionally the same throughout the design process. Additionally, an output line is required in the main module in order to compile. This is because the main module is treated like any other module except that it is required for a valid UNI-corn file.

4.4. Logical Operators

Logic gates were a simple implementation with the only significant evolution being a transition from prefix to infix notation.

4.5. Iterations of sequential circuit design

Version 1: Overly verbose syntax, prefix expressions, and module blocks

```
module seqCirc() {

    //Variable A of type register with two inputs (input, state)
    wire a = val X;
    wire b = val Y;
    wire c = XOR(a,b);
    wire d = NOT(b);

    register init0 X := c;
    register init1 Y := d;

    out:c;
    out:d;

}

Main() {

    seqCirc();
    wire z = seqCirc()[c];
```

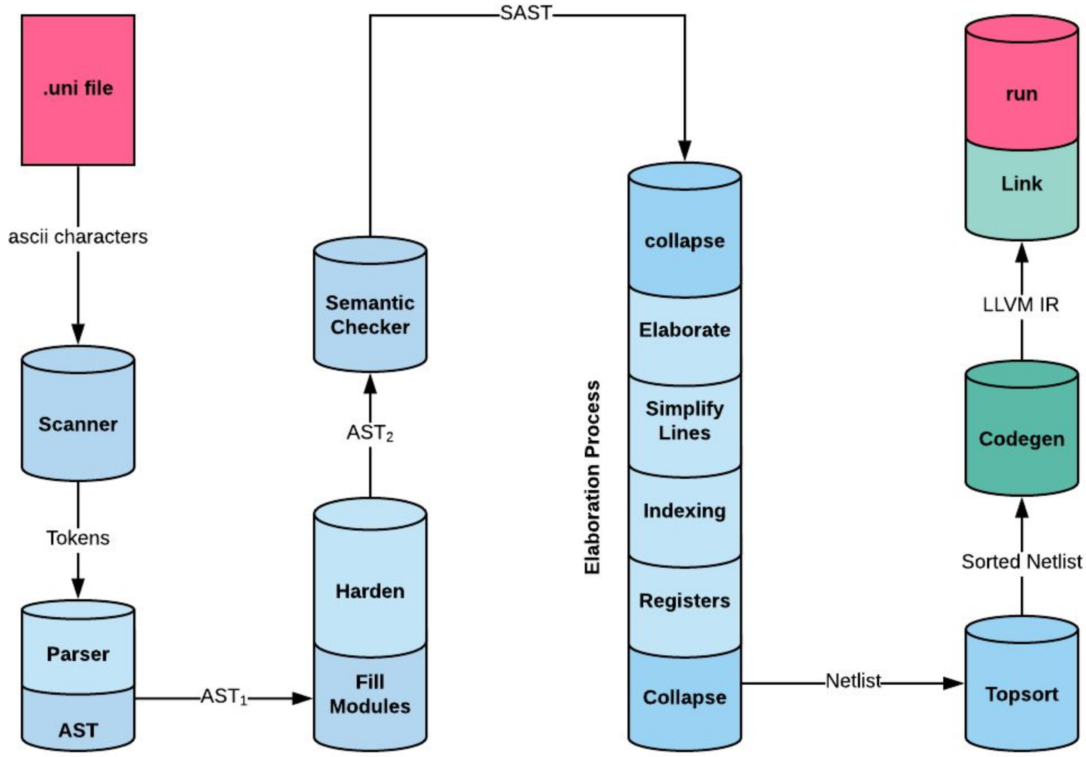
Version 2: removal of data type and module initialization keywords, use of module parameters as circuit inputs, generics, indexing, integer expressions in indexing, loops, infix logical expressions

```
main(a[5], b[5]) {
    rippleAdder(a, b);
}

fullAdder(a[0], b[0], carryIn[0]){
    x = a xor b;
    y = a and b;
    z = carryIn and x;
    sum = x xor carryIn;
    carryOut = y or z;
}

 rippleAdder(a<N>, b<N>){
    zero = 0;
    sum[0] = fullAdder(a[0], b[0], zero)[sum];

    for (i from 1 to N){
        lastCarry[i] = fullAdder
            (a[i-1], b[i-1], lastCarry[i-1])[carryOut];
        sum[i] = fullAdder(a[i], b[i], lastCarry[I])[sum];
    }

}
```

MVP: use of carryIn parameter tick function, standard library print function from module output, bus variables now ending with ‘b’, nested binary expressions, integer expressions in for loop.

```
main() {
      a = 1011010b;
      b = 0011101b;
      m = modA(a,b)[sum];
      print m: m;
      out:;
}

fA(a, b, carryIn){
      axb = a xor b;
    sum = axb xor carryIn;
    carry = (axb and carryIn) or (a and b);
      out: sum, carry;
}

modA(a<n>,b<n>){
      c[0] = 0b;

      for (i from 0 to n-1){
            sum[i] = fA(a[i],b[i],c[i])[sum];
            c[i+1] = fA(a[i],b[i],c[i])[carry];
      };
      out:sum<n>;
}
```

5. Translator Architecture



6. Test Plan

There were a series of unit tests suites and integration test suites written for the UNI-corn compiler. Following the implementation of each compiler component (e.g. harden.ml, codgen.ml etc) all previous and latest test scripts were ran to ensure the most recent compilation did not interfere with a previous functionality of the language.

1) **The test suite is all under one directory called testCases**
    a) **Every subdirectory in ./testsCases/**
        i) checks that every aspect of the Uni-Corn language reference manual produces the expected output. Each subdirectory is named after the feature it's testing. There are test cases in there that contain invalid programs which should report errors.

    b) **There is a programs folder in ./testCases called programs:**
        i) This has a bunch of programs like alu.uni and genAdder2.uni which are programs that are fully written in Uni-Corn.

6.1. Unit Testing

As the compiler was developed, tests were written for every new feature or component to verify its functionality. For each feature, depending on the number of equivalence classes, one to two unit tests were written and added to the appropriate subdirectory within the test directory.

6.2. Integration Testing

The integration tests were slowly added over the development timeline. Within these tests, most bugs within the compiler were uncovered. Some, bug that showed up during integration testing displayed a fundamental problem within the compiler implementation and therefore evoked restructuring.

6.3. Test Automation

The testall.sh script in the unicorn directory compiles, runs and links all the files within the test directory. The results of the testall script are then pretty printed with results showing if the test either failed or passed. Those intended to pass are denoted with the word “pass” and those intended to fail are marked with a “fail.”

6.4. Team Responsibilities for Testing

The team responsibilities were divided roughly across the five members of the team; however Maryam, the team tester, took responsibility of introducing new tests, approving other team members tests and maintaining test directory integrity and cleanliness.

generic.uni:

```
main() {
      z = 101b;
      b = 101b;
      m = modA(z)[m];
      print m: m;
      out:;
}

modA(a<n>){
      for (i from 0 to n-1){
            m[i] = a[i] or (not a[i]);
      };
      out:m<n>;
}
```

generic.uni as target language:

```
; ModuleID = 'Unic'
source_filename = "Unic"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@prt = private unnamed_addr constant [10 x i8] c"m[0]: %d\0A\00"
@prt.2 = private unnamed_addr constant [10 x i8] c"m[1]: %d\0A\00"
@prt.3 = private unnamed_addr constant [10 x i8] c"m[2]: %d\0A\00"

declare i32 @printf(i8*, ...)

declare i32 @printbig(i32)

define i1 @tick() {
entry:
  %m_2 = alloca i1
```

```
%m_1 = alloca i1
%m_0 = alloca i1
%".modA_0_m[2]" = alloca i1
%".modA_0_m[1]" = alloca i1
%".modA_0_m[0]" = alloca i1
%"#6[0]" = alloca i1
%"#8[0]" = alloca i1
%"#7[0]" = alloca i1
%z_2 = alloca i1
%"#3[0]" = alloca i1
%"#5[0]" = alloca i1
%"#4[0]" = alloca i1
%z_1 = alloca i1
%"#0[0]" = alloca i1
%"#2[0]" = alloca i1
%"#1[0]" = alloca i1
%z_0 = alloca i1
%b_2 = alloca i1
%b_1 = alloca i1
%b_0 = alloca i1
%0 = alloca i1
store i1 true, i1* %b_0
store i1 false, i1* %b_1
store i1 true, i1* %b_2
store i1 true, i1* %z_0
%z_01 = load i1, i1* %z_0
store i1 %z_01, i1* %"#1[0]"
%"#1[0]2" = load i1, i1* %"#1[0]"
%"#2[0]'svalue" = xor i1 %"#1[0]2", true
store i1 %"#2[0]'svalue", i1* %"#2[0]"
%z_03 = load i1, i1* %z_0
store i1 %z_03, i1* %"#0[0]"
store i1 false, i1* %z_1
%z_14 = load i1, i1* %z_1
store i1 %z_14, i1* %"#4[0]"
%"#4[0]5" = load i1, i1* %"#4[0]"
%"#5[0]'svalue" = xor i1 %"#4[0]5", true
store i1 %"#5[0]'svalue", i1* %"#5[0]"
%z_16 = load i1, i1* %z_1
store i1 %z_16, i1* %"#3[0]"
store i1 true, i1* %z_2
%z_27 = load i1, i1* %z_2
```

```
  store i1 %z_27, i1* %"#7[0]"
  %"#7[0]8" = load i1, i1* %"#7[0]"
  %"#8[0]'svalue" = xor i1 %"#7[0]8", true
  store i1 %"#8[0]'svalue", i1* %"#8[0]"
  %z_29 = load i1, i1* %z_2
  store i1 %z_29, i1* %"#6[0]"
  %"#2[0]10" = load i1, i1* %"#2[0]"
  %"#0[0]11" = load i1, i1* %"#0[0]"
  %".modA_0_m[0]'svalue" = or i1 %"#0[0]11", %"#2[0]10"
  store i1 %".modA_0_m[0]'svalue", i1* %".modA_0_m[0]"
  %"#5[0]12" = load i1, i1* %"#5[0]"
  %"#3[0]13" = load i1, i1* %"#3[0]"
  %".modA_0_m[1]'svalue" = or i1 %"#3[0]13", %"#5[0]12"
  store i1 %".modA_0_m[1]'svalue", i1* %".modA_0_m[1]"
  %"#8[0]14" = load i1, i1* %"#8[0]"
  %"#6[0]15" = load i1, i1* %"#6[0]"
  %".modA_0_m[2]'svalue" = or i1 %"#6[0]15", %"#8[0]14"
  store i1 %".modA_0_m[2]'svalue", i1* %".modA_0_m[2]"
  %".modA_0_m[0]16" = load i1, i1* %".modA_0_m[0]"
  store i1 %".modA_0_m[0]16", i1* %m_0
  %m_017 = load i1, i1* %m_0
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt, i32 0, i32 0), i1 %m_017)
  %".modA_0_m[1]18" = load i1, i1* %".modA_0_m[1]"
  store i1 %".modA_0_m[1]18", i1* %m_1
  %m_119 = load i1, i1* %m_1
  %printf20 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt.2, i32 0, i32 0), i1 %m_119)
  %".modA_0_m[2]21" = load i1, i1* %".modA_0_m[2]"
  store i1 %".modA_0_m[2]21", i1* %m_2
  %m_222 = load i1, i1* %m_2
  %printf23 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt.3, i32 0, i32 0), i1 %m_222)
  ret i1 false
}
```

Test Suite Structure is as follows:

```
Structure is as follows:
```

```
./testCases/
--- EOFTerminators/
        ---test1.uni
        --- test2.uni
--- Main/
        ---test1.uni
        ---test2.uni
---comments/
        ---test1.uni
        ---test2.uni
        ---test3.uni
---creatingBuses/
        ---test1.uni
       ---test2.uni
       ---test3.uni
       ---test4.uni
       ---test5.uni
       ---test6.uni
---evaluatingGates/
       ---test1.uni
       ---test2.uni
       ---test3.uni
       ---test4.uni
       ---test5.uni
       ---test6.uni
       ---test7.uni
       ---test8.uni
       ---test9.uni
       ---evaluatingGates
---gatePrecedence/
       ---test1.uni
       ---test2.uni
       ---test3.uni
       ---test4.uni
---indexing/
       ---test1.uni
       ---test2.uni
       ---test3.uni
       ---test4.uni
       ---test5.uni
       ---test6.uni
       ---test7.uni
       ---test8.uni
       ---test9.uni
---keywords/
       ---test1.uni
       ---test2.uni
       ---test3.uni
       ---test4.uni
       ---test5.uni
       ---test6.uni
       ---test7.uni
       ---test8.uni
       ---test9.uni
---overloading/
       ---test1.uni
      ---test2.uni
---printFunc
      ---test1.uni
      ---test2.uni
---registers/
      ---test1.uni
      ---test2.uni
      ---test3.uni
      ---test4.uni
      ---test5.uni
---programs/
      ---alu.uni
      ---genAdder2.uni
---run.sh
--- c.c
---testall.sh
```

Test Cases:

```
unicorn/testCases/comments/test2.uni
```

```
main() {
      //print a : 110b;
      //print b : 101b;
      print c : 011b;
      out:;
}


//program should print 011

unicorn/testCases/comments/test3.uni
main() {
      //print a : 110b;

      /**
      print b : 101b;
      hello there how are you today
      hope at this point we're almost done with this project :)
      **/

      print c : 01011b;
      out:;
}



//program should print out 01011


unicorn/testCases/comments/test1.uni
main() {
      print a : 110b;
      /**
      print b : 101b;
      print c : 011b;
      **/
      out:;
}


//program should print out 110
```

```
unicorn/testCases/printFunc/test2.uni
main(){
      a = 1110b;
      b = 111111111b;

      print : b;


}


//should give an error because there is nothing after print

unicorn/testCases/printFunc/test1.uni
main(){

      a = 10101b;
      b = 11111b;

      c = a and b;

      print printedItem: c;
      out: ;

}
neigh!
//10101

unicorn/testCases/creatingBuses/test4.uni
main(){
      1212assak = 10111b;
      out:;

}
//error because name of bus starts with number


unicorn/testCases/creatingBuses/test5.uni
main(){
      !hellp= 1011b;
      out:;
}

//error because bus name doesn't start with a letter

unicorn/testCases/creatingBuses/test6.uni
main(){
      busOne = 121222b;
      out:;


}

//should give an error because its not 1's and 0's

unicorn/testCases/creatingBuses/test2.uni
main(){

      busHelloThere = 101010101010; //error --> no b
      out:;
}


unicorn/testCases/creatingBuses/test3.uni
main(){

      busZ= 1011b;
      bussO= 111111b;
      bbOO= 000000b;
      out:;
}


unicorn/testCases/creatingBuses/test1.uni
main(){
      busOne = 100011b; //valid
      busTwo = 1111100000b; //valid
      sasakSSEWWW = 1001011b; //valid

      out:;

}
```

```
unicorn/testCases/overloading/test2.uni
main(){

      a = 101b;
      a = 111b; //invalid because of overloading
      print x: a;
      out: a<3>;
}
neigh!
//error

unicorn/testCases/overloading/test1.uni
main(){
      a = 1011b;
      b = 11111b;
      a = 1111b; // should be invalid because its overloading
      out:;
}
neigh!
//error

unicorn/testCases/registers/test4.uni
main(){

      a = 1101b;
      b = 1101010010b;

      x:= a *0101010010101*; //error because of size of initializing
      out:;
}
neigh!

unicorn/testCases/registers/test5.uni
main(){
      a=100b;
      x:= a *000b*;
      print asa: x;
      out:;

}
```

```
unicorn/testCases/registers/test2.uni
main(){
      a = 1101b;
      b = 110000001111b;

      a:= a *0000b*; //error because bus is already called a -- overloading
      print asas: a;
      out:;
}
neigh!
//should print 1101

unicorn/testCases/registers/test3.uni
main(){

      a = 1100b;
      b = 1010b;

      x := (a or b) *1111b*;
      print asasa: x;
      out:;
}
neigh!
//should print 1110

unicorn/testCases/registers/test1.uni
main(){
      a = 111b;
      b =101b;

      c = a or b;

      x:=c *000b*;
      print xyc: x;
      out:;
}

//should output 111

unicorn/testCases/EOFTerminators/test2.uni
main(){
      bustwo = 1111b;
      print b: bustwo;
      out:;
}


//program should compile

unicorn/testCases/EOFTerminators/test1.uni
main(){
      busone = 101b;
      print a : busone;
      out:;
}
neigh!
//program should compile

unicorn/testCases/gatePrecedence/test4.uni
main(){

      a= 10b;
      b =11b;
      c =1b;
      d =1b;

      res = a[0] !* b[0] # c !+ a[1];
      print answer: res;
      out:;

}
neigh!
//should output 1

unicorn/testCases/gatePrecedence/test2.uni
main(){

      a= 1010b;
      b =1111b;
      c =1001b;
      d =1000b;

      //res = (a and b) !  (((! c) + a) !+ d);
      res = a and b + d !+ c + a !+ d;
      print answer: res;
      out:;

}
neigh!
//should output 0001

unicorn/testCases/gatePrecedence/test3.uni
main(){

      a= 10b;
      b =11b;
      c =00b;

      res = a or b !+ c + a !# c !+ b;
      print answer: res;
      out:;

}
neigh!
//should output 01

unicorn/testCases/gatePrecedence/test1.uni
main(){

      a= 1010b;
      b =1111b;
      c =1001b;
      d =1000b;

      //res = (a and b) !  (((! c) + a) !+ d);
      res = a and b # ! c + a !+ d;
      print answer: res;
      out:;

}
neigh!
//should output 1011
//a and b: 1010
//!c = 0110
//!c + a = 1110
```

```
//0001 xor 1010
//1011


unicorn/testCases/programs/genAdder2.uni
main() {
      a = 1011010b;
      b = 0011101b;
      m = modA(a,b)[sum];
      print m: m;
      out:;
}

fA(a, b, carryIn){
      axb = a xor b;
    sum = axb xor carryIn;
    carry = (axb and carryIn) or (a and b);
      out: sum, carry;
}

modA(a<n>,b<n>){
      c[0] = 0b;

      for (i from 0 to n-1){
            sum[i] = fA(a[i],b[i],c[i])[sum];
            c[i+1] = fA(a[i],b[i],c[i])[carry];
      };
      out:sum<n>;
}


unicorn/testCases/programs/alu.uni
/**
      000 -> A+B
      001 -> A-B
      100 -> A&B
      101 -> A|B
      110 -> A xor B
      111 -> Not A
**/

main(){
      a = 00111111b;
      b = 11000100b;
      x = alu(111b,a,b);
      print x: x;
      out:;
}

alu(instr<3>,a<n>,b<n>){
      m = math(a,b,instr[0]);
      l = logic(a,b,instr[0:1]);
      o = mux(m,l,instr[2]);
      out:o<n>;
}

math(a<n>,b<n>,sel){
      bc = comp(b);
      bnew = mux(b,bc,sel);
      sum = add(a,bnew);
      out:sum<n>;
}

comp(a<n>){
      a1 = not a;
      for (i from 1 to n-1){
            one[i] = 0b;
      };
      one[0] = 1b;
      cmp = add(a1,one);
      out:cmp<n>;
}

fA(a, b, carryIn){
      axb = a xor b;
    sum = axb xor carryIn;
    carry = (axb and carryIn) or (a and b);
      out: sum, carry;
}

add(a<n>,b<n>){
      c[0] = 0b;

      for (i from 0 to n-1){
            sum[i] = fA(a[i],b[i],c[i])[sum];
            c[i+1] = fA(a[i],b[i],c[i])[carry];
      };
      out:sum<n>;
}

mux(a<n>,b<n>,s){
      for (i from 0 to n-1){
            as[i] = a[i] and not s;
            bs[i] = b[i] and s;
            o[i] = as[i] or bs[i];
      };
      out:o<n>;
}

mux2(a<n>,b<n>,c<n>,d<n>,sel<2>){
      m1 = mux(a,b,sel[0]);
      m2 = mux(c,d,sel[0]);
      m  = mux(m1,m2,sel[1]);
      out:m<n>;
}

logic(a<n>, b<n>, sel<2>){
      AND = a and b;
      OR = a or b;
      XOR = a xor b;
      NOT = not a;
      o = mux2(AND,OR,XOR,NOT,sel);
      out:o<n>;
}


unicorn/testCases/indexing/test8.uni
main(){

      a = 1010b;
      b[1:4] = 1101b;
      b[0] = 1b;
      print asa : b;
      out:;

}
```

```
neigh!
//prints 11011

unicorn/testCases/indexing/test9.uni
main(){
      s = 10101010010111111b;
      c = s[0:7][0:3][1];
      print asa : c;
      out:;
}

//print 1

unicorn/testCases/indexing/test4.uni
main(){

      a = 101010b;
      b = a[0:3];

      print saa: b;
      out:;

}
neigh!
//print 0101

unicorn/testCases/indexing/test5.uni
main(){
      a=10101b;
      b= a[0+2:4];

      print ahsa: b;
      out:;

}
neigh!
//prints 101

unicorn/testCases/indexing/test7.uni
main(){

      a =10101b;
      b [1:3]= 101b;
      //error because b[0] is not defined
}     out:;
neigh!

unicorn/testCases/indexing/test6.uni
main(){

      a = 0101b;
      b = a[-1:3]; //error because no negative slicing
      out:;

}
neigh!


unicorn/testCases/indexing/test2.uni
main(){

      c[0]= 1b;
      c[1]= 0b;
      c[2]= 1b;
      c[3] =1b;
      print asa: c;
      out:;

}
neigh!
//print 1101

unicorn/testCases/indexing/test3.uni
main(){

      c = 01010b;
      a = c[1];

      print asa: a;
      out:;

}
neigh!
//print 1
```

```
unicorn/testCases/indexing/test1.uni
main(){

      a = 1010b;
      a[0]= 1b;
      a[1]= 1b;
      a[2]= 1b;
      a[3]= 1b;

      print asa: a;
      out:;
}
neigh!
//should give an error because you're overloading


unicorn/testCases/main/test2.uni
main(){

      print a : 10111111b;
      print x : 10111b;
      out: ;
}
neigh!
//this should print out 1011111 and 10111

unicorn/testCases/main/test1.uni
method1(a<2>, b<2>){
      print a: 1011b;
      out: ;
}
neigh!
//program should give an error because there is no main in this program


unicorn/testCases/keywords/test8.uni
main(){
      neigh! = 101010b;
      out:;

}
```

```
//should give an error because neigh! is a reserved word (EOF)

unicorn/testCases/keywords/test9.uni
main(){

      busTwooo= 10011101b; //valid
      neigh = 11001b; //error

}
neigh

unicorn/testCases/keywords/test4.uni
main(){
      busOne= 1111000b; //valid
      out = busOne; //error

}
neigh!

unicorn/testCases/keywords/test5.uni
main(){
      helloThereBus = 1011100000b; //valid
      from = 100111b; //error
      out:;
}
neigh!

unicorn/testCases/keywords/test7.uni
main(){
      busOne= 100111b; //valid
      to = 11111100b; //error
      out:;
}
neigh!

unicorn/testCases/keywords/test6.uni
main(){

      busExpr = 100111b; //valid
      for = 11111100b; //error
      out:;
```

```
}
neigh!

unicorn/testCases/keywords/test2.uni
main(){
      busTwo = 1000b; //valid
      print = 1111b; //error

      out:;
}
neigh!

unicorn/testCases/keywords/test3.uni
main(){
      busThree = 11111111b; //valid
      neigh=1100000b; //valid

      out:;
}
neigh!

unicorn/testCases/keywords/test1.uni
main() {
      busOne = 11101b; //valid
      make = 110b; //valid

      out:;

}
neigh!

unicorn/testCases/evaluatingGates/test8.uni
main(){

      a = 111b;
      b = 101010100b;


      c = a and b; //error because a and b are different sizes
      print w: c;
      out: c<3>;
```

```
}
neigh!

unicorn/testCases/evaluatingGates/test9.uni
main(){

      a= 111111b;
      b=101b;

      c= a or b;   //error because of different size
      print q : c;
      out: c<3>;
}
neigh!

unicorn/testCases/evaluatingGates/test4.uni
main(){

      one = 10111b;
      two = not one;

      print a: two;
      out: two<5>;
}
neigh!
//should print 01000

unicorn/testCases/evaluatingGates/test5.uni
main(){

      bOne = 101010b;
      bTwo = 111110b;

      bThr=not(bOne and bTwo);
      print e: bThr;
      out: bThr<6>;
}
neigh!
//should print 010101

unicorn/testCases/evaluatingGates/test7.uni
main(){
      bOne = 10101b;
      bTwo = 11110b;

      bThr = not(bOne xor bTwo);

      print p: bThr;
      out: bThr<5>;
}
neigh!
//should print 10100

unicorn/testCases/evaluatingGates/test6.uni
main(){
      bOne = 101010b;
      bTwo = 101111b;

      bThr = not(bOne or bTwo) ;
      print w: bThr;
      out: bThr<6>;

}
neigh!
//should print 000101

unicorn/testCases/evaluatingGates/test2.uni
main(){

      bOne = 10101b;
      bTwo = 11100b;

      bThree = bOne or bTwo;

      print z: bThree;
      out: bThree<5>;

}
neigh!
//program should print 11101

unicorn/testCases/evaluatingGates/test3.uni
main(){
      bOne = 1100b;
      bTwo = 1101b;

      bRes = not(bOne or bTwo);
      print o: bRes;
      out: bRes<4>;
}
neigh!
//program should output 0010

unicorn/testCases/evaluatingGates/test1.uni
main(){

      busOne = 1110b;
      busTwo = 1011b;
      busThree = busOne and busTwo;

      print a : busThree;
      out: busThree<4>;
}
neigh!

//program should print 1010

unicorn/testCases/evaluatingGates/test10.uni
main(){

      a = 1010b;
      b = 1111b;

      c = a and b;
      print q: c;
      out: c<2>;
}

//this gives you an error because c<2> not c<4>
```

testall.sh:

```
#!/bin/bash
echo
===========================================================================
echo -e "\n"
echo "           Here is our automated testing per feature
"
echo -e "\n"
echo
===========================================================================
=
echo -e "\n"
echo ----------------------
echo Testing Comments
echo ----------------------

./run.sh comments/test1 c.c
rm comments/test1
./run.sh comments/test2 c.c
rm comments/test2
./run.sh comments/test3 c.c
rm comments/test3

echo -e "\n"
echo ---------------------
echo Testing Buses Datatype
echo ---------------------

./run.sh creatingBuses/test1 c.c
rm creatingBuses/test1
./run.sh creatingBuses/test2 c.c
rm creatingBuses/test2
echo =======================this should give an error====================
./run.sh creatingBuses/test3 c.c
rm creatingBuses/test3
./run.sh creatingBuses/test4 c.c
rm creatingBuses/test4
echo =================this should give an error===================
./run.sh creatingBuses/test5 c.c
rm creatingBuses/test5
echo ============this should give an error======================
./run.sh creatingBuses/test6 c.c
```

```
rm creatingBuses/test6
echo ==================this should give an error================
echo -e "\n"
echo ----------------------
echo Testing EOF Terminators
echo ----------------------

./run.sh EOFTerminators/test1 c.c
rm EOFTerminators/test1
./run.sh EOFTerminators/test2 c.c
rm EOFTerminators/test2

echo -e "\n"
echo -----------------------
echo Testing Built-in Gates
echo -----------------------

./run.sh evaluatingGates/test1 c.c
rm evaluatingGates/test1
./run.sh evaluatingGates/test2 c.c
rm evaluatingGates/test2
./run.sh evaluatingGates/test3 c.c
rm evaluatingGates/test3
./run.sh evaluatingGates/test4 c.c
rm evaluatingGates/test4
./run.sh evaluatingGates/test5 c.c
rm evaluatingGates/test5
./run.sh evaluatingGates/test6 c.c
rm evaluatingGates/test6
./run.sh evaluatingGates/test7 c.c
rm evaluatingGates/test7
./run.sh evaluatingGates/test8 c.c
rm evaluatingGates/test8
echo ===================this should give an error================
./run.sh evaluatingGates/test9 c.c
rm evaluatingGates/test9
echo ======================this should give an error=================
./run.sh evaluatingGates/test10 c.c
rm evaluatingGates/test10
echo ===================this should give an error====================
echo -e "\n"
echo ---------------------
echo Testing Indexing
echo ----------------------


./run.sh indexing/test1 c.c
rm indexing/test1
echo ======================this should give an error====================
./run.sh indexing/test2 c.c
rm indexing/test2
./run.sh indexing/test3 c.c
rm indexing/test3
./run.sh indexing/test4 c.c
rm indexing/test4
./run.sh indexing/test5 c.c
rm indexing/test5
./run.sh indexing/test6 c.c
rm indexing/test6
echo =====================this should give an error================
./run.sh indexing/test7 c.c
echo =======================this should give an error=================
rm indexing/test7
./run.sh indexing/test8 c.c
rm indexing/test8
./run.sh indexing/test9 c.c
rm indexing/test9


echo -e "\n"
echo ---------------------------
echo Testing Reserved Words
echo --------------------------


./run.sh keywords/test1 c.c
rm keywords/test1
./run.sh keywords/test2 c.c
rm keywords/test2
echo ====================this should give an error================
./run.sh keywords/test3 c.c
rm keywords/test3
```

```
./run.sh keywords/test4 c.c
rm keywords/test4
echo ===================this should give an error=============
 ./run.sh keywords/test5 c.c
rm keywords/test5
echo ====================this should give an error=================
./run.sh keywords/test6 c.c
rm keywords/test6
echo ================this should give an error=========================
 ./run.sh keywords/test7 c.c
rm keywords/test7
echo ==============this should give an error======================
./run.sh keywords/test8 c.c
rm keywords/test8
echo ===============this should give an error===========================
 ./run.sh keywords/test9 c.c
rm keywords/test9
echo ===================this should give an error======================



echo -e "\n"
echo -------------------------
echo Testing Main in Module
echo ------------------------

./run.sh Main/test1 c.c
rm Main/test1
echo ===========================this should give an error==================
 ./run.sh Main/test2 c.c
rm Main/test2


echo -e "\n"
echo -----------------------
echo Testing overloading and Re-assignement
echo -----------------------

./run.sh overloading/test1 c.c
rm overloading/test1
echo ===============this should give an error============================
 ./run.sh overloading/test2 c.c
```

```
rm overloading/test2
echo =============this should give an error============================


echo -e "\n"
echo -----------------------
echo Testing the print function
echo -----------------------
./run.sh printFunc/test1 c.c
rm printFunc/test1
 ./run.sh printFunc/test2 c.c
rm printFunc/test2
echo ===================this should give an error==========================

echo -e "\n"
echo -----------------------
echo Testing registers
echo ----------------------

./run.sh registers/test1 c.c
rm registers/test1
 ./run.sh registers/test2 c.c
rm registers/test2
./run.sh registers/test3 c.c
rm registers/test3
 ./run.sh registers/test4 c.c
rm registers/test4
echo ========================this should give an
error=======================
./run.sh registers/test5 c.c
rm registers/test5

echo -e "\n"
echo -----------------------
echo Testing Sample Program: Generic Adder
echo -----------------------
./run.sh programs/genAdder2 c.c
rm programs/genAdder2

echo -e "\n"
echo -----------------------
echo Testing Sample Program: ALU
echo -----------------------
./run.sh programs/alu c.c
rm programs/alu

echo -----------------------
echo Testing gate Precedence
echo -----------------------
./run.sh gatePrecedence/test1 c.c
rm gatePrecedence/test1
./run.sh gatePrecedence/test2 c.c
rm gatePrecedence/test2
./run.sh gatePrecedence/test3 c.c
rm gatePrecedence/test3
./run.sh gatePrecedence/test4 c.c
rm gatePrecedence/test4

echo -e "\n"
echo -e "\n"
echo =====================================================================
echo "                            End of Testing!
"
echo -e "\n"
echo -e "\n"
```

testall.sh results:

```
===========================================================================
=========================================================================


                                                     Here is our automated
testing per feature


===========================================================================
=========================================================================



----------------------
Testing Comments
----------------------
comments/test1
tick1
a[0]: 0
a[1]: 1
a[2]: 1

tick2
a[0]: 0
a[1]: 1
a[2]: 1

tick3
a[0]: 0
a[1]: 1
a[2]: 1


comments/test2
tick1
c[0]: 1
c[1]: 1
c[2]: 0

tick2
c[0]: 1
c[1]: 1
```

```
c[2]: 0

tick3
c[0]: 1
c[1]: 1
c[2]: 0


comments/test3
tick1
c[0]: 1
c[1]: 1
c[2]: 0
c[3]: 1
c[4]: 0

tick2
c[0]: 1
c[1]: 1
c[2]: 0
c[3]: 1
c[4]: 0

tick3
c[0]: 1
c[1]: 1
c[2]: 0
c[3]: 1
c[4]: 0




---------------------
Testing Buses Datatype
---------------------
creatingBuses/test1
tick1

tick2

tick3
```

```
creatingBuses/test2
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-063271.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
./run.sh: line 7: ./creatingBuses/test2: No such file or directory


rm: creatingBuses/test2: No such file or directory
============================================================ this should give an error ============================================================
creatingBuses/test3
tick1

tick2

tick3


creatingBuses/test4
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-72a2f2.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
./run.sh: line 7: ./creatingBuses/test4: No such file or directory


rm: creatingBuses/test4: No such file or directory
============================================================ this should give an error ============================================================
creatingBuses/test5
Fatal error: exception Semant2.InvalidAssignment(""~hellp" may not be assigned to")
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-92d462.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./creatingBuses/test5: No such file or directory


rm: creatingBuses/test5: No such file or directory
============================================================ this should
give an error ===========================================================
creatingBuses/test6
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-6529c8.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./creatingBuses/test6: No such file or directory


rm: creatingBuses/test6: No such file or directory
=============================================================this should
give an error ==========================================================



----------------------
Testing EOF Terminators
----------------------
EOFTerminators/test1
tick1
a[0]: 1
a[1]: 0
a[2]: 1

tick2
a[0]: 1
a[1]: 0
a[2]: 1

tick3
```

```
a[0]: 1
a[1]: 0
a[2]: 1


EOFTerminators/test2
tick1
b[0]: 1
b[1]: 1
b[2]: 1
b[3]: 1

tick2
b[0]: 1
b[1]: 1
b[2]: 1
b[3]: 1

tick3
b[0]: 1
b[1]: 1
b[2]: 1
b[3]: 1




-----------------------
Testing Built-in Gates
-----------------------
evaluatingGates/test1
tick1
a[0]: 0
a[1]: 1
a[2]: 0
a[3]: 1

tick2
a[0]: 0
a[1]: 1
a[2]: 0
a[3]: 1
```

```
tick3
a[0]: 0
a[1]: 1
a[2]: 0
a[3]: 1


evaluatingGates/test2
tick1
z[0]: 1
z[1]: 0
z[2]: 1
z[3]: 1
z[4]: 1

tick2
z[0]: 1
z[1]: 0
z[2]: 1
z[3]: 1
z[4]: 1

tick3
z[0]: 1
z[1]: 0
z[2]: 1
z[3]: 1
z[4]: 1


evaluatingGates/test3
tick1
o[0]: 0
o[1]: 1
o[2]: 0
o[3]: 0

tick2
o[0]: 0
o[1]: 1
o[2]: 0
```

```
o[3]: 0

tick3
o[0]: 0
o[1]: 1
o[2]: 0
o[3]: 0


evaluatingGates/test4
tick1
a[0]: 0
a[1]: 0
a[2]: 0
a[3]: 1
a[4]: 0

tick2
a[0]: 0
a[1]: 0
a[2]: 0
a[3]: 1
a[4]: 0

tick3
a[0]: 0
a[1]: 0
a[2]: 0
a[3]: 1
a[4]: 0


evaluatingGates/test5
tick1
e[0]: 1
e[1]: 0
e[2]: 1
e[3]: 0
e[4]: 1
e[5]: 0

tick2
```

```
e[0]: 1
e[1]: 0
e[2]: 1
e[3]: 0
e[4]: 1
e[5]: 0

tick3
e[0]: 1
e[1]: 0
e[2]: 1
e[3]: 0
e[4]: 1
e[5]: 0


evaluatingGates/test6
tick1
w[0]: 0
w[1]: 0
w[2]: 0
w[3]: 0
w[4]: 1
w[5]: 0

tick2
w[0]: 0
w[1]: 0
w[2]: 0
w[3]: 0
w[4]: 1
w[5]: 0

tick3
w[0]: 0
w[1]: 0
w[2]: 0
w[3]: 0
w[4]: 1
w[5]: 0
```

```
evaluatingGates/test7
tick1
p[0]: 0
p[1]: 0
p[2]: 1
p[3]: 0
p[4]: 1

tick2
p[0]: 0
p[1]: 0
p[2]: 1
p[3]: 0
p[4]: 1

tick3
p[0]: 0
p[1]: 0
p[2]: 1
p[3]: 0
p[4]: 1


evaluatingGates/test8
Fatal error: exception ERR.TypeMismatch("You tried performing And on a and
b but these are of different sizes. But You can fix that!!    ")
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-9f8a11.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./evaluatingGates/test8: No such file or directory


rm: evaluatingGates/test8: No such file or directory
========================================================= This should give
an error ==============================================================
evaluatingGates/test9
Fatal error: exception ERR.TypeMismatch("You tried performing Or on a and b
but these are of different sizes. But You can fix that!!    ")
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-6747c8.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./evaluatingGates/test9: No such file or directory


rm: evaluatingGates/test9: No such file or directory
===========================================================This should give
an error ==========================================================
evaluatingGates/test10
Fatal error: exception ERR.TypeMismatch("The output call c<2> does not
match assignment of size 4So it goes, I guess ¯\_(ツ)_/¯")
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-2957f2.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./evaluatingGates/test10: No such file or directory


rm: evaluatingGates/test10: No such file or directory
============================================================This should
give an error ============================================================



---------------------
Testing Indexing
----------------------
indexing/test1
Fatal error: exception Topsort4.Reassign(".main_0_a[0] has been assigned to
twice!")
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-57b28d.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./indexing/test1: No such file or directory
```

```
rm: indexing/test1: No such file or directory
=============================================================== This should
give an error ========================================================
indexing/test2
tick1
asa[0]: 1
asa[1]: 0
asa[2]: 1
asa[3]: 1

tick2
asa[0]: 1
asa[1]: 0
asa[2]: 1
asa[3]: 1

tick3
asa[0]: 1
asa[1]: 0
asa[2]: 1
asa[3]: 1


indexing/test3
tick1
asa[0]: 1

tick2
asa[0]: 1

tick3
asa[0]: 1


indexing/test4
tick1
saa[0]: 0
saa[1]: 1
saa[2]: 0
saa[3]: 1
```

```
tick2
saa[0]: 0
saa[1]: 1
saa[2]: 0
saa[3]: 1

tick3
saa[0]: 0
saa[1]: 1
saa[2]: 0
saa[3]: 1


indexing/test5
tick1
ahsa[0]: 1
ahsa[1]: 0
ahsa[2]: 1

tick2
ahsa[0]: 1
ahsa[1]: 0
ahsa[2]: 1

tick3
ahsa[0]: 1
ahsa[1]: 0
ahsa[2]: 1


indexing/test6
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-f92c4f.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./indexing/test6: No such file or directory


rm: indexing/test6: No such file or directory
```

```
=============================================================== This should
give an error ========================================================
indexing/test7
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-339c25.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./indexing/test7: No such file or directory


================================================================This should
give an error ========================================================
rm: indexing/test7: No such file or directory
indexing/test8
tick1
asa[0]: 1
asa[1]: 1
asa[2]: 0
asa[3]: 1
asa[4]: 1

tick2
asa[0]: 1
asa[1]: 1
asa[2]: 0
asa[3]: 1
asa[4]: 1

tick3
asa[0]: 1
asa[1]: 1
asa[2]: 0
asa[3]: 1
asa[4]: 1


indexing/test9
tick1
asa[0]: 1
```

```
tick2
asa[0]: 1

tick3
asa[0]: 1




---------------------------
Testing Reserved Words
--------------------------
keywords/test1
tick1

tick2

tick3


keywords/test2
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-7e8dc7.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see invocation)
./run.sh: line 7: ./keywords/test2: No such file or directory


rm: keywords/test2: No such file or directory
=================================================================This should give an error
=======================================================
keywords/test3
tick1

tick2

tick3
```

```
keywords/test4
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-891a38.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./keywords/test4: No such file or directory


rm: keywords/test4: No such file or directory
==================================================================This
should give an error ======================================================
keywords/test5
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-45b9a8.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./keywords/test5: No such file or directory


rm: keywords/test5: No such file or directory
==================================================================This
should give an error=======================================================
keywords/test6
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-5d3db1.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./keywords/test6: No such file or directory


rm: keywords/test6: No such file or directory
```

```
==================================================================This
should give an error ======================================================
keywords/test7
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-9c3b13.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./keywords/test7: No such file or directory


rm: keywords/test7: No such file or directory
==================================================================This
should give an error ======================================================
keywords/test8
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-6b40ab.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./keywords/test8: No such file or directory


rm: keywords/test8: No such file or directory
================================================================== This
should give an error =====================================================
keywords/test9
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-5395e4.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./keywords/test9: No such file or directory


rm: keywords/test9: No such file or directory
```

```
===================================================================This
should give an error ======================================================



-------------------------
Testing Main in Module
------------------------
Main/test1
Fatal error: exception Modfill.MissingFunction("There is no main function.
Please create a main")
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-5f26dc.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./Main/test1: No such file or directory


rm: Main/test1: No such file or directory
===================================================================This
should give an error ======================================================
Main/test2
tick1
a[0]: 1
a[1]: 1
a[2]: 1
a[3]: 1
a[4]: 1
a[5]: 1
a[6]: 0
a[7]: 1
x[0]: 1
x[1]: 1
x[2]: 1
x[3]: 0
x[4]: 1

tick2
a[0]: 1
a[1]: 1
a[2]: 1
```

```
a[3]: 1
a[4]: 1
a[5]: 1
a[6]: 0
a[7]: 1
x[0]: 1
x[1]: 1
x[2]: 1
x[3]: 0
x[4]: 1

tick3
a[0]: 1
a[1]: 1
a[2]: 1
a[3]: 1
a[4]: 1
a[5]: 1
a[6]: 0
a[7]: 1
x[0]: 1
x[1]: 1
x[2]: 1
x[3]: 0
x[4]: 1





-----------------------
Testing overloading and Re-assignement
-----------------------
overloading/test1
Fatal error: exception Topsort4.Reassign(".main_0_a[0] has been assigned to
twice!")
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-307efd.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./overloading/test1: No such file or directory
```

```
rm: overloading/test1: No such file or directory
==================================================================== This
should give an error ====================================================
overloading/test2
Fatal error: exception Topsort4.Reassign(".main_0_a[0] has been assigned to
twice!")
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-73e8d5.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./overloading/test2: No such file or directory


rm: overloading/test2: No such file or directory
===================================================================== This
should give an error ===================================================


-----------------------
Testing the print function
-----------------------
printFunc/test1
tick1
printedItem[0]: 1
printedItem[1]: 0
printedItem[2]: 1
printedItem[3]: 0
printedItem[4]: 1

tick2
printedItem[0]: 1
printedItem[1]: 0
printedItem[2]: 1
printedItem[3]: 0
printedItem[4]: 1

tick3
printedItem[0]: 1
```

```
printedItem[1]: 0
printedItem[2]: 1
printedItem[3]: 0
printedItem[4]: 1


printFunc/test2
Fatal error: exception Stdlib.Parsing.Parse_error
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-9b9d0d.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./printFunc/test2: No such file or directory


rm: printFunc/test2: No such file or directory
=======================================================================This
should give an error ==================================================



-----------------------
Testing registers
----------------------
registers/test1
tick1
xyc[0]: 0
xyc[1]: 0
xyc[2]: 0

tick2
xyc[0]: 1
xyc[1]: 1
xyc[2]: 1

tick3
xyc[0]: 1
xyc[1]: 1
xyc[2]: 1
```

```
registers/test2
tick1
asas[0]: 1
asas[1]: 0
asas[2]: 1
asas[3]: 1

tick2
asas[0]: 1
asas[1]: 0
asas[2]: 1
asas[3]: 1

tick3
asas[0]: 1
asas[1]: 0
asas[2]: 1
asas[3]: 1


registers/test3
tick1
asasa[0]: 1
asasa[1]: 1
asasa[2]: 1
asasa[3]: 1

tick2
asasa[0]: 0
asasa[1]: 1
asasa[2]: 1
asasa[3]: 1

tick3
asasa[0]: 0
asasa[1]: 1
asasa[2]: 1
asasa[3]: 1


registers/test4
Fatal error: exception Stdlib.Parsing.Parse_error
```

```
Undefined symbols for architecture x86_64:
  "_tick", referenced from:
      _main in c-41f39a.o
ld: symbol(s) not found for architecture x86_64
clang: error: linker command failed with exit code 1 (use -v to see
invocation)
./run.sh: line 7: ./registers/test4: No such file or directory


rm: registers/test4: No such file or directory
========================================================================Thi
s should give an error ====================================================
registers/test5
tick1
asa[0]: 0
asa[1]: 0
asa[2]: 0

tick2
asa[0]: 0
asa[1]: 0
asa[2]: 1

tick3
asa[0]: 0
asa[1]: 0
asa[2]: 1




-----------------------
Testing Sample Program: Generic Adder
-----------------------
programs/genAdder2
tick1
m[0]: 1
m[1]: 1
m[2]: 1
m[3]: 0
m[4]: 1
m[5]: 1
```

```
m[6]: 1

tick2
m[0]: 1
m[1]: 1
m[2]: 1
m[3]: 0
m[4]: 1
m[5]: 1
m[6]: 1

tick3
m[0]: 1
m[1]: 1
m[2]: 1
m[3]: 0
m[4]: 1
m[5]: 1
m[6]: 1




-----------------------
Testing Sample Program: ALU
-----------------------
programs/alu
tick1
x[0]: 0
x[1]: 0
x[2]: 0
x[3]: 0
x[4]: 0
x[5]: 0
x[6]: 1
x[7]: 1

tick2
x[0]: 0
x[1]: 0
x[2]: 0
x[3]: 0
```

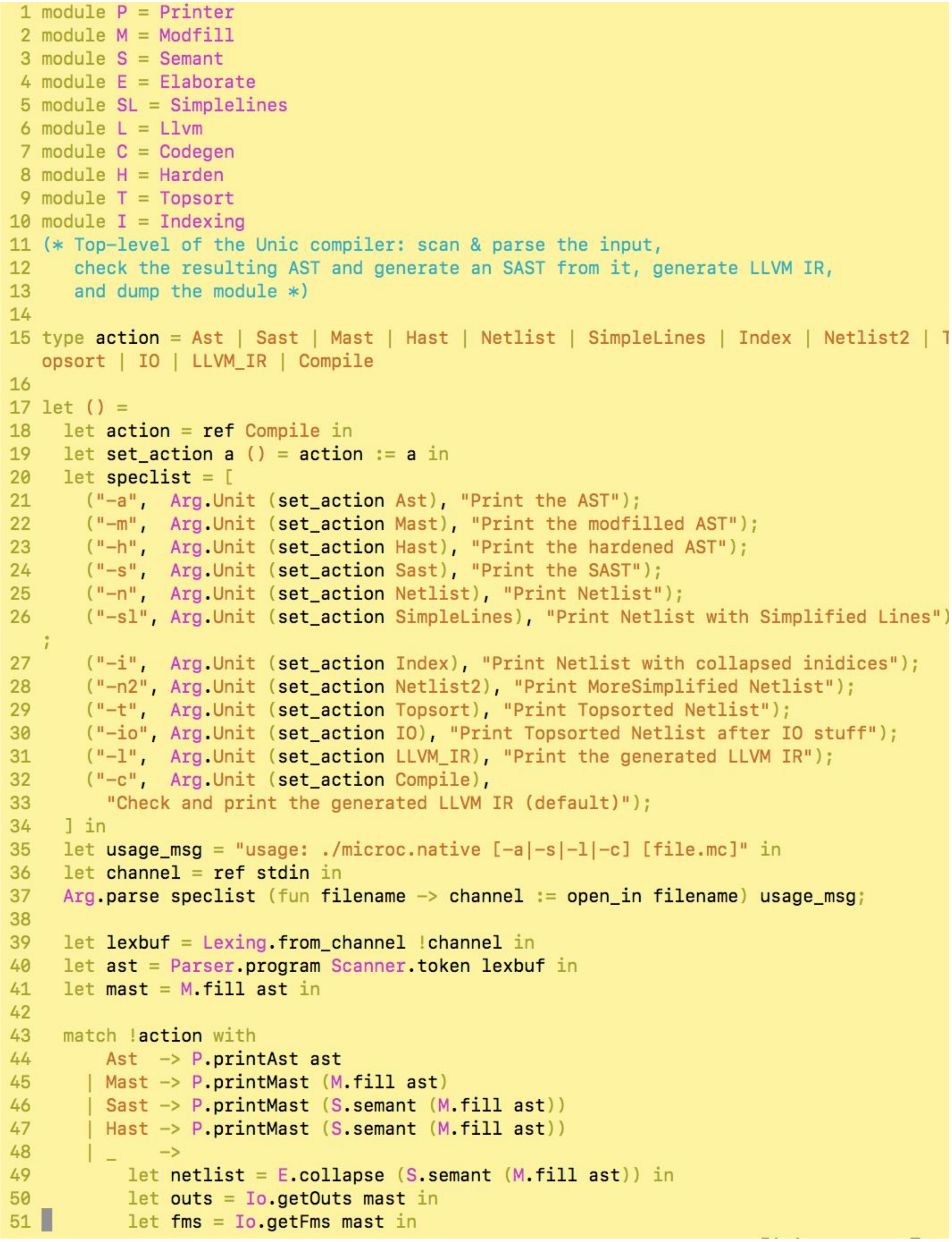
```
x[4]: 0
x[5]: 0
x[6]: 1
x[7]: 1

tick3
x[0]: 0
x[1]: 0
x[2]: 0
x[3]: 0
x[4]: 0
x[5]: 0
x[6]: 1
x[7]: 1


-----------------------
Testing gate Precedence
-----------------------
gatePrecedence/test1
tick1
answer[0]: 1
answer[1]: 1
answer[2]: 0
answer[3]: 1

tick2
answer[0]: 1
answer[1]: 1
answer[2]: 0
answer[3]: 1

tick3
answer[0]: 1
answer[1]: 1
answer[2]: 0
answer[3]: 1


gatePrecedence/test2
tick1
answer[0]: 1
```

```
answer[1]: 0
answer[2]: 0
answer[3]: 0

tick2
answer[0]: 1
answer[1]: 0
answer[2]: 0
answer[3]: 0

tick3
answer[0]: 1
answer[1]: 0
answer[2]: 0
answer[3]: 0


gatePrecedence/test3
tick1
answer[0]: 1
answer[1]: 0

tick2
answer[0]: 1
answer[1]: 0

tick3
answer[0]: 1
answer[1]: 0


gatePrecedence/test4
tick1
answer[0]: 1

tick2
answer[0]: 1

tick3
answer[0]: 1
```

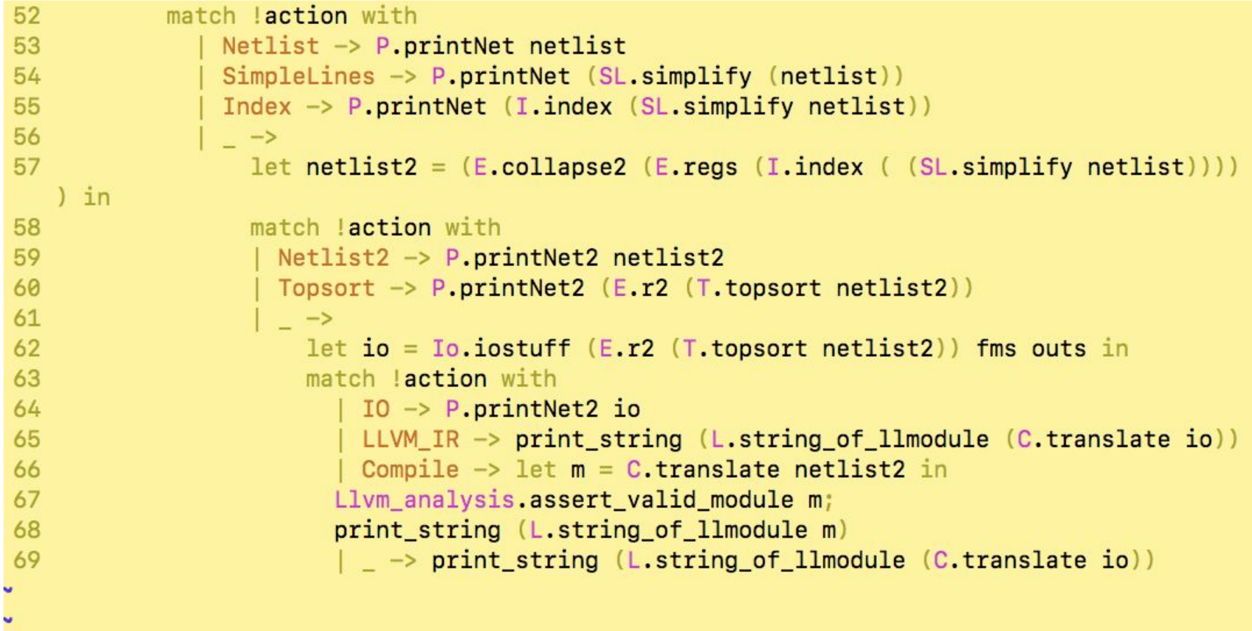
```
===================================================================================
=================================================================================
                                                                       END OF
TESTING!
```

7. Full Code Listing

Listing: unic.ml

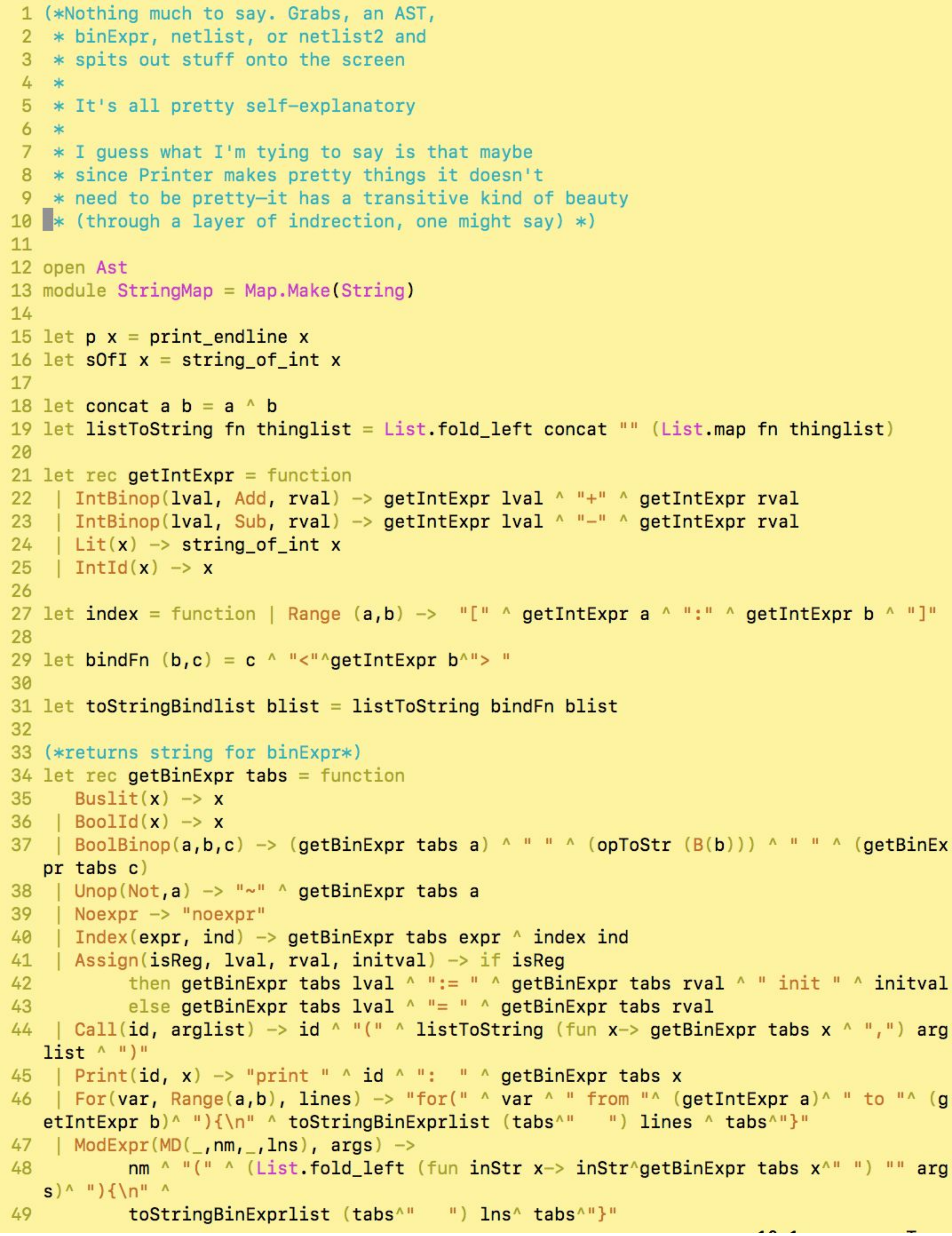


```
module P = Printer
module M = Modfill
module S = Semant
module E = Elaborate
module SL = Simplelines
module L = Llvm
module C = Codegen
module H = Harden
module T = Topsort
module I = Indexing
(* Top-level of the Unic compiler: scan & parse the input,
   check the resulting AST and generate an SAST from it, generate LLVM IR,
   and dump the module *)

type action = Ast | Sast | Mast | Hast | Netlist | SimpleLines | Index | Netlist2 | Topsort | IO | LLVM_IR | Compile

let () =
  let action = ref Compile in
  let set_action a () = action := a in
  let speclist = [
    ("-a",  Arg.Unit (set_action Ast), "Print the AST");
    ("-m",  Arg.Unit (set_action Mast), "Print the modfilled AST");
    ("-h",  Arg.Unit (set_action Hast), "Print the hardened AST");
    ("-s",  Arg.Unit (set_action Sast), "Print the SAST");
    ("-n",  Arg.Unit (set_action Netlist), "Print Netlist");
    ("-sl", Arg.Unit (set_action SimpleLines), "Print Netlist with Simplified Lines");
    ("-i",  Arg.Unit (set_action Index), "Print Netlist with collapsed inidices");
    ("-n2", Arg.Unit (set_action Netlist2), "Print MoreSimplified Netlist");
    ("-t",  Arg.Unit (set_action Topsort), "Print Topsorted Netlist");
    ("-io", Arg.Unit (set_action IO), "Print Topsorted Netlist after IO stuff");
    ("-l",  Arg.Unit (set_action LLVM_IR), "Print the generated LLVM IR");
    ("-c",  Arg.Unit (set_action Compile),
      "Check and print the generated LLVM IR (default)");
  ] in
  let usage_msg = "usage: ./microc.native [-a|-s|-l|-c] [file.mc]" in
  let channel = ref stdin in
  Arg.parse speclist (fun filename -> channel := open_in filename) usage_msg;

  let lexbuf = Lexing.from_channel !channel in
  let ast = Parser.program Scanner.token lexbuf in
  let mast = M.fill ast in

  match !action with
      Ast  -> P.printAst ast
    | Mast -> P.printMast (M.fill ast)
    | Sast -> P.printMast (S.semant (M.fill ast))
    | Hast -> P.printMast (S.semant (M.fill ast))
    | _    ->
        let netlist = E.collapse (S.semant (M.fill ast)) in
        let outs = Io.getOuts mast in
        let fms = Io.getFms mast in
```

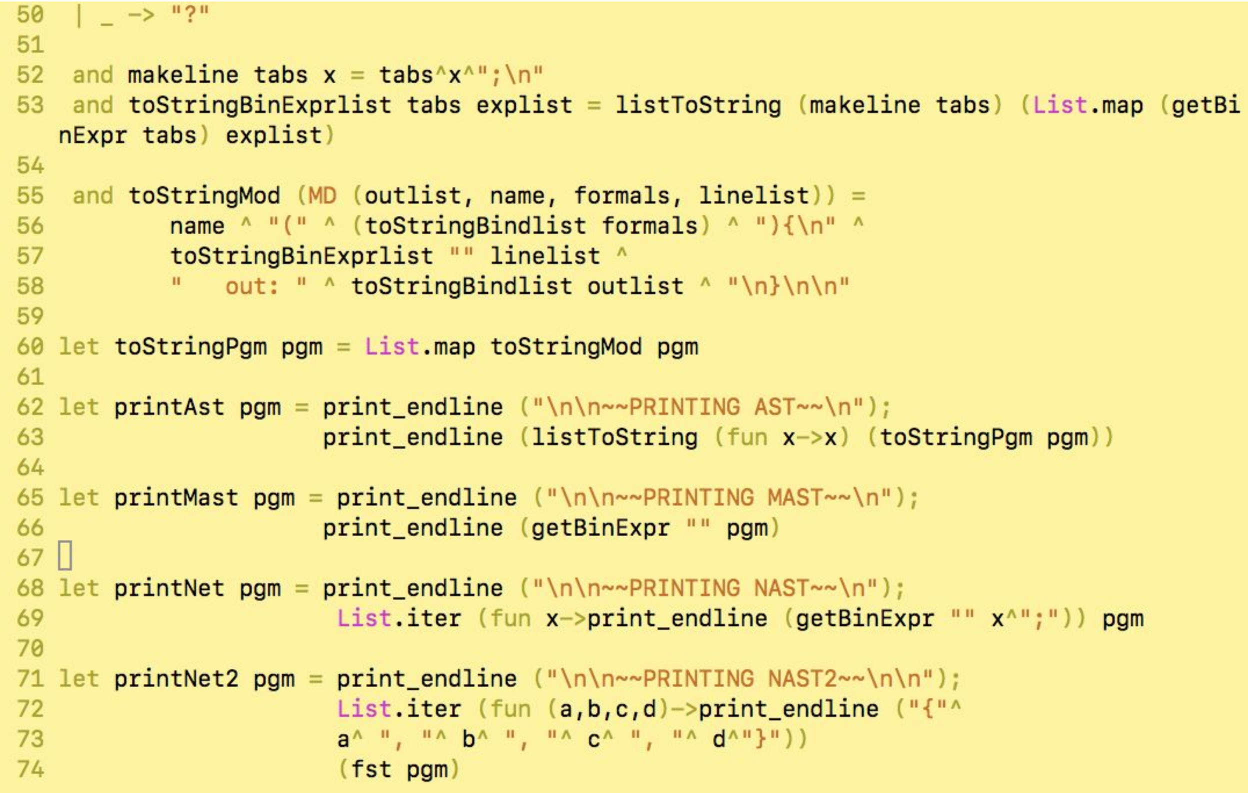


```
        match !action with
          | Netlist -> P.printNet netlist
          | SimpleLines -> P.printNet (SL.simplify (netlist))
          | Index -> P.printNet (I.index (SL.simplify netlist))
          | _ ->
              let netlist2 = (E.collapse2 (E.regs (I.index ( (SL.simplify netlist))))) in
              match !action with
              | Netlist2 -> P.printNet2 netlist2
              | Topsort -> P.printNet2 (E.r2 (T.topsort netlist2))
              | _ ->
                  let io = Io.iostuff (E.r2 (T.topsort netlist2)) fms outs in
                  match !action with
                    | IO -> P.printNet2 io
                    | LLVM_IR -> print_string (L.string_of_llmodule (C.translate io))
                    | Compile -> let m = C.translate netlist2 in
                    Llvm_analysis.assert_valid_module m;
                    print_string (L.string_of_llmodule m)
                    | _ -> print_string (L.string_of_llmodule (C.translate io))
~
~
```

Listing: printer.ml

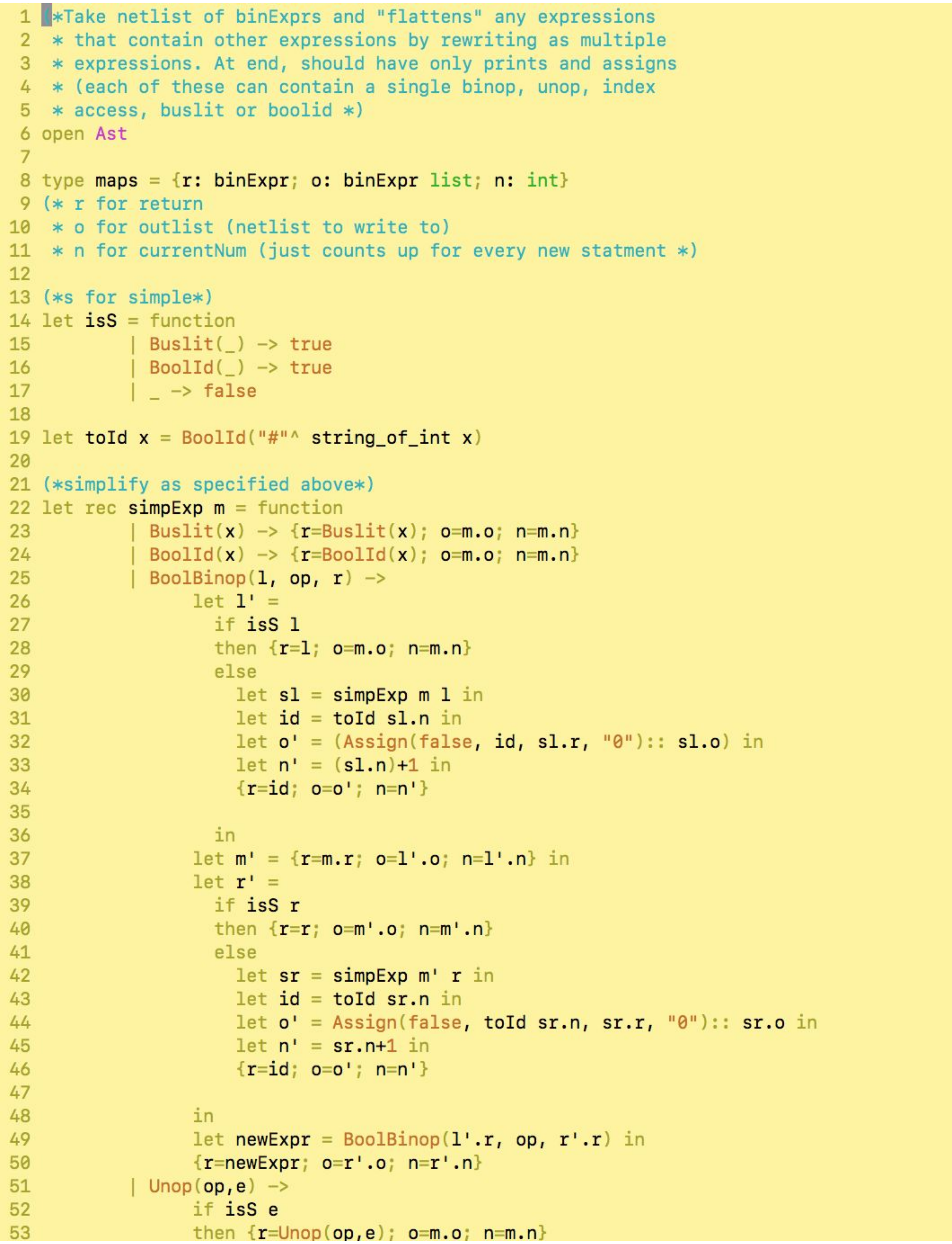


```
(*Nothing much to say. Grabs, an AST,
 * binExpr, netlist, or netlist2 and
 * spits out stuff onto the screen
 *
 * It's all pretty self-explanatory
 *
 * I guess what I'm tying to say is that maybe
 * since Printer makes pretty things it doesn't
 * need to be pretty—it has a transitive kind of beauty
 * (through a layer of indrection, one might say) *)

open Ast
module StringMap = Map.Make(String)

let p x = print_endline x
let sOfI x = string_of_int x

let concat a b = a ^ b
let listToString fn thinglist = List.fold_left concat "" (List.map fn thinglist)

let rec getIntExpr = function
 | IntBinop(lval, Add, rval) -> getIntExpr lval ^ "+" ^ getIntExpr rval
 | IntBinop(lval, Sub, rval) -> getIntExpr lval ^ "-" ^ getIntExpr rval
 | Lit(x) -> string_of_int x
 | IntId(x) -> x

let index = function | Range (a,b) ->  "[" ^ getIntExpr a ^ ":" ^ getIntExpr b ^ "]"

let bindFn (b,c) = c ^ "<"^getIntExpr b^"> "

let toStringBindlist blist = listToString bindFn blist

(*returns string for binExpr*)
let rec getBinExpr tabs = function
   Buslit(x) -> x
 | BoolId(x) -> x
 | BoolBinop(a,b,c) -> (getBinExpr tabs a) ^ " " ^ (opToStr (B(b))) ^ " " ^ (getBinExpr tabs c)
 | Unop(Not,a) -> "~" ^ getBinExpr tabs a
 | Noexpr -> "noexpr"
 | Index(expr, ind) -> getBinExpr tabs expr ^ index ind
 | Assign(isReg, lval, rval, initval) -> if isReg
        then getBinExpr tabs lval ^ ":= " ^ getBinExpr tabs rval ^ " init " ^ initval
        else getBinExpr tabs lval ^ "= " ^ getBinExpr tabs rval
 | Call(id, arglist) -> id ^ "(" ^ listToString (fun x-> getBinExpr tabs x ^ ",") arglist ^ ")"
 | Print(id, x) -> "print " ^ id ^ ":  " ^ getBinExpr tabs x
 | For(var, Range(a,b), lines) -> "for(" ^ var ^ " from "^ (getIntExpr a)^ " to "^ (getIntExpr b)^ "){\n" ^ toStringBinExprlist (tabs^"   ") lines ^ tabs^"}"
 | ModExpr(MD(_,nm,_,lns), args) ->
        nm ^ "(" ^ (List.fold_left (fun inStr x-> inStr^getBinExpr tabs x^" ") "" args)^ "){\n" ^
        toStringBinExprlist (tabs^"   ") lns^ tabs^"}"
```

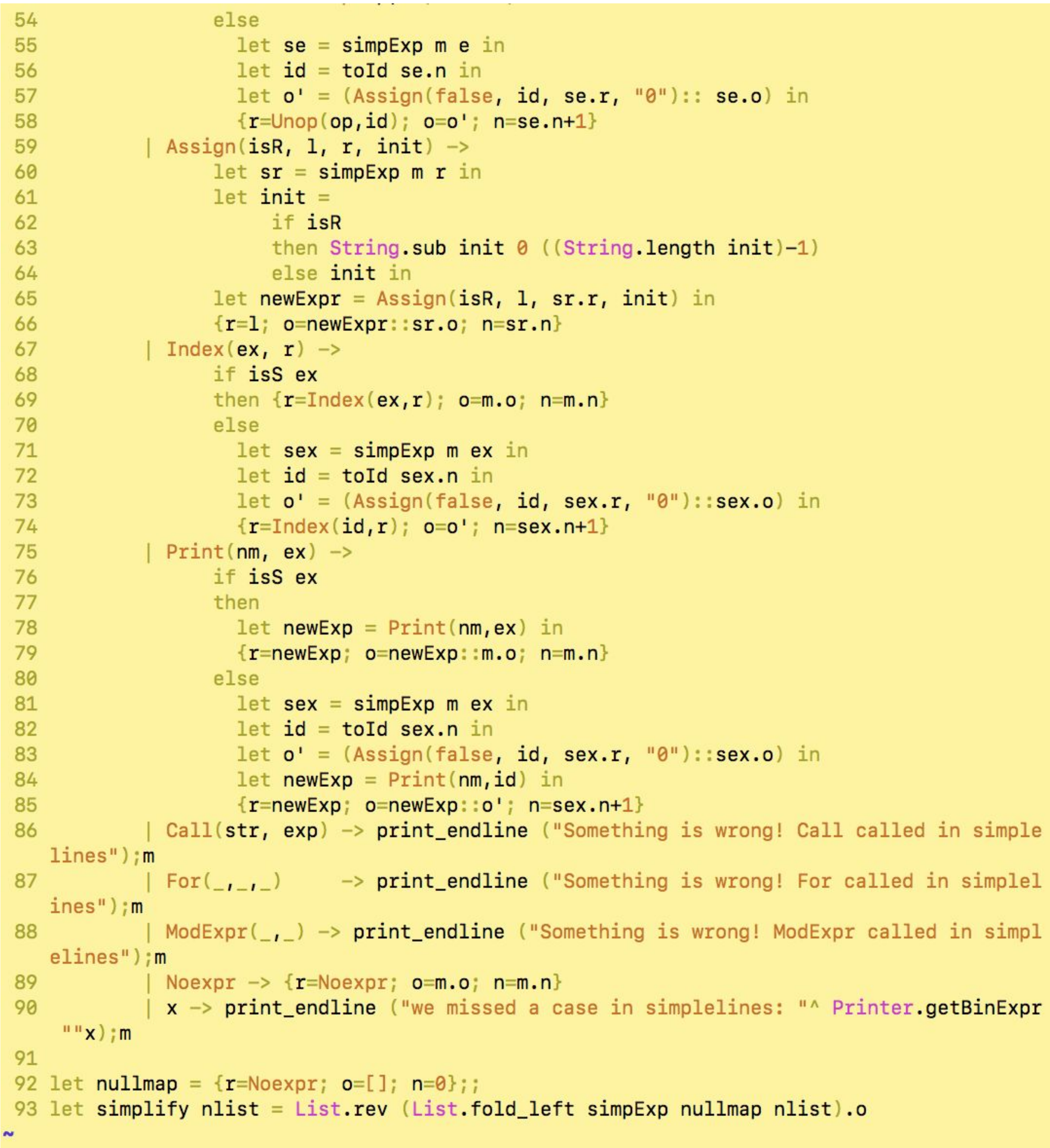


```
 | _ -> "?"

 and makeline tabs x = tabs^x^";\n"
 and toStringBinExprlist tabs explist = listToString (makeline tabs) (List.map (getBinExpr tabs) explist)

 and toStringMod (MD (outlist, name, formals, linelist)) =
        name ^ "(" ^ (toStringBindlist formals) ^ "){\n" ^
        toStringBinExprlist "" linelist ^
        "   out: " ^ toStringBindlist outlist ^ "\n}\n\n"

let toStringPgm pgm = List.map toStringMod pgm

let printAst pgm = print_endline ("\n\n~~PRINTING AST~~\n");
                   print_endline (listToString (fun x->x) (toStringPgm pgm))

let printMast pgm = print_endline ("\n\n~~PRINTING MAST~~\n");
                   print_endline (getBinExpr "" pgm)

let printNet pgm = print_endline ("\n\n~~PRINTING NAST~~\n");
                    List.iter (fun x->print_endline (getBinExpr "" x^";")) pgm

let printNet2 pgm = print_endline ("\n\n~~PRINTING NAST2~~\n\n");
                    List.iter (fun (a,b,c,d)->print_endline ("{"^
                    a^ ", "^ b^ ", "^ c^ ", "^ d^"}"))
                    (fst pgm)
```

Listing: simplelines.ml

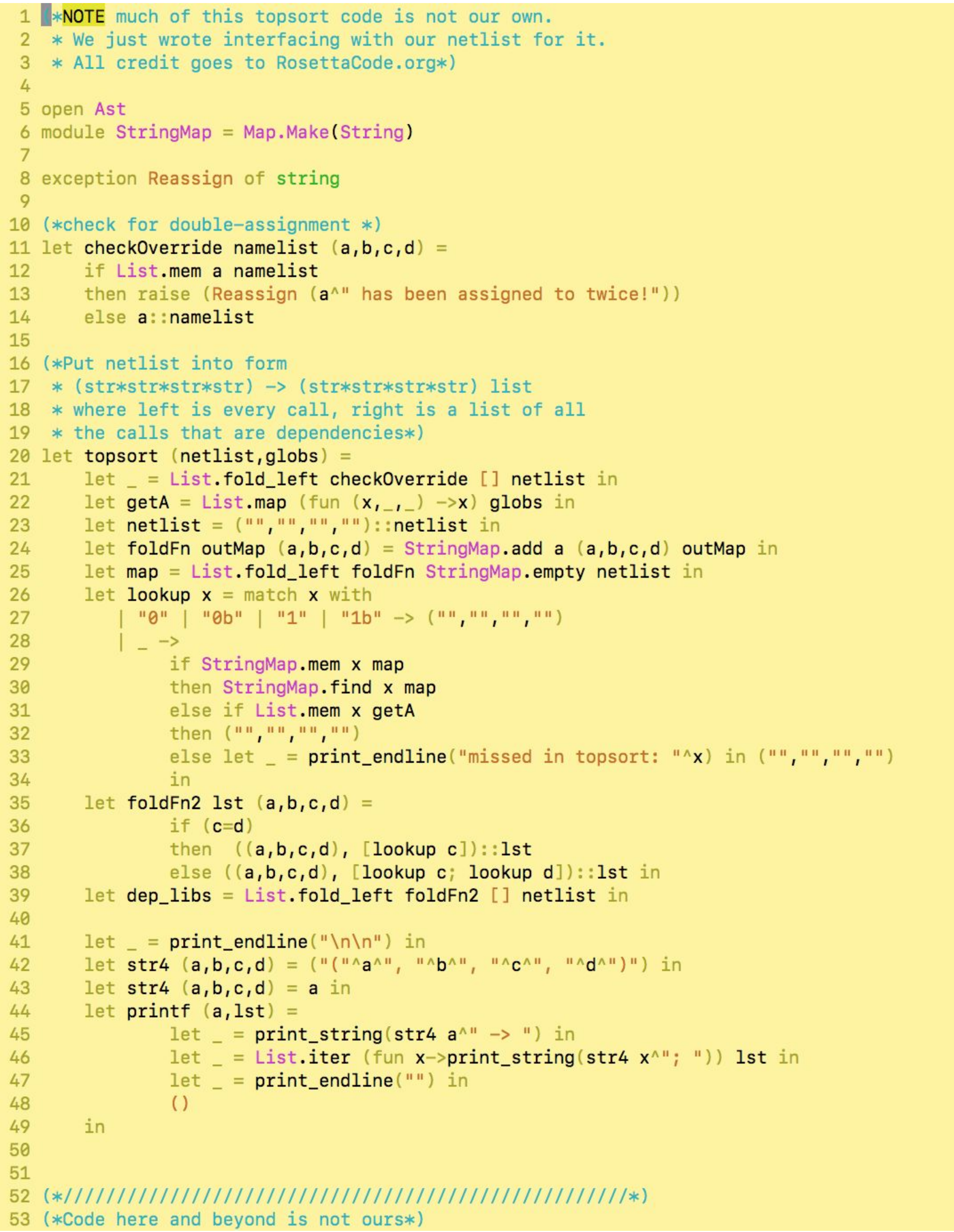


```
(*Take netlist of binExprs and "flattens" any expressions
 * that contain other expressions by rewriting as multiple
 * expressions. At end, should have only prints and assigns
 * (each of these can contain a single binop, unop, index
 * access, buslit or boolid *)
open Ast

type maps = {r: binExpr; o: binExpr list; n: int}
(* r for return
 * o for outlist (netlist to write to)
 * n for currentNum (just counts up for every new statment *)

(*s for simple*)
let isS = function
    | Buslit(_) -> true
    | BoolId(_) -> true
    | _ -> false

let toId x = BoolId("#"^ string_of_int x)

(*simplify as specified above*)
let rec simpExp m = function
    | Buslit(x) -> {r=Buslit(x); o=m.o; n=m.n}
    | BoolId(x) -> {r=BoolId(x); o=m.o; n=m.n}
    | BoolBinop(l, op, r) ->
       let l' =
        if isS l
        then {r=l; o=m.o; n=m.n}
        else
         let sl = simpExp m l in
         let id = toId sl.n in
         let o' = (Assign(false, id, sl.r, "0"):: sl.o) in
         let n' = (sl.n)+1 in
         {r=id; o=o'; n=n'}

        in
       let m' = {r=m.r; o=l'.o; n=l'.n} in
       let r' =
        if isS r
        then {r=r; o=m'.o; n=m'.n}
        else
         let sr = simpExp m' r in
         let id = toId sr.n in
         let o' = Assign(false, toId sr.n, sr.r, "0"):: sr.o in
         let n' = sr.n+1 in
         {r=id; o=o'; n=n'}

       in
       let newExpr = BoolBinop(l'.r, op, r'.r) in
       {r=newExpr; o=r'.o; n=r'.n}
    | Unop(op,e) ->
       if isS e
       then {r=Unop(op,e); o=m.o; n=m.n}
```

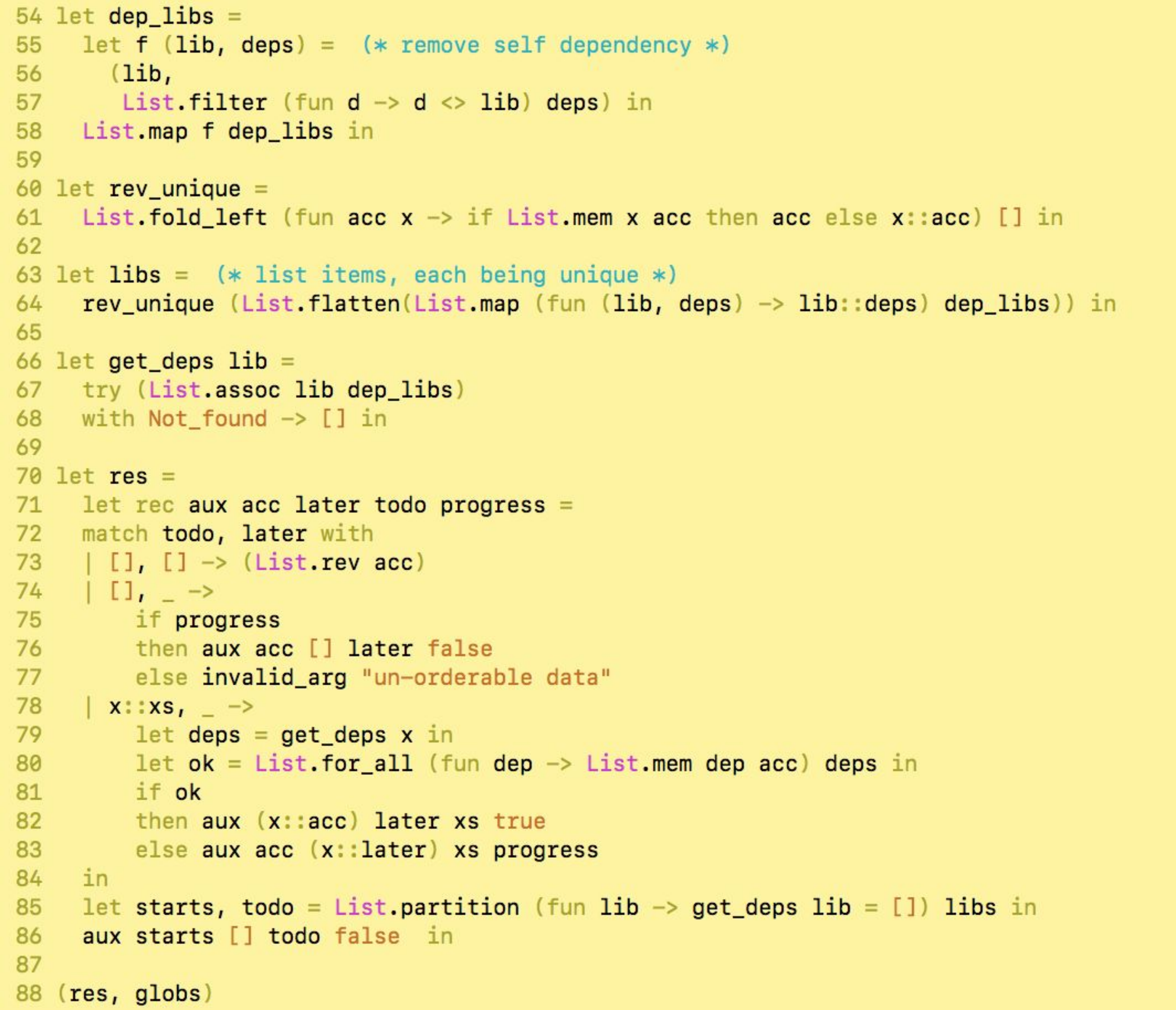


```
              else
                let se = simpExp m e in
                let id = toId se.n in
                let o' = (Assign(false, id, se.r, "0"):: se.o) in
                {r=Unop(op,id); o=o'; n=se.n+1}
        | Assign(isR, l, r, init) ->
              let sr = simpExp m r in
              let init =
                   if isR
                   then String.sub init 0 ((String.length init)-1)
                   else init in
              let newExpr = Assign(isR, l, sr.r, init) in
              {r=l; o=newExpr::sr.o; n=sr.n}
        | Index(ex, r) ->
              if isS ex
              then {r=Index(ex,r); o=m.o; n=m.n}
              else
                let sex = simpExp m ex in
                let id = toId sex.n in
                let o' = (Assign(false, id, sex.r, "0")::sex.o) in
                {r=Index(id,r); o=o'; n=sex.n+1}
        | Print(nm, ex) ->
              if isS ex
              then
                let newExp = Print(nm,ex) in
                {r=newExp; o=newExp::m.o; n=m.n}
              else
                let sex = simpExp m ex in
                let id = toId sex.n in
                let o' = (Assign(false, id, sex.r, "0")::sex.o) in
                let newExp = Print(nm,id) in
                {r=newExp; o=newExp::o'; n=sex.n+1}
        | Call(str, exp) -> print_endline ("Something is wrong! Call called in simplelines");m
        | For(_,_,_)     -> print_endline ("Something is wrong! For called in simplelines");m
        | ModExpr(_,_) -> print_endline ("Something is wrong! ModExpr called in simplelines");m
        | Noexpr -> {r=Noexpr; o=m.o; n=m.n}
        | x -> print_endline ("we missed a case in simplelines: "^ Printer.getBinExpr ""x);m

let nullmap = {r=Noexpr; o=[]; n=0};;
let simplify nlist = List.rev (List.fold_left simpExp nullmap nlist).o
~
```

Listing: topsort.ml

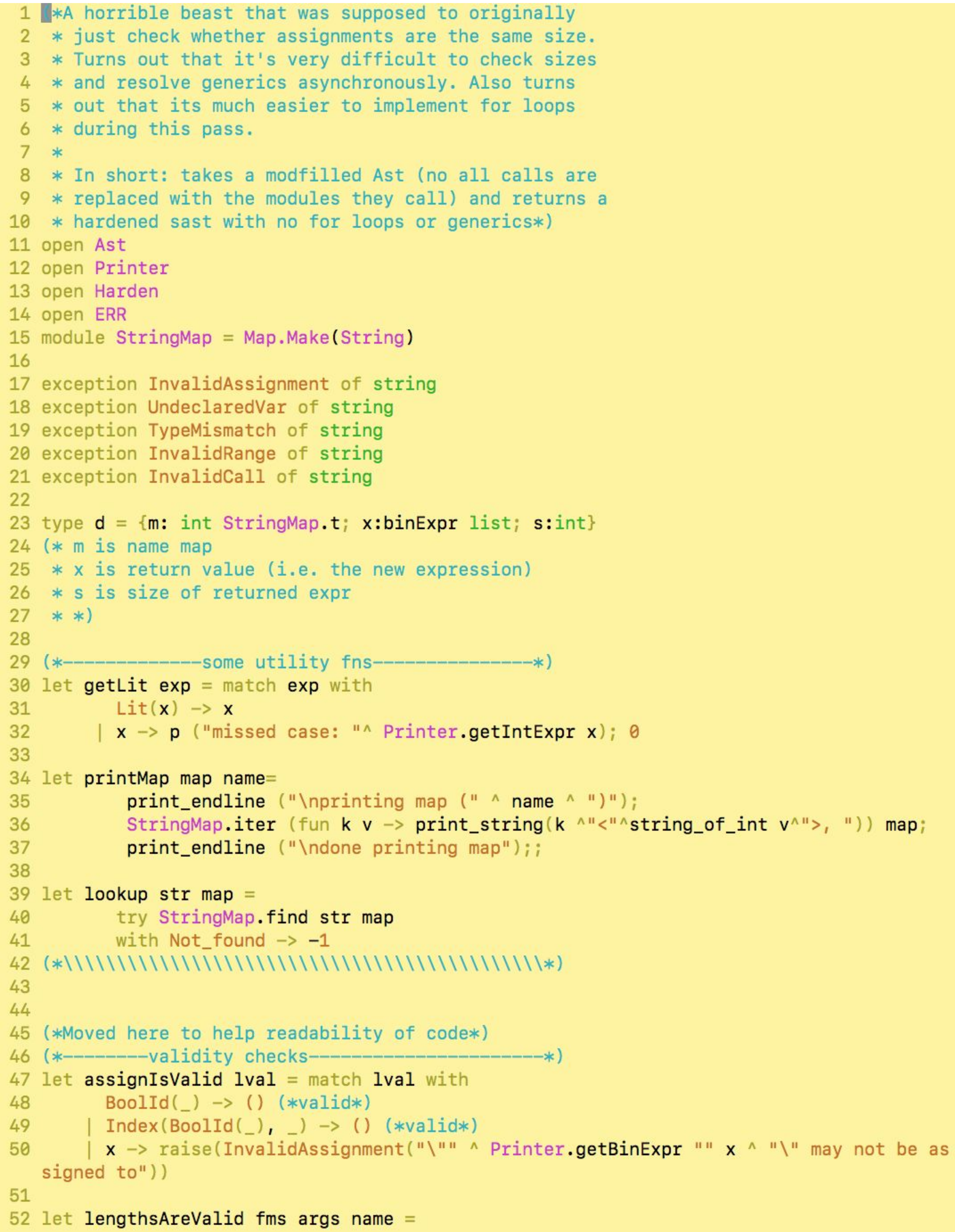


```ocaml
(*NOTE much of this topsort code is not our own.
 * We just wrote interfacing with our netlist for it.
 * All credit goes to RosettaCode.org*)

open Ast
module StringMap = Map.Make(String)

exception Reassign of string

(*check for double-assignment *)
let checkOverride namelist (a,b,c,d) =
    if List.mem a namelist
    then raise (Reassign (a^" has been assigned to twice!"))
    else a::namelist

(*Put netlist into form
 * (str*str*str*str) -> (str*str*str*str) list
 * where left is every call, right is a list of all
 * the calls that are dependencies*)
let topsort (netlist,globs) =
    let _ = List.fold_left checkOverride [] netlist in
    let getA = List.map (fun (x,_,_) ->x) globs in
    let netlist = ("","","","")::netlist in
    let foldFn outMap (a,b,c,d) = StringMap.add a (a,b,c,d) outMap in
    let map = List.fold_left foldFn StringMap.empty netlist in
    let lookup x = match x with
       | "0" | "0b" | "1" | "1b" -> ("","","","")
       | _ ->
            if StringMap.mem x map
            then StringMap.find x map
            else if List.mem x getA
            then ("","","","")
            else let _ = print_endline("missed in topsort: "^x) in ("","","","")
            in
    let foldFn2 lst (a,b,c,d) =
            if (c=d)
            then  ((a,b,c,d), [lookup c])::lst
            else ((a,b,c,d), [lookup c; lookup d])::lst in
    let dep_libs = List.fold_left foldFn2 [] netlist in

    let _ = print_endline("\n\n") in
    let str4 (a,b,c,d) = ("("^a^", "^b^", "^c^", "^d^")") in
    let str4 (a,b,c,d) = a in
    let printf (a,lst) =
            let _ = print_string(str4 a^" -> ") in
            let _ = List.iter (fun x->print_string(str4 x^"; ")) lst in
            let _ = print_endline("") in
            ()
    in


(*////////////////////////////////////////////////////*)
(*Code here and beyond is not ours*)
```

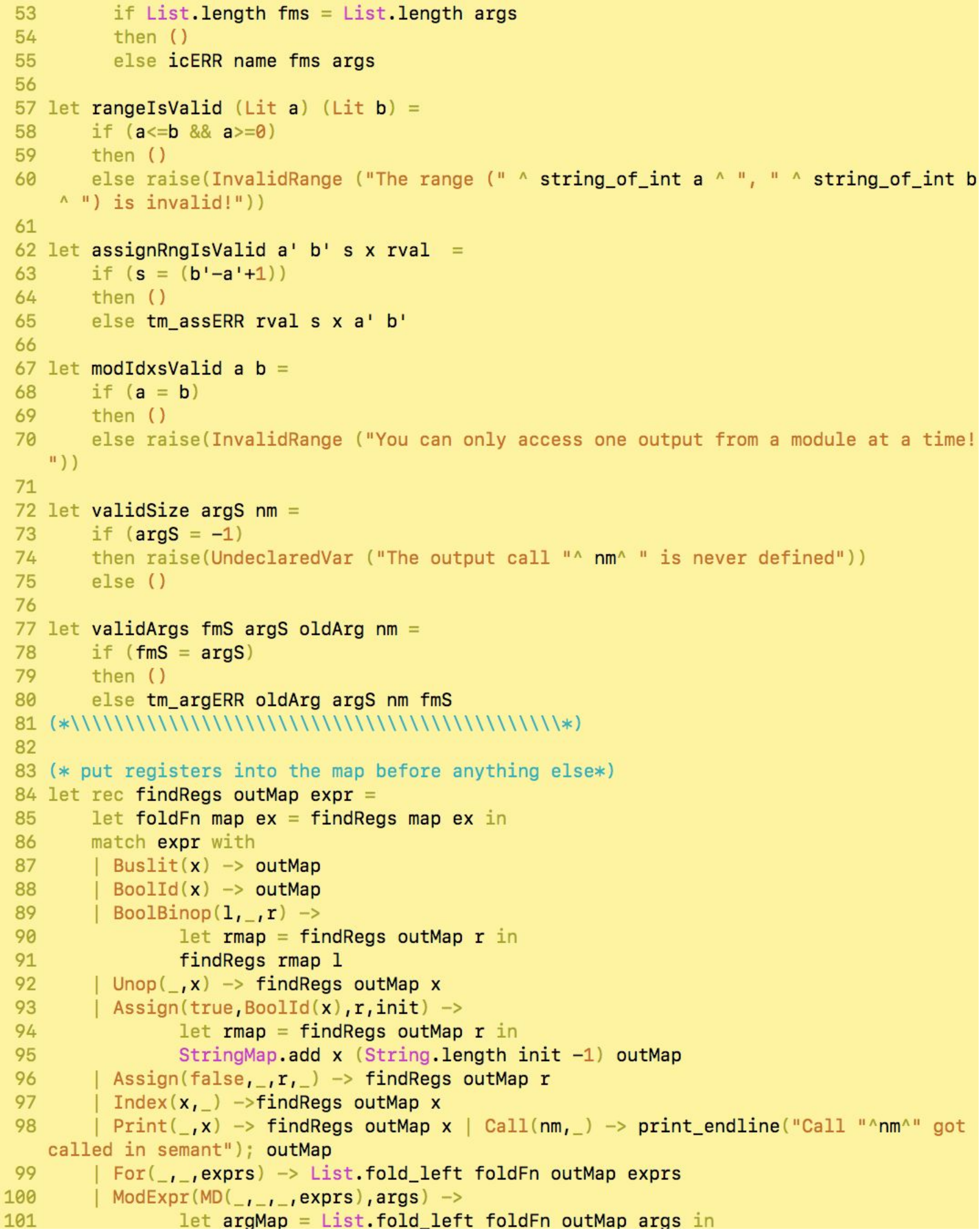


```
let dep_libs =
  let f (lib, deps) =  (* remove self dependency *)
    (lib,
     List.filter (fun d -> d <> lib) deps) in
  List.map f dep_libs in

let rev_unique =
  List.fold_left (fun acc x -> if List.mem x acc then acc else x::acc) [] in

let libs =  (* list items, each being unique *)
  rev_unique (List.flatten(List.map (fun (lib, deps) -> lib::deps) dep_libs)) in

let get_deps lib =
  try (List.assoc lib dep_libs)
  with Not_found -> [] in

let res =
  let rec aux acc later todo progress =
  match todo, later with
  | [], [] -> (List.rev acc)
  | [], _ ->
      if progress
      then aux acc [] later false
      else invalid_arg "un-orderable data"
  | x::xs, _ ->
      let deps = get_deps x in
      let ok = List.for_all (fun dep -> List.mem dep acc) deps in
      if ok
      then aux (x::acc) later xs true
      else aux acc (x::later) xs progress
  in
  let starts, todo = List.partition (fun lib -> get_deps lib = []) libs in
  aux starts [] todo false  in

(res, globs)
```

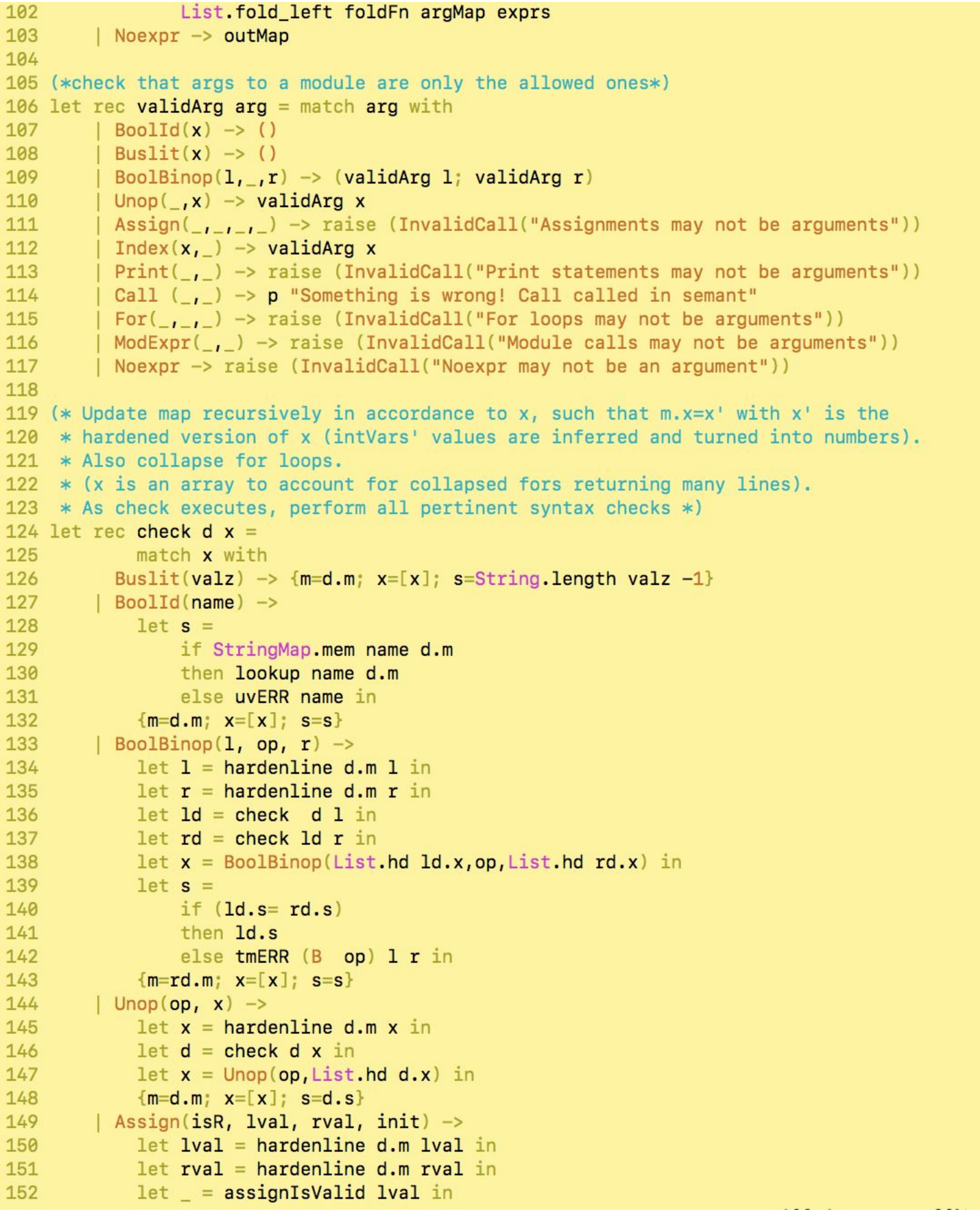
Listing: semant2.ml



```
(*A horrible beast that was supposed to originally
 * just check whether assignments are the same size.
 * Turns out that it's very difficult to check sizes
 * and resolve generics asynchronously. Also turns
 * out that its much easier to implement for loops
 * during this pass.
 *
 * In short: takes a modfilled Ast (no all calls are
 * replaced with the modules they call) and returns a
 * hardened sast with no for loops or generics*)
open Ast
open Printer
open Harden
open ERR
module StringMap = Map.Make(String)

exception InvalidAssignment of string
exception UndeclaredVar of string
exception TypeMismatch of string
exception InvalidRange of string
exception InvalidCall of string

type d = {m: int StringMap.t; x:binExpr list; s:int}
(* m is name map
 * x is return value (i.e. the new expression)
 * s is size of returned expr
 * *)

(*-------------some utility fns--------------*)
let getLit exp = match exp with
       Lit(x) -> x
     | x -> p ("missed case: "^ Printer.getIntExpr x); 0

let printMap map name=
        print_endline ("\nprinting map (" ^ name ^ ")");
        StringMap.iter (fun k v -> print_string(k ^"<"^string_of_int v^">, ")) map;
        print_endline ("\ndone printing map");;

let lookup str map =
       try StringMap.find str map
       with Not_found -> -1
(*\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*)


(*Moved here to help readability of code*)
(*--------validity checks--------------------*)
let assignIsValid lval = match lval with
      BoolId(_) -> () (*valid*)
    | Index(BoolId(_), _) -> () (*valid*)
    | x -> raise(InvalidAssignment("\"" ^ Printer.getBinExpr "" x ^ "\" may not be assigned to"))

let lengthsAreValid fms args name =
```

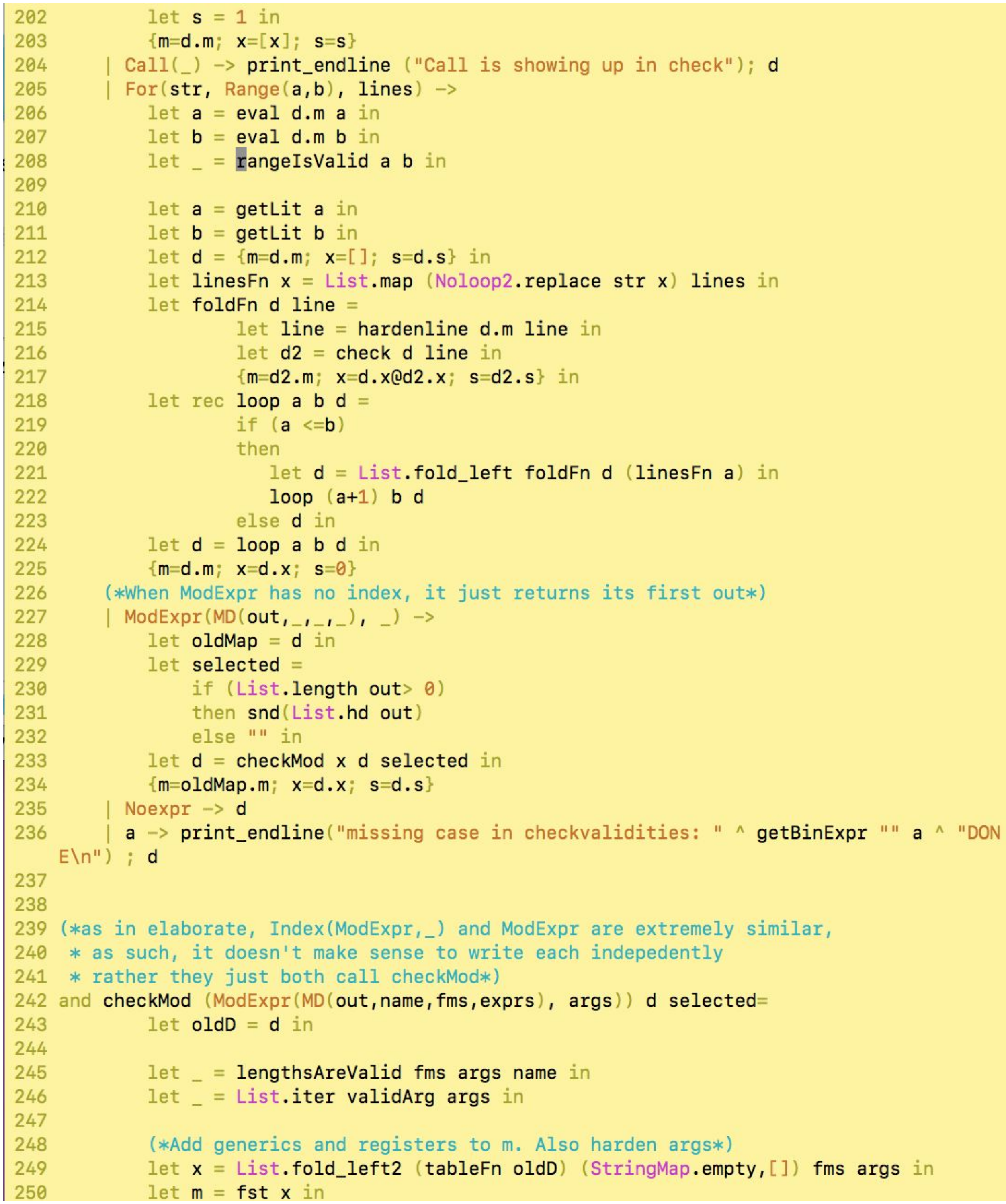


```
   if List.length fms = List.length args
   then ()
   else icERR name fms args

let rangeIsValid (Lit a) (Lit b) =
  if (a<=b && a>=0)
  then ()
  else raise(InvalidRange ("The range (" ^ string_of_int a ^ ", " ^ string_of_int b ^ ") is invalid!"))

let assignRngIsValid a' b' s x rval  =
  if (s = (b'-a'+1))
  then ()
  else tm_assERR rval s x a' b'

let modIdxsValid a b =
  if (a = b)
  then ()
  else raise(InvalidRange ("You can only access one output from a module at a time!"))

let validSize argS nm =
  if (argS = -1)
  then raise(UndeclaredVar ("The output call "^ nm^ " is never defined"))
  else ()

let validArgs fmS argS oldArg nm =
  if (fmS = argS)
  then ()
  else tm_argERR oldArg argS nm fmS
(*\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\*)

(* put registers into the map before anything else*)
let rec findRegs outMap expr =
  let foldFn map ex = findRegs map ex in
  match expr with
  | Buslit(x) -> outMap
  | BoolId(x) -> outMap
  | BoolBinop(l,_,r) ->
      let rmap = findRegs outMap r in
      findRegs rmap l
  | Unop(_,x) -> findRegs outMap x
  | Assign(true,BoolId(x),r,init) ->
      let rmap = findRegs outMap r in
      StringMap.add x (String.length init -1) outMap
  | Assign(false,_,r,_) -> findRegs outMap r
  | Index(x,_) ->findRegs outMap x
  | Print(_,x) -> findRegs outMap x | Call(nm,_) -> print_endline("Call "^nm^" got called in semant"); outMap
  | For(_,_,exprs) -> List.fold_left foldFn outMap exprs
  | ModExpr(MD(_,_,_,exprs),args) ->
      let argMap = List.fold_left foldFn outMap args in
```

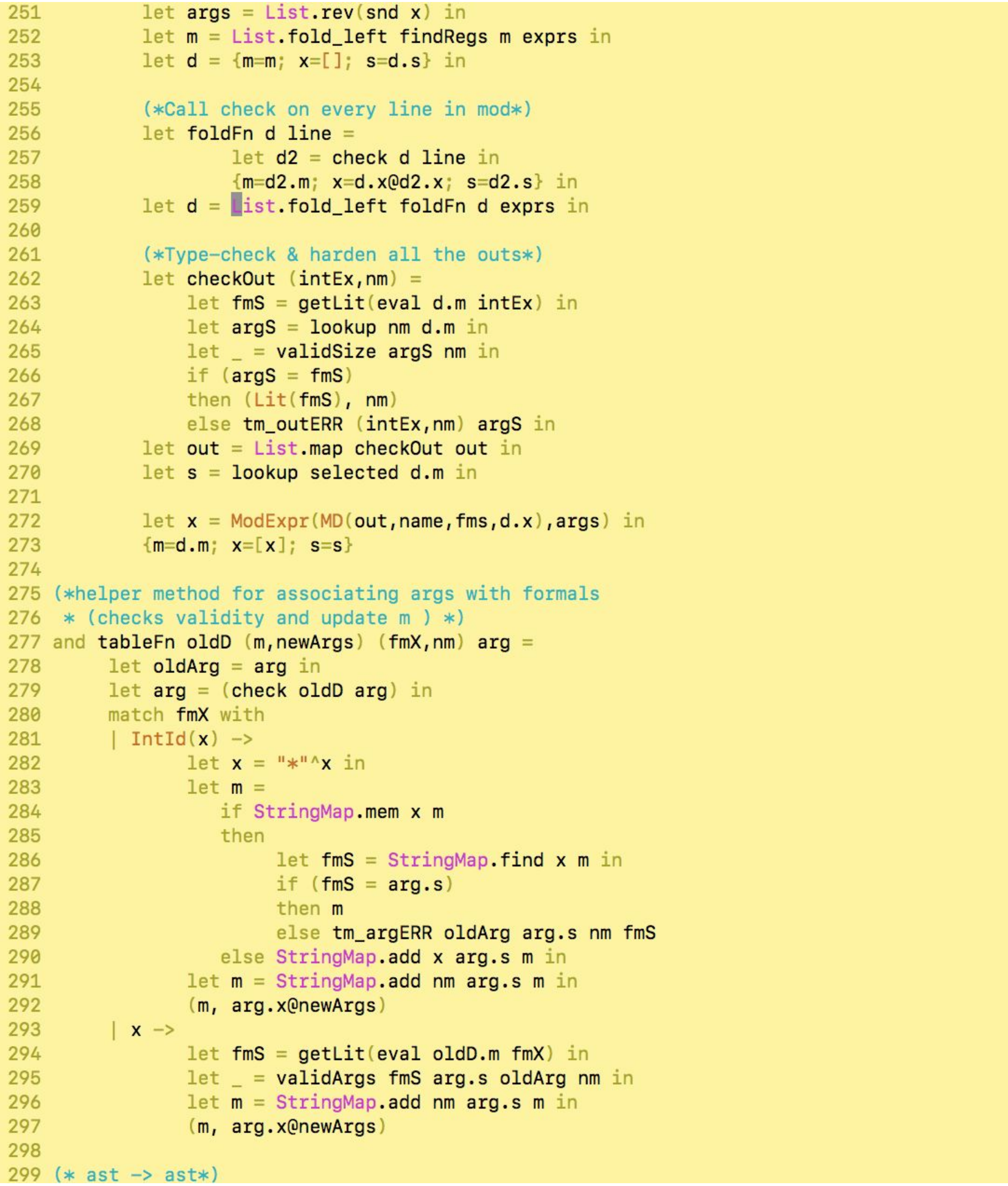


```
      List.fold_left foldFn argMap exprs
  | Noexpr -> outMap

(*check that args to a module are only the allowed ones*)
let rec validArg arg = match arg with
  | BoolId(x) -> ()
  | Buslit(x) -> ()
  | BoolBinop(l,_,r) -> (validArg l; validArg r)
  | Unop(_,x) -> validArg x
  | Assign(_,_,_,_) -> raise (InvalidCall("Assignments may not be arguments"))
  | Index(x,_) -> validArg x
  | Print(_,_) -> raise (InvalidCall("Print statements may not be arguments"))
  | Call (_,_) -> p "Something is wrong! Call called in semant"
  | For(_,_,_) -> raise (InvalidCall("For loops may not be arguments"))
  | ModExpr(_,_) -> raise (InvalidCall("Module calls may not be arguments"))
  | Noexpr -> raise (InvalidCall("Noexpr may not be an argument"))

(* Update map recursively in accordance to x, such that m.x=x' with x' is the
 * hardened version of x (intVars' values are inferred and turned into numbers).
 * Also collapse for loops.
 * (x is an array to account for collapsed fors returning many lines).
 * As check executes, perform all pertinent syntax checks *)
let rec check d x =
    match x with
   Buslit(valz) -> {m=d.m; x=[x]; s=String.length valz -1}
  | BoolId(name) ->
    let s =
      if StringMap.mem name d.m
      then lookup name d.m
      else uvERR name in
    {m=d.m; x=[x]; s=s}
  | BoolBinop(l, op, r) ->
    let l = hardenline d.m l in
    let r = hardenline d.m r in
    let ld = check  d l in
    let rd = check ld r in
    let x = BoolBinop(List.hd ld.x,op,List.hd rd.x) in
    let s =
      if (ld.s= rd.s)
      then ld.s
      else tmERR (B  op) l r in
    {m=rd.m; x=[x]; s=s}
  | Unop(op, x) ->
    let x = hardenline d.m x in
    let d = check d x in
    let x = Unop(op,List.hd d.x) in
    {m=d.m; x=[x]; s=d.s}
  | Assign(isR, lval, rval, init) ->
    let lval = hardenline d.m lval in
    let rval = hardenline d.m rval in
    let _ = assignIsValid lval in
```

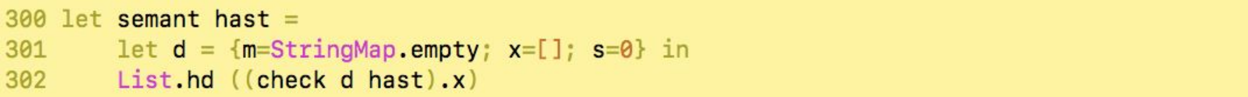
```
        let rd = check d rval in
        let d =
          (match lval with
          | BoolId(x) ->
             let m = StringMap.add x rd.s rd.m in
             {m=m; x=[]; s=rd.s}
          | Index(BoolId(x), Range(a,b)) ->
             let b' = getLit b in
             let a' = getLit a in
             let _ = assignRngIsValid a' b' rd.s x rval in
             let s = max (lookup x rd.m) (b'+1) in
             let m = StringMap.add x s rd.m in
             {m=m; x=[]; s=s}
          ) in
        let x = Assign(isR,lval,List.hd rd.x,init) in
        {m=d.m; x=[x]; s=d.s}
    | Index(ModExpr(MD(outs,nm,fm,lns),args), Range(a,b)) ->
        let oldMap = d in
        let _ = modIdxsValid a b in
        let selected =
          match a with
          | Lit(x) ->
                if (List.length outs > x)
                then snd(List.nth outs x)
                else irERR x nm outs
          | IntId(x) ->
                if (List.exists (fun (_,a) -> a=x) outs)
                then x
                else raise(InvalidRange("ERROR: You tried to access "^x^" but "^nm^" has no such outputs!")) in
        let x = ModExpr(MD(outs,nm,fm,lns),args) in
        let d = checkMod x d selected in
        let x = Index(List.hd d.x,Range(a,b)) in
        {m=oldMap.m; x=[x]; s=d.s}
    | Index(x, Range(a,b)) ->
        let a' = getLit (eval d.m a) in
        let b' = getLit (eval d.m b) in
        let _ = rangeIsValid (Lit(a')) (Lit(b')) in
        let x = hardenline d.m x in
        let d = check d x in
        let s =
            if (d.s > b')
            then b'-a'+1
            else raise(TypeMismatch ("ERROR: You tried accessing the "^sOfI b'^"th element of a "^sOfI d.s^"-bit bus")) in
        let x = Index(x, Range(Lit(a'),Lit(b'))) in
        {m=d.m; x=[x]; s=s}
    | Print(nm, x) ->
        let x = hardenline d.m x in
        let d = check d x in
        let x = Print(nm, List.hd d.x) in
```



```
    let s = 1 in
    {m=d.m; x=[x]; s=s}
  | Call(_) -> print_endline ("Call is showing up in check"); d
  | For(str, Range(a,b), lines) ->
    let a = eval d.m a in
    let b = eval d.m b in
    let _ = rangeIsValid a b in

    let a = getLit a in
    let b = getLit b in
    let d = {m=d.m; x=[]; s=d.s} in
    let linesFn x = List.map (Noloop2.replace str x) lines in
    let foldFn d line =
        let line = hardenline d.m line in
        let d2 = check d line in
        {m=d2.m; x=d.x@d2.x; s=d2.s} in
    let rec loop a b d =
        if (a <=b)
        then
          let d = List.fold_left foldFn d (linesFn a) in
          loop (a+1) b d
        else d in
    let d = loop a b d in
    {m=d.m; x=d.x; s=0}
  (*When ModExpr has no index, it just returns its first out*)
  | ModExpr(MD(out,_,_,_), _) ->
    let oldMap = d in
    let selected =
      if (List.length out> 0)
      then snd(List.hd out)
      else "" in
    let d = checkMod x d selected in
    {m=oldMap.m; x=d.x; s=d.s}
  | Noexpr -> d
  | a -> print_endline("missing case in checkvalidities: " ^ getBinExpr "" a ^ "DONE\n") ; d


(*as in elaborate, Index(ModExpr,_) and ModExpr are extremely similar,
 * as such, it doesn't make sense to write each indepedently
 * rather they just both call checkMod*)
and checkMod (ModExpr(MD(out,name,fms,exprs), args)) d selected=
    let oldD = d in

    let _ = lengthsAreValid fms args name in
    let _ = List.iter validArg args in

    (*Add generics and registers to m. Also harden args*)
    let x = List.fold_left2 (tableFn oldD) (StringMap.empty,[]) fms args in
    let m = fst x in
```

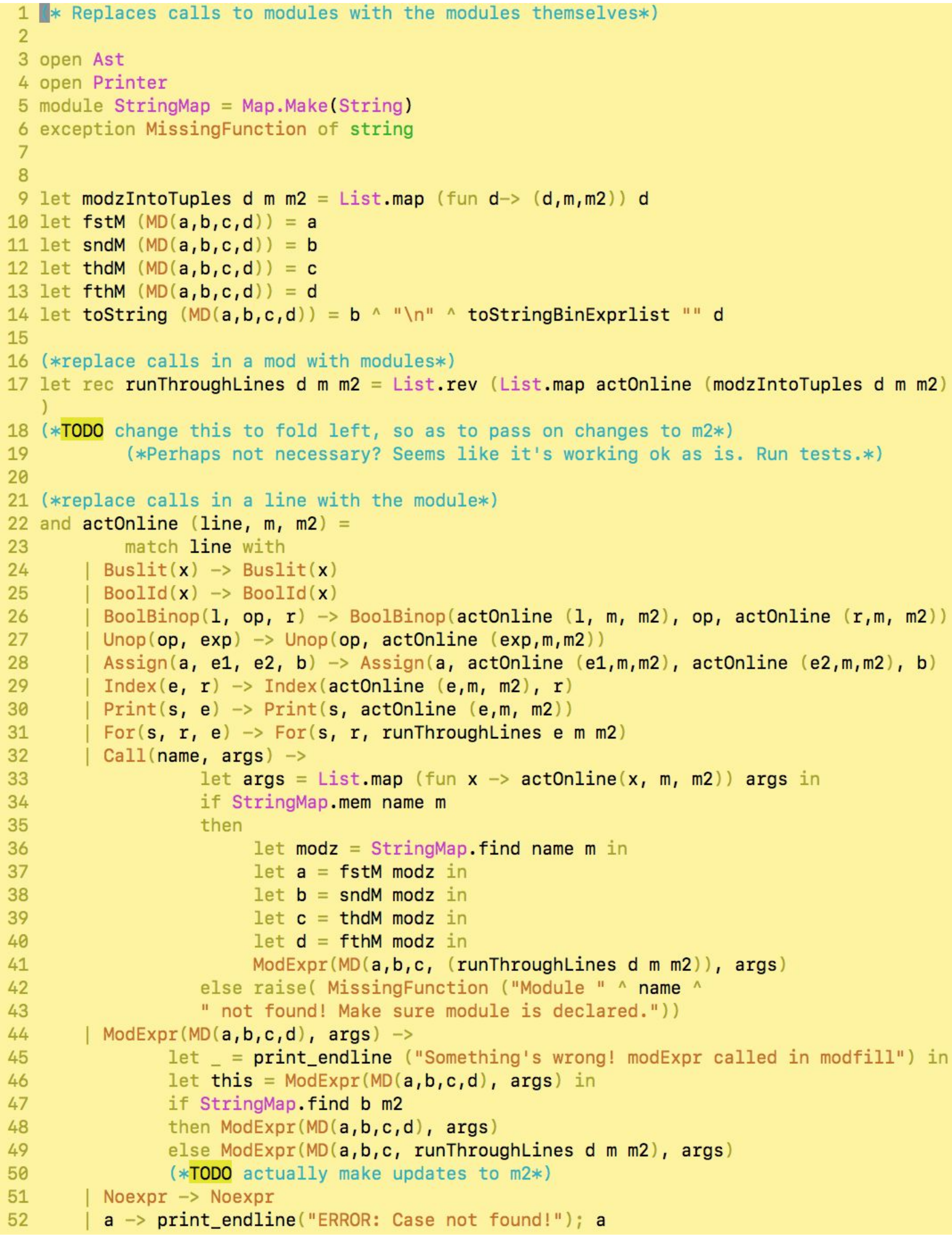


```
        let args = List.rev(snd x) in
        let m = List.fold_left findRegs m exprs in
        let d = {m=m; x=[]; s=d.s} in

        (*Call check on every line in mod*)
        let foldFn d line =
                let d2 = check d line in
                {m=d2.m; x=d.x@d2.x; s=d2.s} in
        let d = List.fold_left foldFn d exprs in

        (*Type-check & harden all the outs*)
        let checkOut (intEx,nm) =
            let fmS = getLit(eval d.m intEx) in
            let argS = lookup nm d.m in
            let _ = validSize argS nm in
            if (argS = fmS)
            then (Lit(fmS), nm)
            else tm_outERR (intEx,nm) argS in
        let out = List.map checkOut out in
        let s = lookup selected d.m in

        let x = ModExpr(MD(out,name,fms,d.x),args) in
        {m=d.m; x=[x]; s=s}

(*helper method for associating args with formals
 * (checks validity and update m ) *)
and tableFn oldD (m,newArgs) (fmX,nm) arg =
     let oldArg = arg in
     let arg = (check oldD arg) in
     match fmX with
     | IntId(x) ->
            let x = "*"^x in
            let m =
               if StringMap.mem x m
               then
                    let fmS = StringMap.find x m in
                    if (fmS = arg.s)
                    then m
                    else tm_argERR oldArg arg.s nm fmS
               else StringMap.add x arg.s m in
            let m = StringMap.add nm arg.s m in
            (m, arg.x@newArgs)
     | x ->
            let fmS = getLit(eval oldD.m fmX) in
            let _ = validArgs fmS arg.s oldArg nm in
            let m = StringMap.add nm arg.s m in
            (m, arg.x@newArgs)

(* ast -> ast*)
```

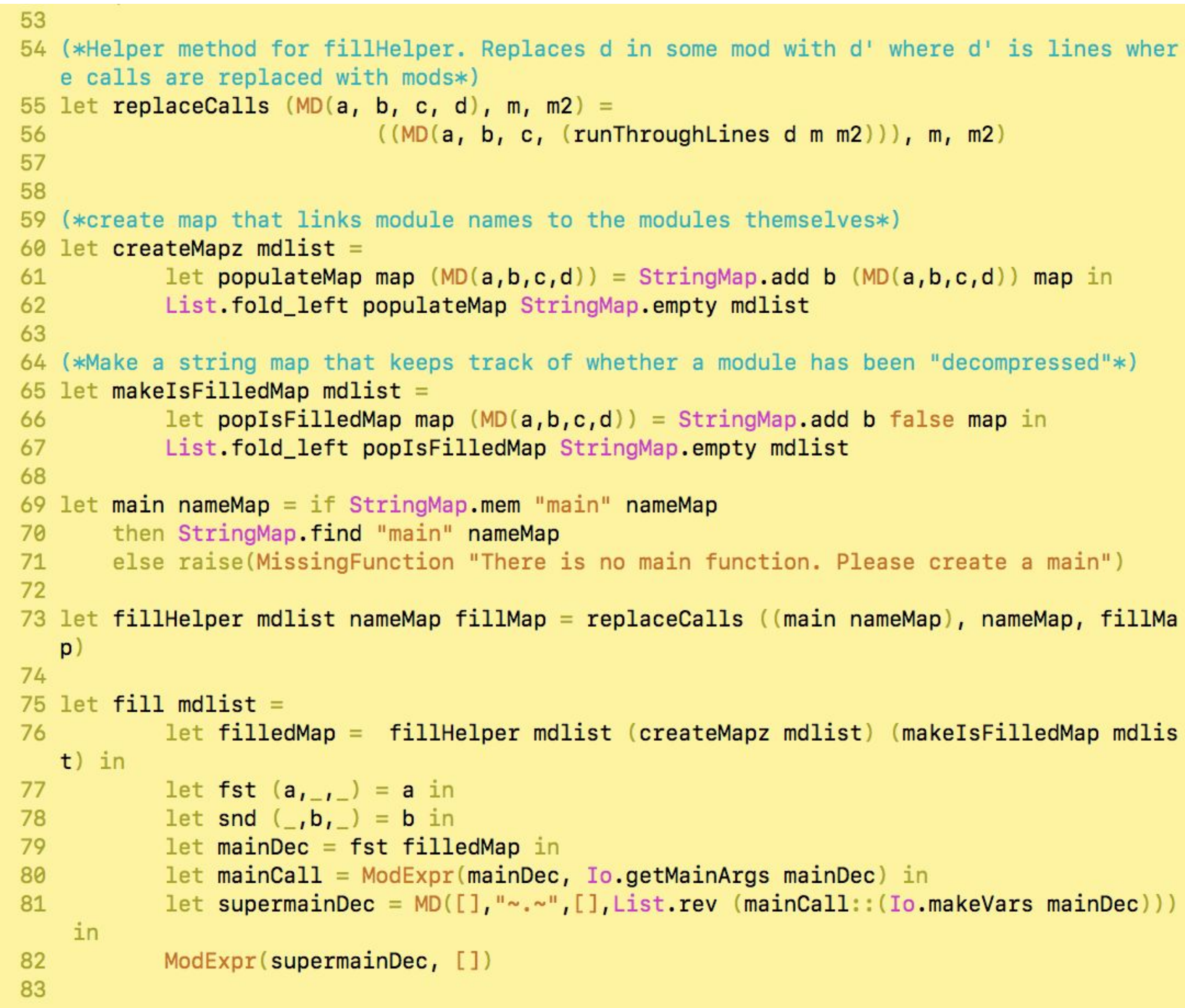


```
let semant hast =
    let d = {m=StringMap.empty; x=[]; s=0} in
    List.hd ((check d hast).x)
```

Listing: modfill.ml

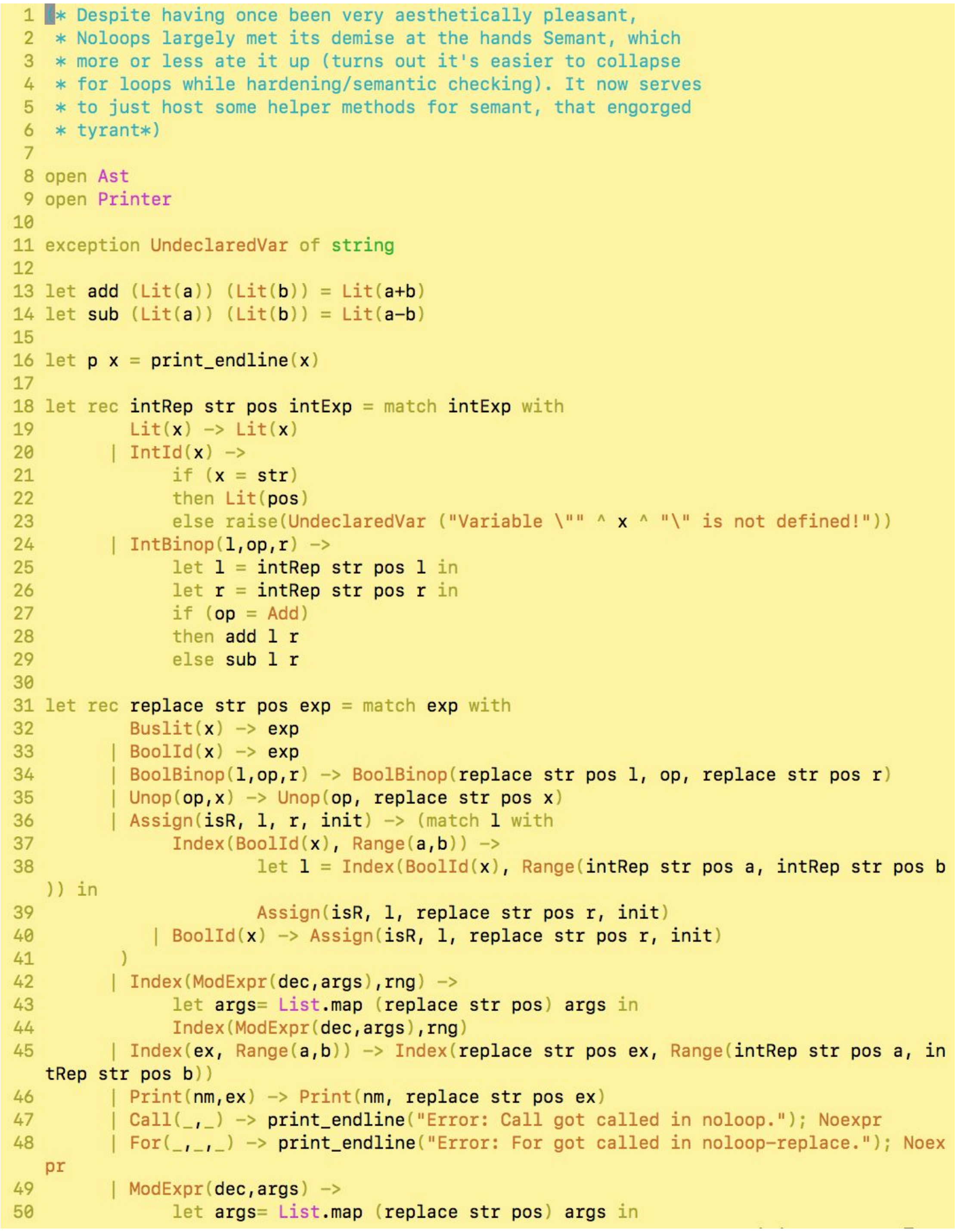


```
(* Replaces calls to modules with the modules themselves*)

open Ast
open Printer
module StringMap = Map.Make(String)
exception MissingFunction of string


let modzIntoTuples d m m2 = List.map (fun d-> (d,m,m2)) d
let fstM (MD(a,b,c,d)) = a
let sndM (MD(a,b,c,d)) = b
let thdM (MD(a,b,c,d)) = c
let fthM (MD(a,b,c,d)) = d
let toString (MD(a,b,c,d)) = b ^ "\n" ^ toStringBinExprlist "" d

(*replace calls in a mod with modules*)
let rec runThroughLines d m m2 = List.rev (List.map actOnline (modzIntoTuples d m m2))
(*TODO change this to fold left, so as to pass on changes to m2*)
        (*Perhaps not necessary? Seems like it's working ok as is. Run tests.*)

(*replace calls in a line with the module*)
and actOnline (line, m, m2) =
        match line with
    | Buslit(x) -> Buslit(x)
    | BoolId(x) -> BoolId(x)
    | BoolBinop(l, op, r) -> BoolBinop(actOnline (l, m, m2), op, actOnline (r,m, m2))
    | Unop(op, exp) -> Unop(op, actOnline (exp,m,m2))
    | Assign(a, e1, e2, b) -> Assign(a, actOnline (e1,m,m2), actOnline (e2,m,m2), b)
    | Index(e, r) -> Index(actOnline (e,m, m2), r)
    | Print(s, e) -> Print(s, actOnline (e,m, m2))
    | For(s, r, e) -> For(s, r, runThroughLines e m m2)
    | Call(name, args) ->
               let args = List.map (fun x -> actOnline(x, m, m2)) args in
               if StringMap.mem name m
               then
                    let modz = StringMap.find name m in
                    let a = fstM modz in
                    let b = sndM modz in
                    let c = thdM modz in
                    let d = fthM modz in
                    ModExpr(MD(a,b,c, (runThroughLines d m m2)), args)
               else raise( MissingFunction ("Module " ^ name ^
               " not found! Make sure module is declared."))
    | ModExpr(MD(a,b,c,d), args) ->
            let _ = print_endline ("Something's wrong! modExpr called in modfill") in
            let this = ModExpr(MD(a,b,c,d), args) in
            if StringMap.find b m2
            then ModExpr(MD(a,b,c,d), args)
            else ModExpr(MD(a,b,c, runThroughLines d m m2), args)
            (*TODO actually make updates to m2*)
    | Noexpr -> Noexpr
    | a -> print_endline("ERROR: Case not found!"); a
```

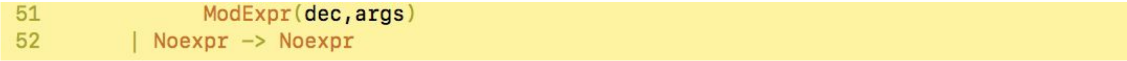


```

(*Helper method for fillHelper. Replaces d in some mod with d' where d' is lines where calls are replaced with mods*)
let replaceCalls (MD(a, b, c, d), m, m2) =
            ((MD(a, b, c, (runThroughLines d m m2))), m, m2)


(*create map that links module names to the modules themselves*)
let createMapz mdlist =
    let populateMap map (MD(a,b,c,d)) = StringMap.add b (MD(a,b,c,d)) map in
    List.fold_left populateMap StringMap.empty mdlist

(*Make a string map that keeps track of whether a module has been "decompressed"*)
let makeIsFilledMap mdlist =
    let popIsFilledMap map (MD(a,b,c,d)) = StringMap.add b false map in
    List.fold_left popIsFilledMap StringMap.empty mdlist

let main nameMap = if StringMap.mem "main" nameMap
  then StringMap.find "main" nameMap
  else raise(MissingFunction "There is no main function. Please create a main")

let fillHelper mdlist nameMap fillMap = replaceCalls ((main nameMap), nameMap, fillMap)

let fill mdlist =
    let filledMap =  fillHelper mdlist (createMapz mdlist) (makeIsFilledMap mdlist) in
    let fst (a,_,_) = a in
    let snd (_,b,_) = b in
    let mainDec = fst filledMap in
    let mainCall = ModExpr(mainDec, Io.getMainArgs mainDec) in
    let supermainDec = MD([],"~.~",[],List.rev (mainCall::(Io.makeVars mainDec))) in
    ModExpr(supermainDec, [])

```

Listing: noloop2.ml

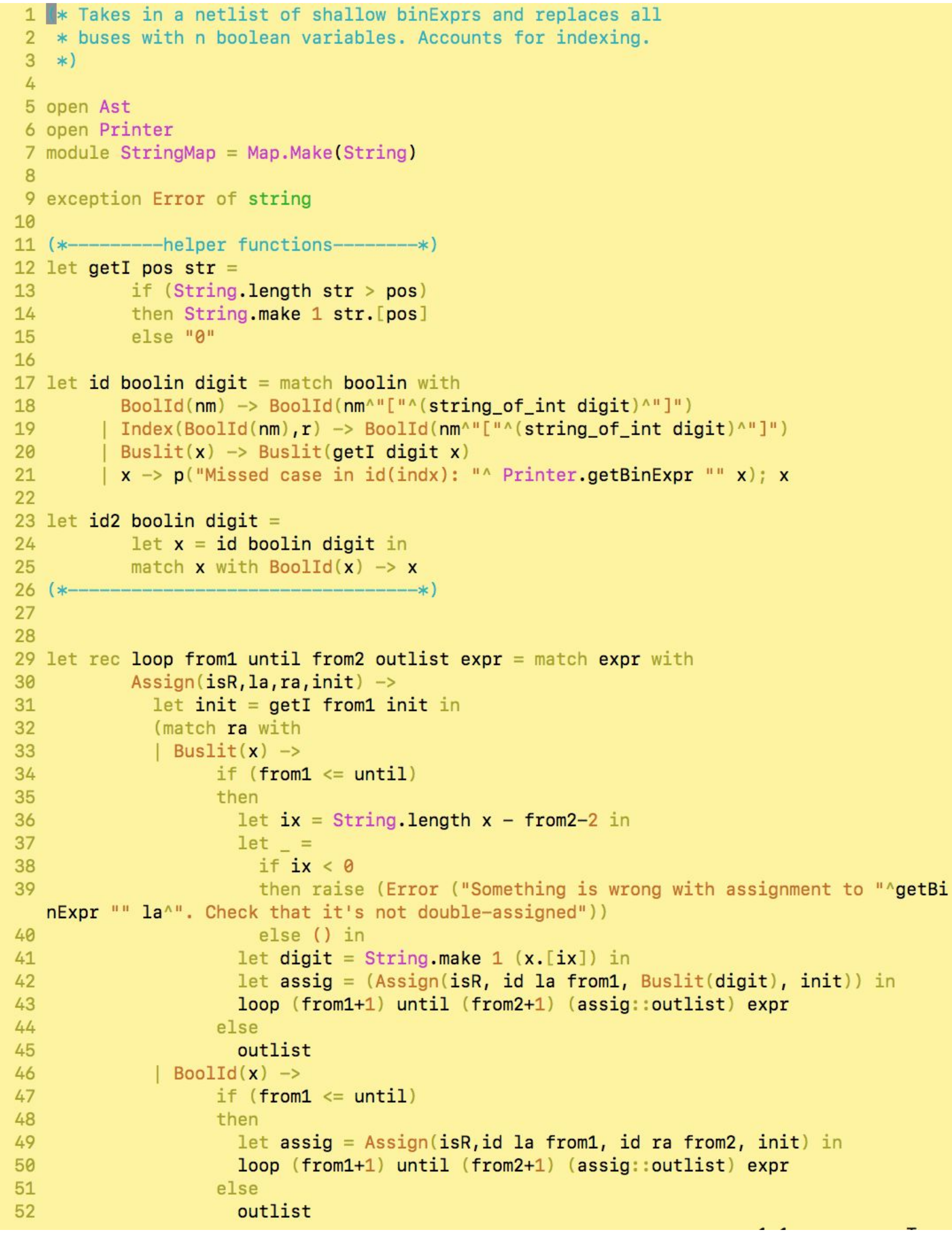


```
(* Despite having once been very aesthetically pleasant,
 * Noloops largely met its demise at the hands Semant, which
 * more or less ate it up (turns out it's easier to collapse
 * for loops while hardening/semantic checking). It now serves
 * to just host some helper methods for semant, that engorged
 * tyrant*)

open Ast
open Printer

exception UndeclaredVar of string

let add (Lit(a)) (Lit(b)) = Lit(a+b)
let sub (Lit(a)) (Lit(b)) = Lit(a-b)

let p x = print_endline(x)

let rec intRep str pos intExp = match intExp with
        Lit(x) -> Lit(x)
      | IntId(x) ->
            if (x = str)
            then Lit(pos)
            else raise(UndeclaredVar ("Variable \"" ^ x ^ "\" is not defined!"))
      | IntBinop(l,op,r) ->
            let l = intRep str pos l in
            let r = intRep str pos r in
            if (op = Add)
            then add l r
            else sub l r

let rec replace str pos exp = match exp with
        Buslit(x) -> exp
      | BoolId(x) -> exp
      | BoolBinop(l,op,r) -> BoolBinop(replace str pos l, op, replace str pos r)
      | Unop(op,x) -> Unop(op, replace str pos x)
      | Assign(isR, l, r, init) -> (match l with
            Index(BoolId(x), Range(a,b)) ->
                    let l = Index(BoolId(x), Range(intRep str pos a, intRep str pos b)) in
                    Assign(isR, l, replace str pos r, init)
          | BoolId(x) -> Assign(isR, l, replace str pos r, init)
       )
      | Index(ModExpr(dec,args),rng) ->
            let args= List.map (replace str pos) args in
            Index(ModExpr(dec,args),rng)
      | Index(ex, Range(a,b)) -> Index(replace str pos ex, Range(intRep str pos a, intRep str pos b))
      | Print(nm,ex) -> Print(nm, replace str pos ex)
      | Call(_,_) -> print_endline("Error: Call got called in noloop."); Noexpr
      | For(_,_,_) -> print_endline("Error: For got called in noloop-replace."); Noexpr
      | ModExpr(dec,args) ->
            let args= List.map (replace str pos) args in
```

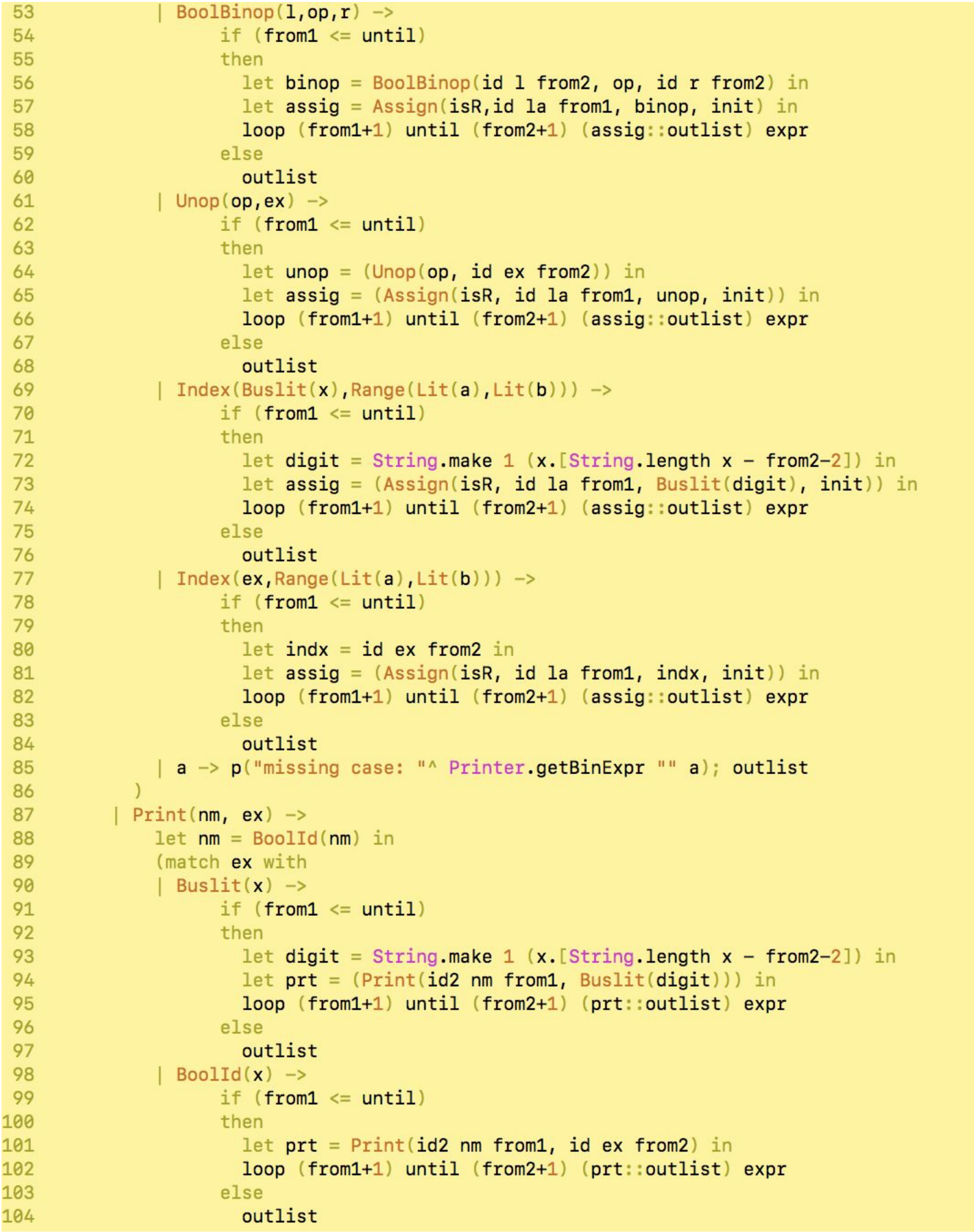


```
            ModExpr(dec,args)
      | Noexpr -> Noexpr
```

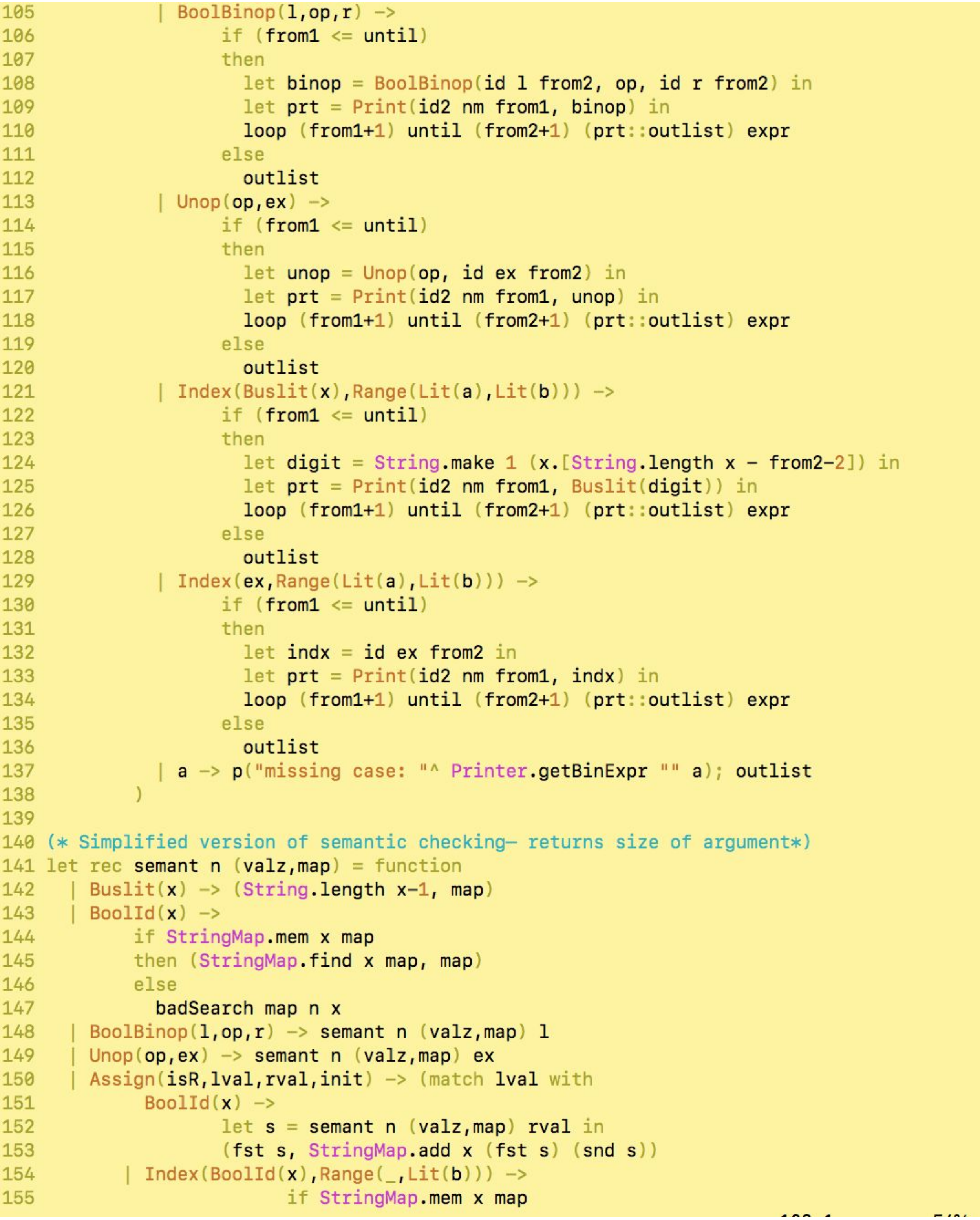
Listing: indexing.ml



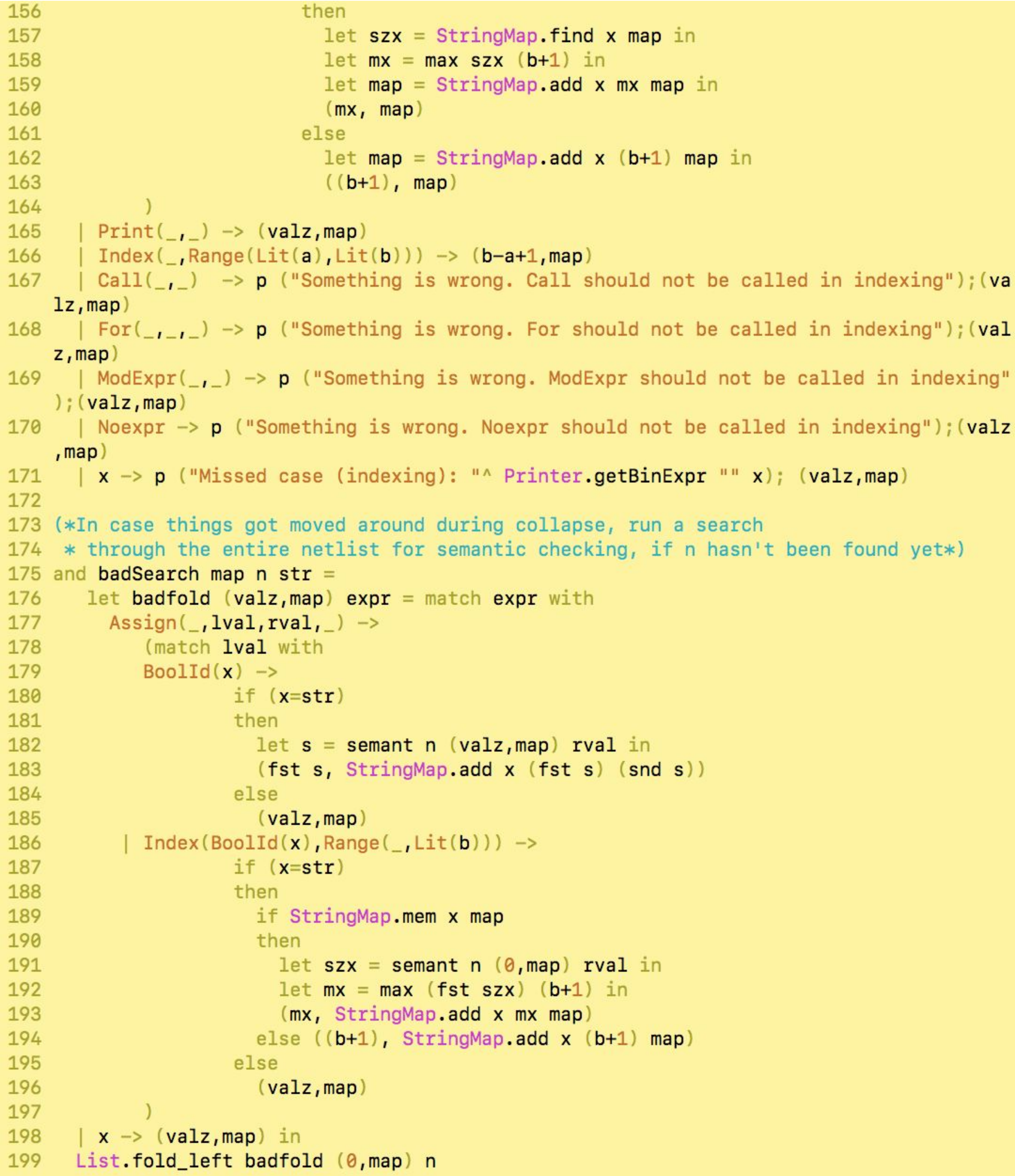
```ocaml
(* Takes in a netlist of shallow binExprs and replaces all
 * buses with n boolean variables. Accounts for indexing.
 *)

open Ast
open Printer
module StringMap = Map.Make(String)

exception Error of string

(*---------helper functions--------*)
let getI pos str =
        if (String.length str > pos)
        then String.make 1 str.[pos]
        else "0"

let id boolin digit = match boolin with
       BoolId(nm) -> BoolId(nm^"["^(string_of_int digit)^"]")
     | Index(BoolId(nm),r) -> BoolId(nm^"["^(string_of_int digit)^"]")
     | Buslit(x) -> Buslit(getI digit x)
     | x -> p("Missed case in id(indx): "^ Printer.getBinExpr "" x); x

let id2 boolin digit =
        let x = id boolin digit in
        match x with BoolId(x) -> x
(*---------------------------------*)


let rec loop from1 until from2 outlist expr = match expr with
        Assign(isR,la,ra,init) ->
          let init = getI from1 init in
          (match ra with
          | Buslit(x) ->
                if (from1 <= until)
                then
                  let ix = String.length x - from2-2 in
                  let _ =
                    if ix < 0
                    then raise (Error ("Something is wrong with assignment to "^getBinExpr "" la^". Check that it's not double-assigned"))
                    else () in
                  let digit = String.make 1 (x.[ix]) in
                  let assig = (Assign(isR, id la from1, Buslit(digit), init)) in
                  loop (from1+1) until (from2+1) (assig::outlist) expr
                else
                  outlist
          | BoolId(x) ->
                if (from1 <= until)
                then
                  let assig = Assign(isR,id la from1, id ra from2, init) in
                  loop (from1+1) until (from2+1) (assig::outlist) expr
                else
                  outlist
```



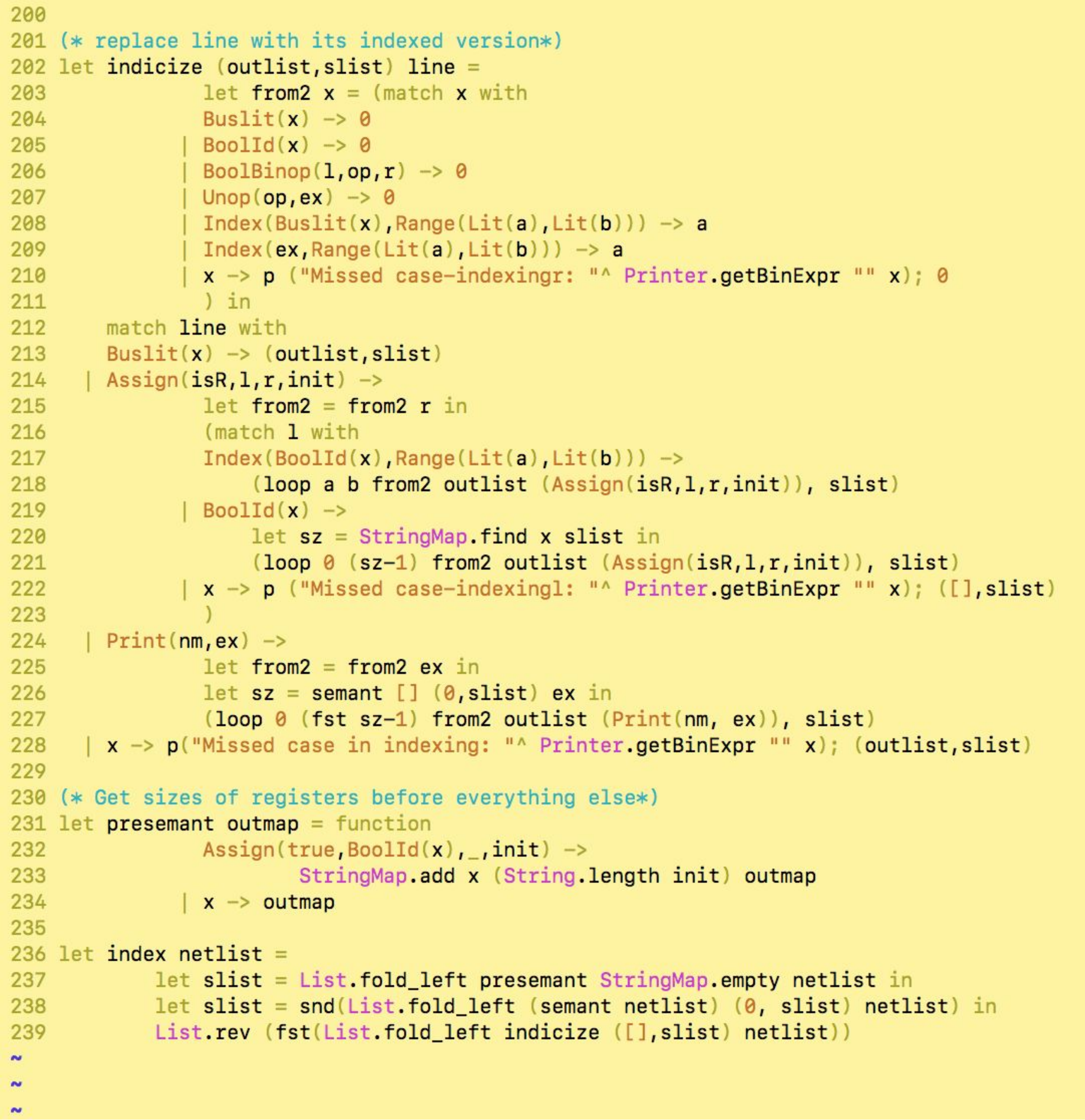
```
          | BoolBinop(l,op,r) ->
                if (from1 <= until)
                then
                  let binop = BoolBinop(id l from2, op, id r from2) in
                  let assig = Assign(isR,id la from1, binop, init) in
                  loop (from1+1) until (from2+1) (assig::outlist) expr
                else
                  outlist
          | Unop(op,ex) ->
                if (from1 <= until)
                then
                  let unop = (Unop(op, id ex from2)) in
                  let assig = (Assign(isR, id la from1, unop, init)) in
                  loop (from1+1) until (from2+1) (assig::outlist) expr
                else
                  outlist
          | Index(Buslit(x),Range(Lit(a),Lit(b))) ->
                if (from1 <= until)
                then
                  let digit = String.make 1 (x.[String.length x - from2-2]) in
                  let assig = (Assign(isR, id la from1, Buslit(digit), init)) in
                  loop (from1+1) until (from2+1) (assig::outlist) expr
                else
                  outlist
          | Index(ex,Range(Lit(a),Lit(b))) ->
                if (from1 <= until)
                then
                  let indx = id ex from2 in
                  let assig = (Assign(isR, id la from1, indx, init)) in
                  loop (from1+1) until (from2+1) (assig::outlist) expr
                else
                  outlist
          | a -> p("missing case: "^ Printer.getBinExpr "" a); outlist
        )
      | Print(nm, ex) ->
          let nm = BoolId(nm) in
          (match ex with
          | Buslit(x) ->
                if (from1 <= until)
                then
                  let digit = String.make 1 (x.[String.length x - from2-2]) in
                  let prt = (Print(id2 nm from1, Buslit(digit))) in
                  loop (from1+1) until (from2+1) (prt::outlist) expr
                else
                  outlist
          | BoolId(x) ->
                if (from1 <= until)
                then
                  let prt = Print(id2 nm from1, id ex from2) in
                  loop (from1+1) until (from2+1) (prt::outlist) expr
                else
                  outlist
```



```
          | BoolBinop(l,op,r) ->
                if (from1 <= until)
                then
                  let binop = BoolBinop(id l from2, op, id r from2) in
                  let prt = Print(id2 nm from1, binop) in
                  loop (from1+1) until (from2+1) (prt::outlist) expr
                else
                  outlist
          | Unop(op,ex) ->
                if (from1 <= until)
                then
                  let unop = Unop(op, id ex from2) in
                  let prt = Print(id2 nm from1, unop) in
                  loop (from1+1) until (from2+1) (prt::outlist) expr
                else
                  outlist
          | Index(Buslit(x),Range(Lit(a),Lit(b))) ->
                if (from1 <= until)
                then
                  let digit = String.make 1 (x.[String.length x - from2-2]) in
                  let prt = Print(id2 nm from1, Buslit(digit)) in
                  loop (from1+1) until (from2+1) (prt::outlist) expr
                else
                  outlist
          | Index(ex,Range(Lit(a),Lit(b))) ->
                if (from1 <= until)
                then
                  let indx = id ex from2 in
                  let prt = Print(id2 nm from1, indx) in
                  loop (from1+1) until (from2+1) (prt::outlist) expr
                else
                  outlist
          | a -> p("missing case: "^ Printer.getBinExpr "" a); outlist
        )

(* Simplified version of semantic checking- returns size of argument*)
let rec semant n (valz,map) = function
  | Buslit(x) -> (String.length x-1, map)
  | BoolId(x) ->
        if StringMap.mem x map
        then (StringMap.find x map, map)
        else
          badSearch map n x
  | BoolBinop(l,op,r) -> semant n (valz,map) l
  | Unop(op,ex) -> semant n (valz,map) ex
  | Assign(isR,lval,rval,init) -> (match lval with
         BoolId(x) ->
                let s = semant n (valz,map) rval in
                (fst s, StringMap.add x (fst s) (snd s))
       | Index(BoolId(x),Range(_,Lit(b))) ->
                      if StringMap.mem x map
```
]

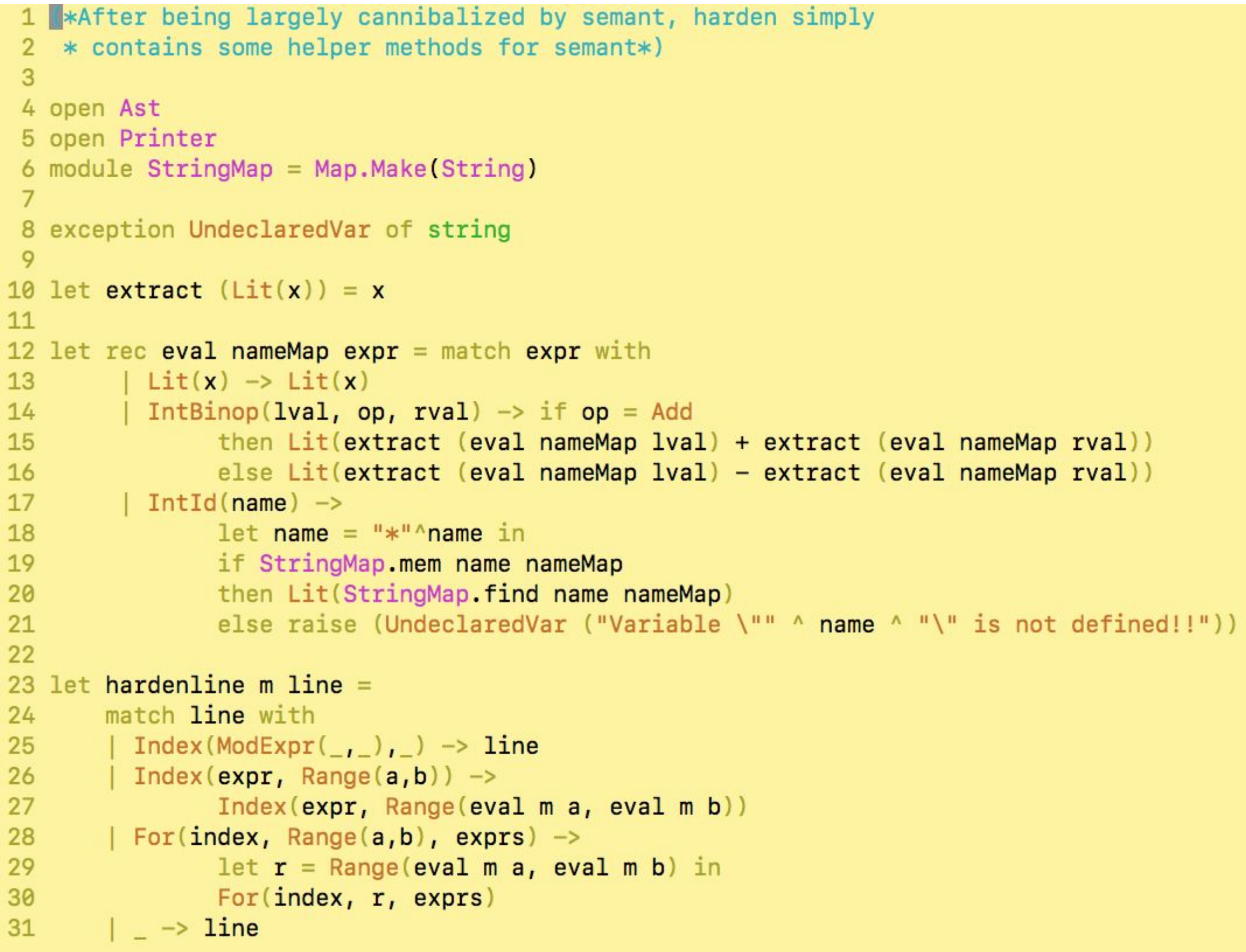


```
                      then
                        let szx = StringMap.find x map in
                        let mx = max szx (b+1) in
                        let map = StringMap.add x mx map in
                        (mx, map)
                      else
                        let map = StringMap.add x (b+1) map in
                        ((b+1), map)
        )
  | Print(_,_) -> (valz,map)
  | Index(_,Range(Lit(a),Lit(b))) -> (b-a+1,map)
  | Call(_,_)  -> p ("Something is wrong. Call should not be called in indexing");(valz,map)
  | For(_,_,_) -> p ("Something is wrong. For should not be called in indexing");(valz,map)
  | ModExpr(_,_) -> p ("Something is wrong. ModExpr should not be called in indexing");(valz,map)
  | Noexpr -> p ("Something is wrong. Noexpr should not be called in indexing");(valz,map)
  | x -> p ("Missed case (indexing): "^ Printer.getBinExpr "" x); (valz,map)

(*In case things got moved around during collapse, run a search
 * through the entire netlist for semantic checking, if n hasn't been found yet*)
and badSearch map n str =
   let badfold (valz,map) expr = match expr with
     Assign(_,lval,rval,_) ->
        (match lval with
        BoolId(x) ->
                if (x=str)
                then
                  let s = semant n (valz,map) rval in
                  (fst s, StringMap.add x (fst s) (snd s))
                else
                  (valz,map)
      | Index(BoolId(x),Range(_,Lit(b))) ->
                if (x=str)
                then
                  if StringMap.mem x map
                  then
                    let szx = semant n (0,map) rval in
                    let mx = max (fst szx) (b+1) in
                    (mx, StringMap.add x mx map)
                  else ((b+1), StringMap.add x (b+1) map)
                else
                  (valz,map)
        )
  | x -> (valz,map) in
  List.fold_left badfold (0,map) n
```



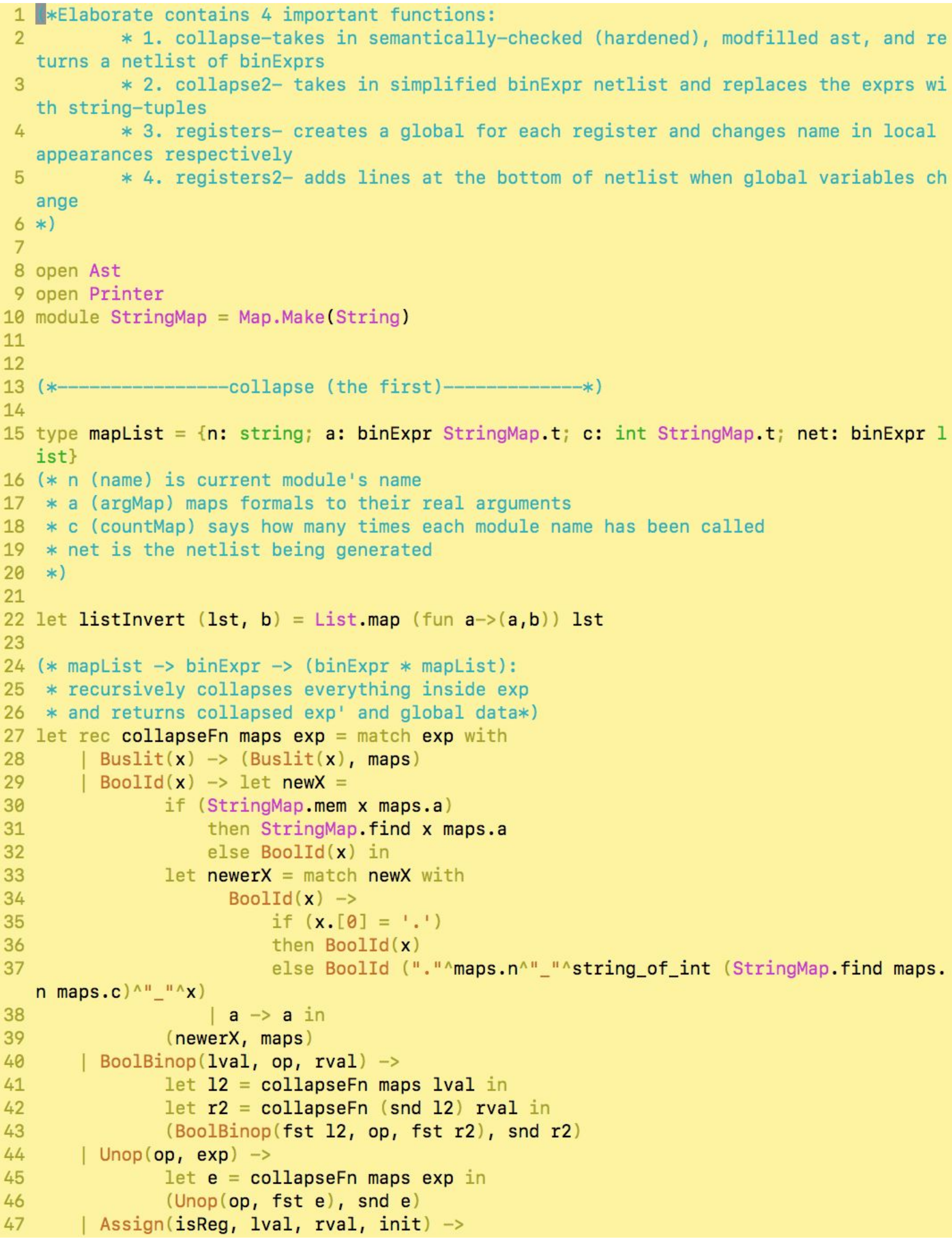
```

(* replace line with its indexed version*)
let indicize (outlist,slist) line =
            let from2 x = (match x with
            Buslit(x) -> 0
          | BoolId(x) -> 0
          | BoolBinop(l,op,r) -> 0
          | Unop(op,ex) -> 0
          | Index(Buslit(x),Range(Lit(a),Lit(b))) -> a
          | Index(ex,Range(Lit(a),Lit(b))) -> a
          | x -> p ("Missed case-indexingr: "^ Printer.getBinExpr "" x); 0
            ) in
    match line with
    Buslit(x) -> (outlist,slist)
  | Assign(isR,l,r,init) ->
            let from2 = from2 r in
            (match l with
            Index(BoolId(x),Range(Lit(a),Lit(b))) ->
                (loop a b from2 outlist (Assign(isR,l,r,init)), slist)
          | BoolId(x) ->
                let sz = StringMap.find x slist in
                (loop 0 (sz-1) from2 outlist (Assign(isR,l,r,init)), slist)
          | x -> p ("Missed case-indexingl: "^ Printer.getBinExpr "" x); ([],slist)
            )
  | Print(nm,ex) ->
            let from2 = from2 ex in
            let sz = semant [] (0,slist) ex in
            (loop 0 (fst sz-1) from2 outlist (Print(nm, ex)), slist)
  | x -> p("Missed case in indexing: "^ Printer.getBinExpr "" x); (outlist,slist)

(* Get sizes of registers before everything else*)
let presemant outmap = function
            Assign(true,BoolId(x),_,init) ->
                    StringMap.add x (String.length init) outmap
          | x -> outmap

let index netlist =
        let slist = List.fold_left presemant StringMap.empty netlist in
        let slist = snd(List.fold_left (semant netlist) (0, slist) netlist) in
        List.rev (fst(List.fold_left indicize ([],slist) netlist))
~
~
~
```

Listing: harden.ml

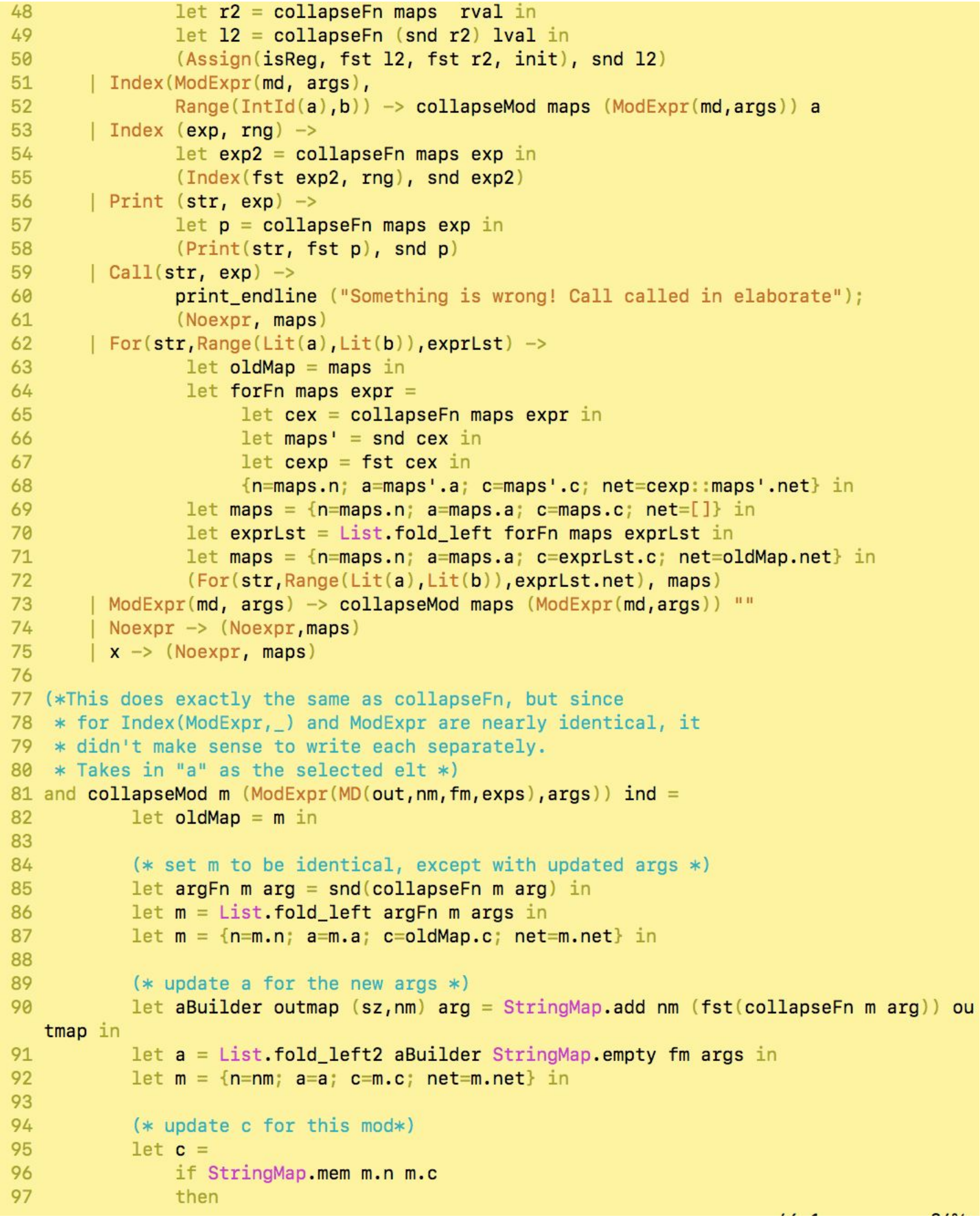


```ocaml
(*After being largely cannibalized by semant, harden simply
 * contains some helper methods for semant*)

open Ast
open Printer
module StringMap = Map.Make(String)

exception UndeclaredVar of string

let extract (Lit(x)) = x

let rec eval nameMap expr = match expr with
     | Lit(x) -> Lit(x)
     | IntBinop(lval, op, rval) -> if op = Add
            then Lit(extract (eval nameMap lval) + extract (eval nameMap rval))
            else Lit(extract (eval nameMap lval) - extract (eval nameMap rval))
     | IntId(name) ->
            let name = "*"^name in
            if StringMap.mem name nameMap
            then Lit(StringMap.find name nameMap)
            else raise (UndeclaredVar ("Variable \"" ^ name ^ "\" is not defined!!"))

let hardenline m line =
    match line with
    | Index(ModExpr(_,_),_) -> line
    | Index(expr, Range(a,b)) ->
            Index(expr, Range(eval m a, eval m b))
    | For(index, Range(a,b), exprs) ->
            let r = Range(eval m a, eval m b) in
            For(index, r, exprs)
    | _ -> line
```

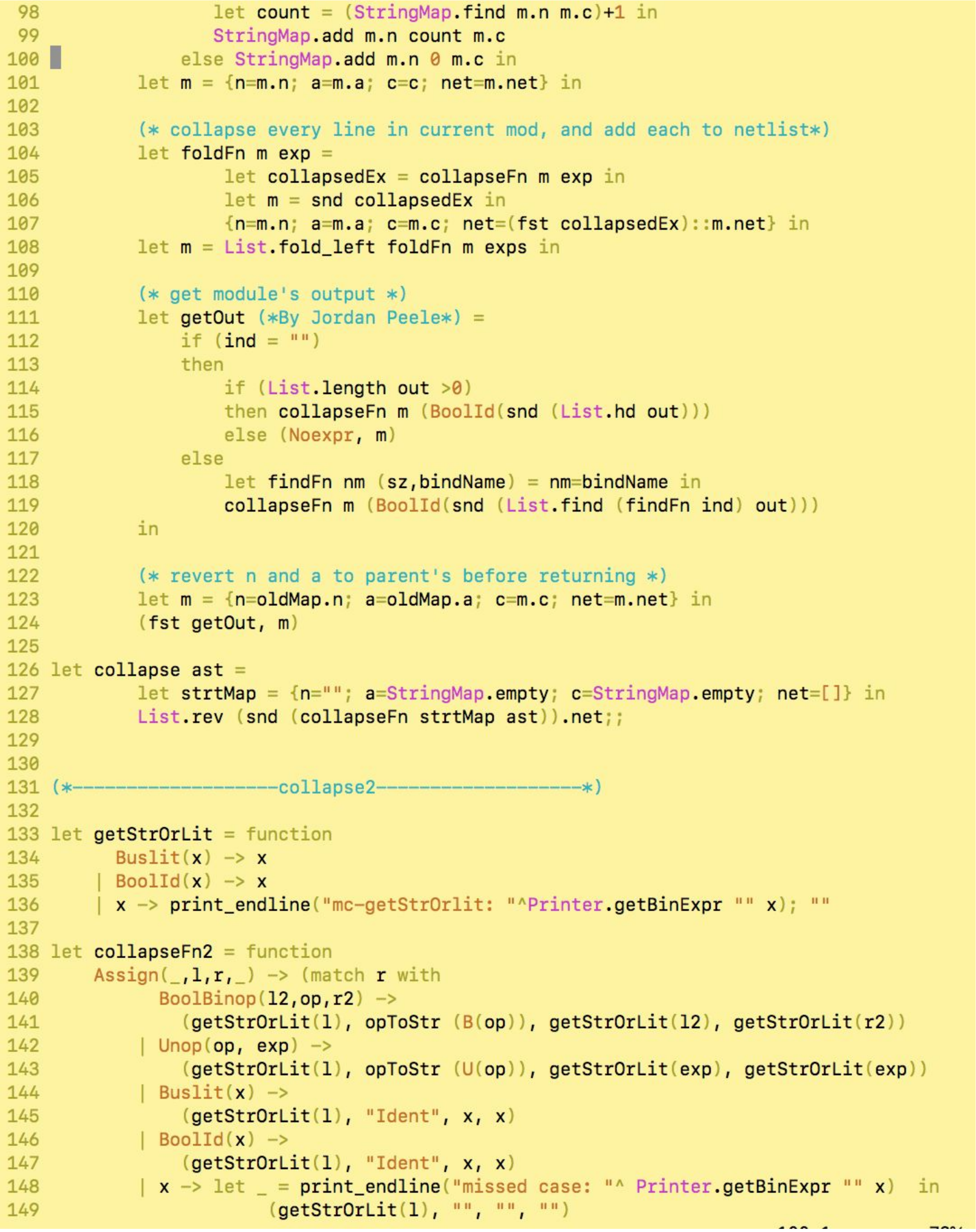
Listing: elaborate.ml



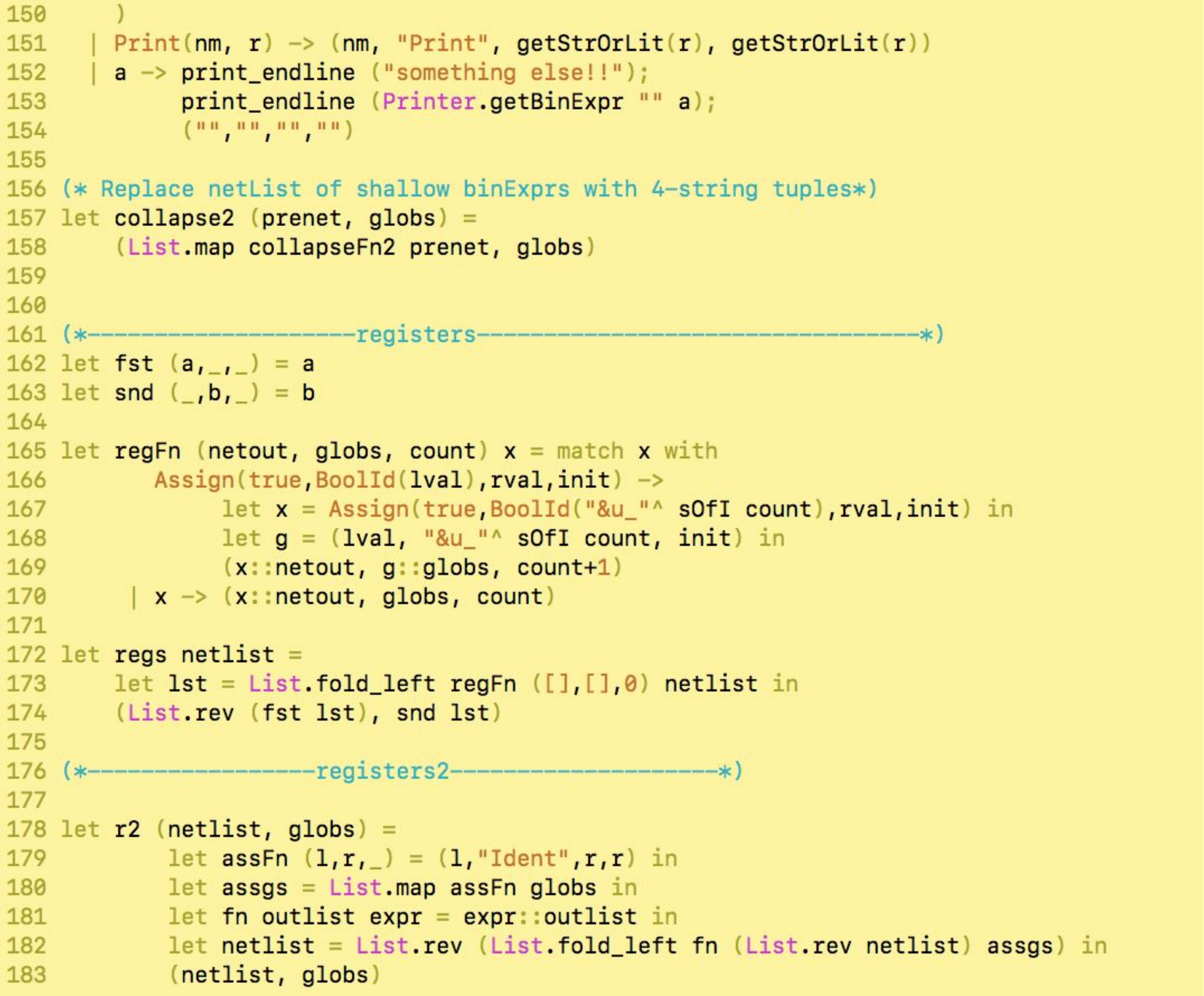
```
(*Elaborate contains 4 important functions:
        * 1. collapse-takes in semantically-checked (hardened), modfilled ast, and returns a netlist of binExprs
        * 2. collapse2- takes in simplified binExpr netlist and replaces the exprs with string-tuples
        * 3. registers- creates a global for each register and changes name in local appearances respectively
        * 4. registers2- adds lines at the bottom of netlist when global variables change
*)

open Ast
open Printer
module StringMap = Map.Make(String)


(*----------------collapse (the first)-------------*)

type mapList = {n: string; a: binExpr StringMap.t; c: int StringMap.t; net: binExpr list}
(* n (name) is current module's name
 * a (argMap) maps formals to their real arguments
 * c (countMap) says how many times each module name has been called
 * net is the netlist being generated
 *)

let listInvert (lst, b) = List.map (fun a->(a,b)) lst

(* mapList -> binExpr -> (binExpr * mapList):
 * recursively collapses everything inside exp
 * and returns collapsed exp' and global data*)
let rec collapseFn maps exp = match exp with
    | Buslit(x) -> (Buslit(x), maps)
    | BoolId(x) -> let newX =
            if (StringMap.mem x maps.a)
                then StringMap.find x maps.a
                else BoolId(x) in
            let newerX = match newX with
                  BoolId(x) ->
                      if (x.[0] = '.')
                      then BoolId(x)
                      else BoolId ("."^maps.n^"_"^string_of_int (StringMap.find maps.n maps.c)^"_"^x)
                | a -> a in
            (newerX, maps)
    | BoolBinop(lval, op, rval) ->
            let l2 = collapseFn maps lval in
            let r2 = collapseFn (snd l2) rval in
            (BoolBinop(fst l2, op, fst r2), snd r2)
    | Unop(op, exp) ->
            let e = collapseFn maps exp in
            (Unop(op, fst e), snd e)
    | Assign(isReg, lval, rval, init) ->
```



```
            let r2 = collapseFn maps  rval in
            let l2 = collapseFn (snd r2) lval in
            (Assign(isReg, fst l2, fst r2, init), snd l2)
    | Index(ModExpr(md, args),
            Range(IntId(a),b)) -> collapseMod maps (ModExpr(md,args)) a
    | Index (exp, rng) ->
            let exp2 = collapseFn maps exp in
            (Index(fst exp2, rng), snd exp2)
    | Print (str, exp) ->
            let p = collapseFn maps exp in
            (Print(str, fst p), snd p)
    | Call(str, exp) ->
            print_endline ("Something is wrong! Call called in elaborate");
            (Noexpr, maps)
    | For(str,Range(Lit(a),Lit(b)),exprLst) ->
             let oldMap = maps in
             let forFn maps expr =
                  let cex = collapseFn maps expr in
                  let maps' = snd cex in
                  let cexp = fst cex in
                  {n=maps.n; a=maps'.a; c=maps'.c; net=cexp::maps'.net} in
             let maps = {n=maps.n; a=maps.a; c=maps.c; net=[]} in
             let exprLst = List.fold_left forFn maps exprLst in
             let maps = {n=maps.n; a=maps.a; c=exprLst.c; net=oldMap.net} in
             (For(str,Range(Lit(a),Lit(b)),exprLst.net), maps)
    | ModExpr(md, args) -> collapseMod maps (ModExpr(md,args)) ""
    | Noexpr -> (Noexpr,maps)
    | x -> (Noexpr, maps)

(*This does exactly the same as collapseFn, but since
 * for Index(ModExpr,_) and ModExpr are nearly identical, it
 * didn't make sense to write each separately.
 * Takes in "a" as the selected elt *)
and collapseMod m (ModExpr(MD(out,nm,fm,exps),args)) ind =
        let oldMap = m in

        (* set m to be identical, except with updated args *)
        let argFn m arg = snd(collapseFn m arg) in
        let m = List.fold_left argFn m args in
        let m = {n=m.n; a=m.a; c=oldMap.c; net=m.net} in

        (* update a for the new args *)
        let aBuilder outmap (sz,nm) arg = StringMap.add nm (fst(collapseFn m arg)) outmap in
        let a = List.fold_left2 aBuilder StringMap.empty fm args in
        let m = {n=nm; a=a; c=m.c; net=m.net} in

        (* update c for this mod*)
        let c =
            if StringMap.mem m.n m.c
            then
```

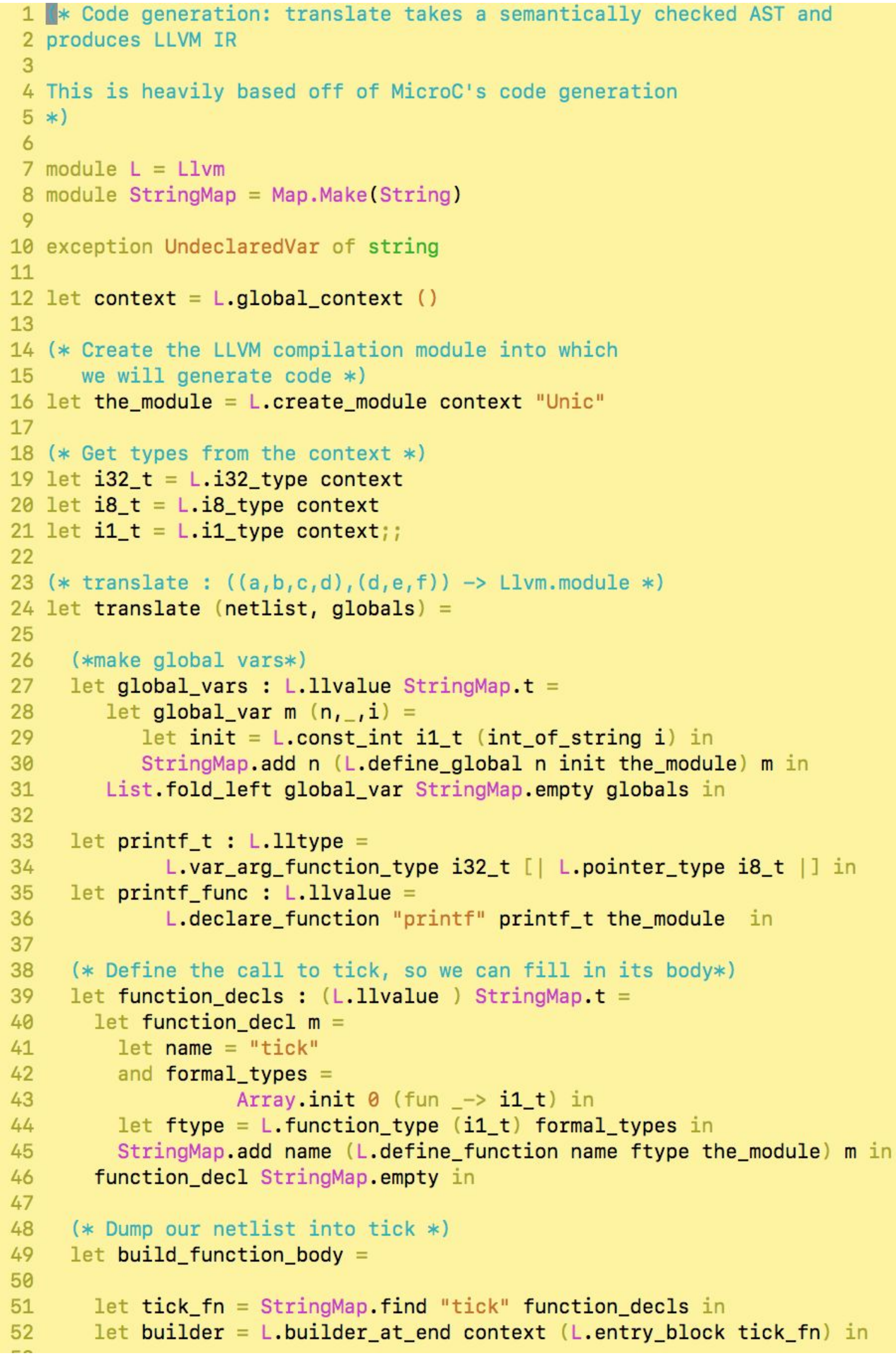


```
               let count = (StringMap.find m.n m.c)+1 in
               StringMap.add m.n count m.c
            else StringMap.add m.n 0 m.c in
        let m = {n=m.n; a=m.a; c=c; net=m.net} in

        (* collapse every line in current mod, and add each to netlist*)
        let foldFn m exp =
                let collapsedEx = collapseFn m exp in
                let m = snd collapsedEx in
                {n=m.n; a=m.a; c=m.c; net=(fst collapsedEx)::m.net} in
        let m = List.fold_left foldFn m exps in

        (* get module's output *)
        let getOut (*By Jordan Peele*) =
            if (ind = "")
            then
                if (List.length out >0)
                then collapseFn m (BoolId(snd (List.hd out)))
                else (Noexpr, m)
            else
                let findFn nm (sz,bindName) = nm=bindName in
                collapseFn m (BoolId(snd (List.find (findFn ind) out)))
        in

        (* revert n and a to parent's before returning *)
        let m = {n=oldMap.n; a=oldMap.a; c=m.c; net=m.net} in
        (fst getOut, m)

let collapse ast =
        let strtMap = {n=""; a=StringMap.empty; c=StringMap.empty; net=[]} in
        List.rev (snd (collapseFn strtMap ast)).net;;


(*--------------------collapse2--------------------*)

let getStrOrLit = function
      Buslit(x) -> x
    | BoolId(x) -> x
    | x -> print_endline("mc-getStrOrlit: "^Printer.getBinExpr "" x); ""

let collapseFn2 = function
    Assign(_,l,r,_) -> (match r with
          BoolBinop(l2,op,r2) ->
            (getStrOrLit(l), opToStr (B(op)), getStrOrLit(l2), getStrOrLit(r2))
        | Unop(op, exp) ->
            (getStrOrLit(l), opToStr (U(op)), getStrOrLit(exp), getStrOrLit(exp))
        | Buslit(x) ->
            (getStrOrLit(l), "Ident", x, x)
        | BoolId(x) ->
            (getStrOrLit(l), "Ident", x, x)
        | x -> let _ = print_endline("missed case: "^ Printer.getBinExpr "" x)  in
                    (getStrOrLit(l), "", "", "")
```

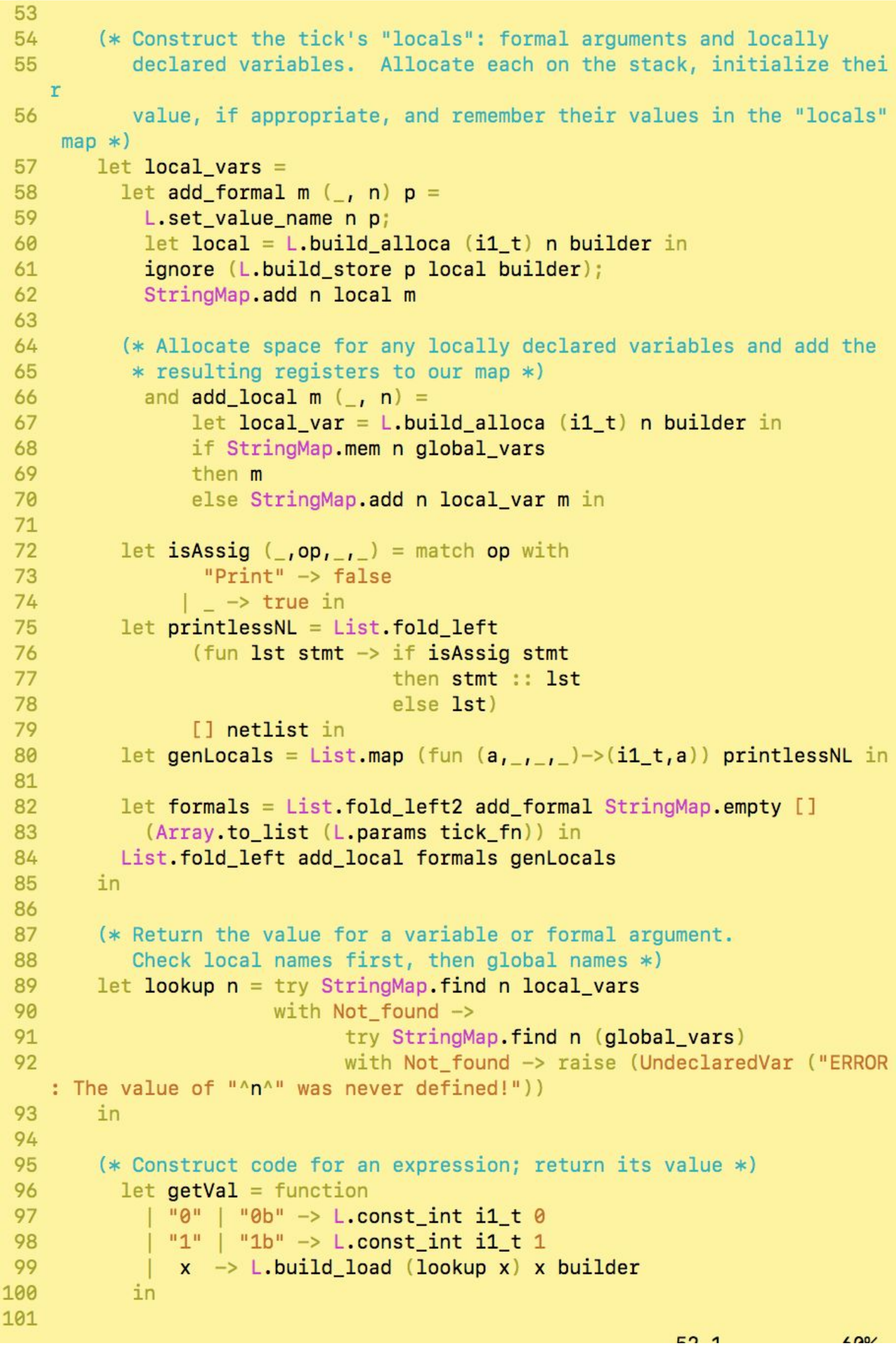


```
    )
  | Print(nm, r) -> (nm, "Print", getStrOrLit(r), getStrOrLit(r))
  | a -> print_endline ("something else!!");
         print_endline (Printer.getBinExpr "" a);
         ("","","","")

(* Replace netList of shallow binExprs with 4-string tuples*)
let collapse2 (prenet, globs) =
    (List.map collapseFn2 prenet, globs)


(*------------------registers--------------------------------*)
let fst (a,_,_) = a
let snd (_,b,_) = b

let regFn (netout, globs, count) x = match x with
       Assign(true,BoolId(lval),rval,init) ->
            let x = Assign(true,BoolId("&u_"^ sOfI count),rval,init) in
            let g = (lval, "&u_"^ sOfI count, init) in
            (x::netout, g::globs, count+1)
     | x -> (x::netout, globs, count)

let regs netlist =
    let lst = List.fold_left regFn ([],[],0) netlist in
    (List.rev (fst lst), snd lst)

(*-----------------registers2--------------------*)

let r2 (netlist, globs) =
        let assFn (l,r,_) = (l,"Ident",r,r) in
        let assgs = List.map assFn globs in
        let fn outlist expr = expr::outlist in
        let netlist = List.rev (List.fold_left fn (List.rev netlist) assgs) in
        (netlist, globs)
```

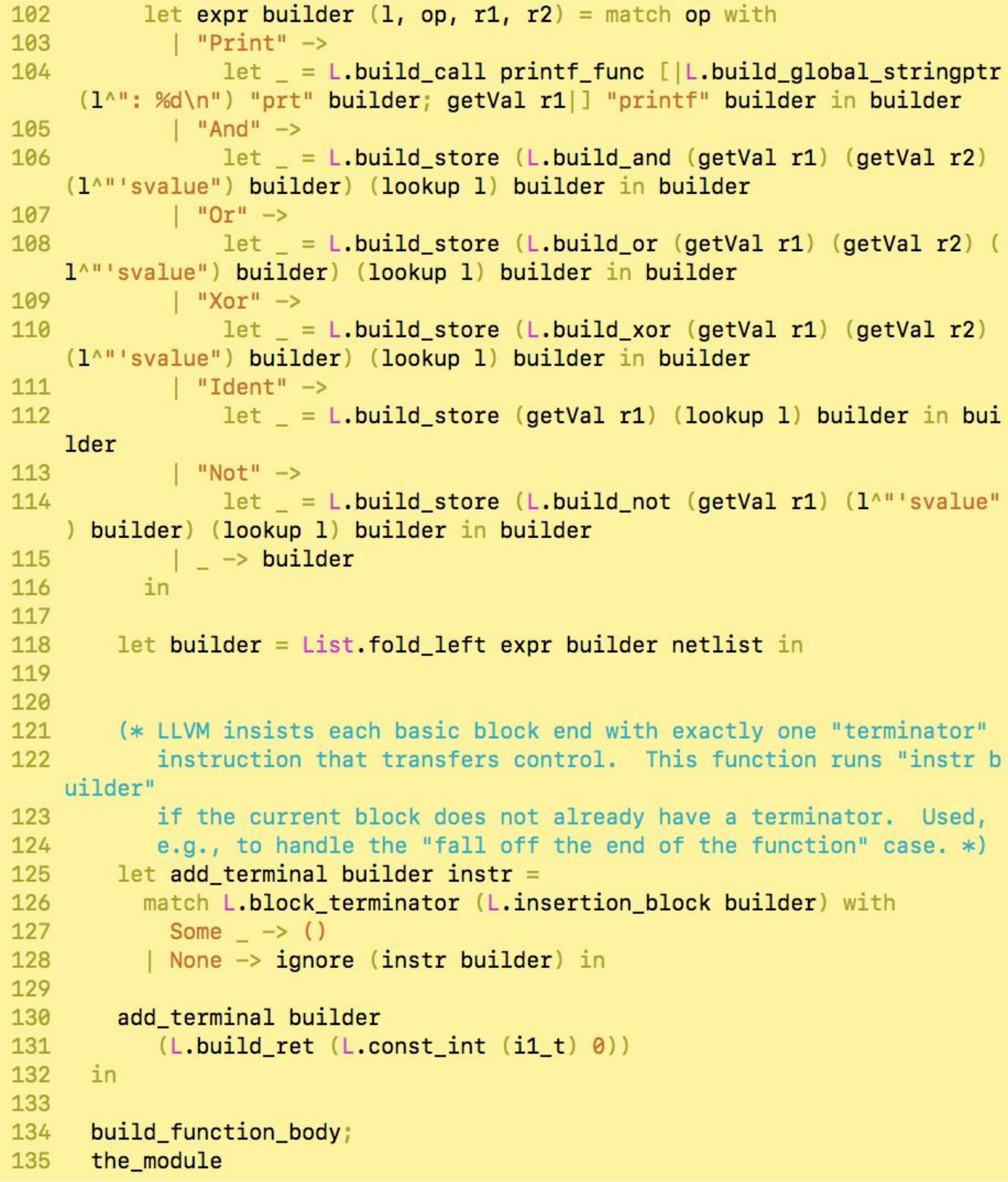
Listing: codegen.ml



```
(* Code generation: translate takes a semantically checked AST and
produces LLVM IR

This is heavily based off of MicroC's code generation
*)

module L = Llvm
module StringMap = Map.Make(String)

exception UndeclaredVar of string

let context = L.global_context ()

(* Create the LLVM compilation module into which
   we will generate code *)
let the_module = L.create_module context "Unic"

(* Get types from the context *)
let i32_t = L.i32_type context
let i8_t = L.i8_type context
let i1_t = L.i1_type context;;

(* translate : ((a,b,c,d),(d,e,f)) -> Llvm.module *)
let translate (netlist, globals) =

  (*make global vars*)
  let global_vars : L.llvalue StringMap.t =
     let global_var m (n,_,i) =
        let init = L.const_int i1_t (int_of_string i) in
        StringMap.add n (L.define_global n init the_module) m in
     List.fold_left global_var StringMap.empty globals in

  let printf_t : L.lltype =
          L.var_arg_function_type i32_t [| L.pointer_type i8_t |] in
  let printf_func : L.llvalue =
          L.declare_function "printf" printf_t the_module  in

  (* Define the call to tick, so we can fill in its body*)
  let function_decls : (L.llvalue ) StringMap.t =
    let function_decl m =
      let name = "tick"
      and formal_types =
                Array.init 0 (fun _-> i1_t) in
      let ftype = L.function_type (i1_t) formal_types in
      StringMap.add name (L.define_function name ftype the_module) m in
    function_decl StringMap.empty in

  (* Dump our netlist into tick *)
  let build_function_body =

    let tick_fn = StringMap.find "tick" function_decls in
    let builder = L.builder_at_end context (L.entry_block tick_fn) in
```

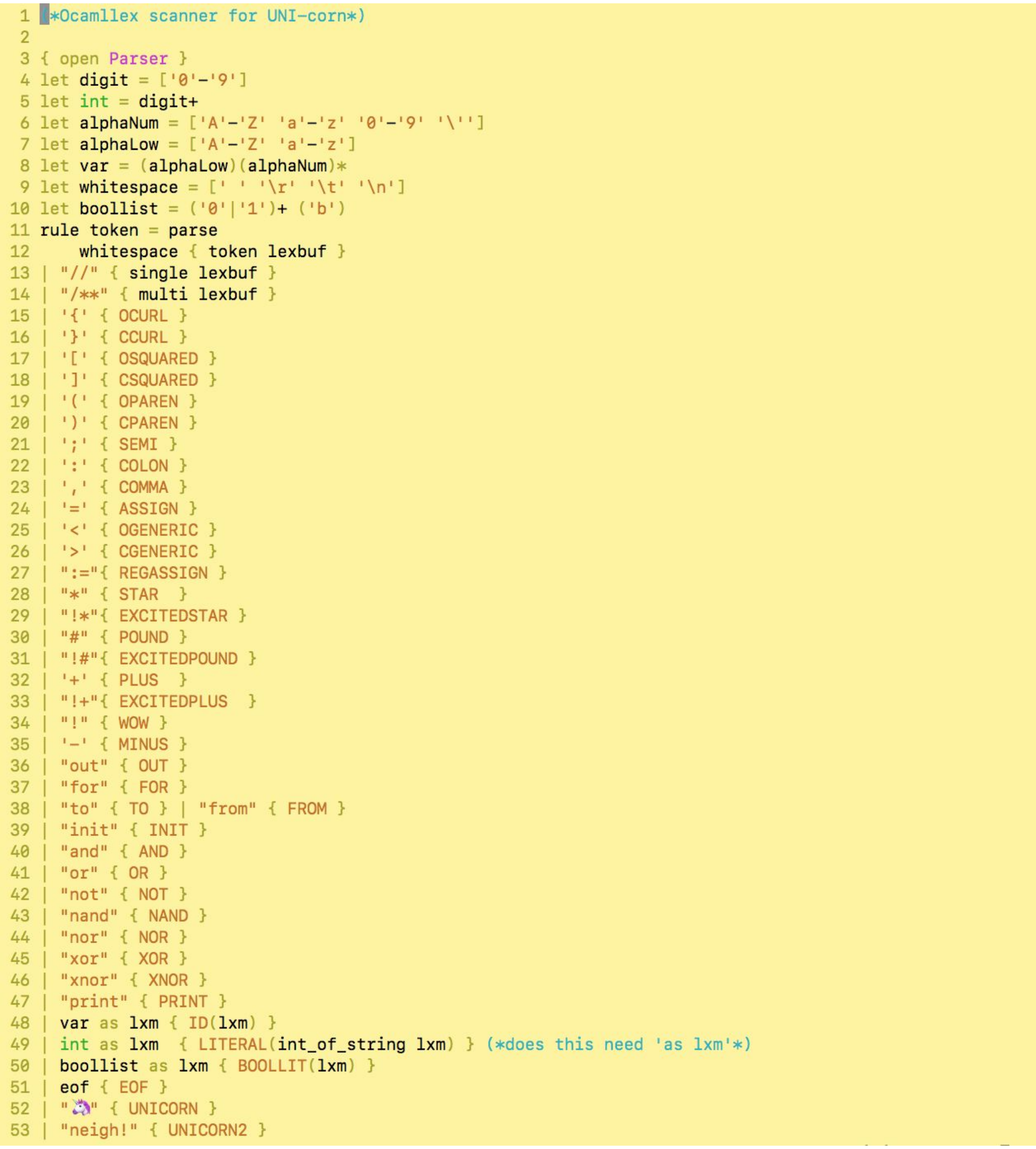


```

    (* Construct the tick's "locals": formal arguments and locally
       declared variables.  Allocate each on the stack, initialize their
       value, if appropriate, and remember their values in the "locals" map *)
    let local_vars =
      let add_formal m (_, n) p =
        L.set_value_name n p;
        let local = L.build_alloca (i1_t) n builder in
        ignore (L.build_store p local builder);
        StringMap.add n local m

      (* Allocate space for any locally declared variables and add the
       * resulting registers to our map *)
        and add_local m (_, n) =
            let local_var = L.build_alloca (i1_t) n builder in
            if StringMap.mem n global_vars
            then m
            else StringMap.add n local_var m in

      let isAssig (_,op,_,_) = match op with
             "Print" -> false
           | _ -> true in
      let printlessNL = List.fold_left
            (fun lst stmt -> if isAssig stmt
                             then stmt :: lst
                             else lst)
            [] netlist in
      let genLocals = List.map (fun (a,_,_,_)->(i1_t,a)) printlessNL in

      let formals = List.fold_left2 add_formal StringMap.empty []
         (Array.to_list (L.params tick_fn)) in
      List.fold_left add_local formals genLocals
    in

    (* Return the value for a variable or formal argument.
       Check local names first, then global names *)
    let lookup n = try StringMap.find n local_vars
                   with Not_found ->
                         try StringMap.find n (global_vars)
                         with Not_found -> raise (UndeclaredVar ("ERROR: The value of "^n^" was never defined!"))
    in

    (* Construct code for an expression; return its value *)
      let getVal = function
        | "0" | "0b" -> L.const_int i1_t 0
        | "1" | "1b" -> L.const_int i1_t 1
        |  x  -> L.build_load (lookup x) x builder
       in

                                                   53,1          60%
```

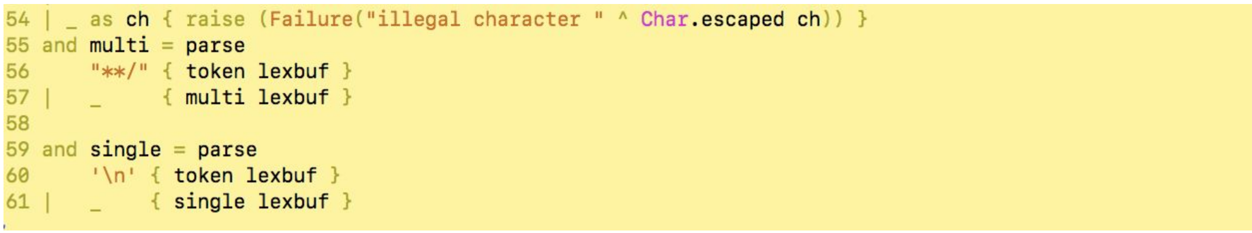


```
      let expr builder (l, op, r1, r2) = match op with
        | "Print" ->
            let _ = L.build_call printf_func [|L.build_global_stringptr (l^": %d\n") "prt" builder; getVal r1|] "printf" builder in builder
        | "And" ->
            let _ = L.build_store (L.build_and (getVal r1) (getVal r2) (l^"'svalue") builder) (lookup l) builder in builder
        | "Or" ->
            let _ = L.build_store (L.build_or (getVal r1) (getVal r2) (l^"'svalue") builder) (lookup l) builder in builder
        | "Xor" ->
            let _ = L.build_store (L.build_xor (getVal r1) (getVal r2) (l^"'svalue") builder) (lookup l) builder in builder
        | "Ident" ->
            let _ = L.build_store (getVal r1) (lookup l) builder in builder
        | "Not" ->
            let _ = L.build_store (L.build_not (getVal r1) (l^"'svalue") builder) (lookup l) builder in builder
        | _ -> builder
      in

    let builder = List.fold_left expr builder netlist in


    (* LLVM insists each basic block end with exactly one "terminator"
       instruction that transfers control.  This function runs "instr builder"
       if the current block does not already have a terminator.  Used,
       e.g., to handle the "fall off the end of the function" case. *)
    let add_terminal builder instr =
      match L.block_terminator (L.insertion_block builder) with
        Some _ -> ()
      | None -> ignore (instr builder) in

    add_terminal builder
       (L.build_ret (L.const_int (i1_t) 0))
  in

  build_function_body;
  the_module
```

Listing: scanner.mll

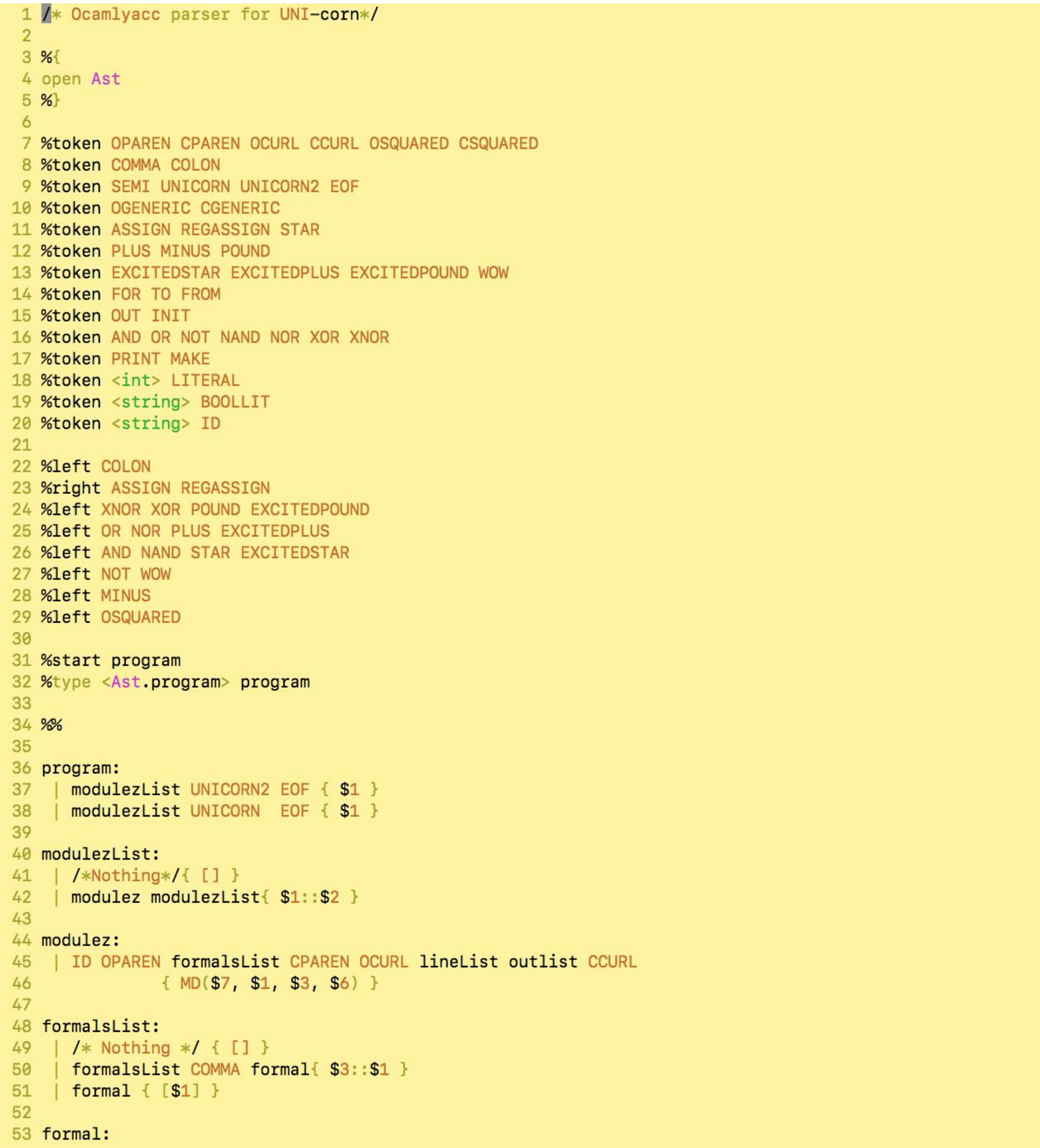


```
(*Ocamllex scanner for UNI-corn*)

{ open Parser }
let digit = ['0'-'9']
let int = digit+
let alphaNum = ['A'-'Z' 'a'-'z' '0'-'9' '\'']
let alphaLow = ['A'-'Z' 'a'-'z']
let var = (alphaLow)(alphaNum)*
let whitespace = [' ' '\r' '\t' '\n']
let boollist = ('0'|'1')+ ('b')
rule token = parse
    whitespace { token lexbuf }
| "//" { single lexbuf }
| "/**" { multi lexbuf }
| '{' { OCURL }
| '}' { CCURL }
| '[' { OSQUARED }
| ']' { CSQUARED }
| '(' { OPAREN }
| ')' { CPAREN }
| ';' { SEMI }
| ':' { COLON }
| ',' { COMMA }
| '=' { ASSIGN }
| '<' { OGENERIC }
| '>' { CGENERIC }
| ":="{ REGASSIGN }
| "*" { STAR  }
| "!*"{ EXCITEDSTAR }
| "#" { POUND }
| "!#"{ EXCITEDPOUND }
| '+' { PLUS  }
| "!+"{ EXCITEDPLUS  }
| "!" { WOW }
| '-' { MINUS }
| "out" { OUT }
| "for" { FOR }
| "to" { TO } | "from" { FROM }
| "init" { INIT }
| "and" { AND }
| "or" { OR }
| "not" { NOT }
| "nand" { NAND }
| "nor" { NOR }
| "xor" { XOR }
| "xnor" { XNOR }
| "print" { PRINT }
| var as lxm { ID(lxm) }
| int as lxm  { LITERAL(int_of_string lxm) } (*does this need 'as lxm'*)
| boollist as lxm { BOOLLIT(lxm) }
| eof { EOF }
| "🦄" { UNICORN }
| "neigh!" { UNICORN2 }
```

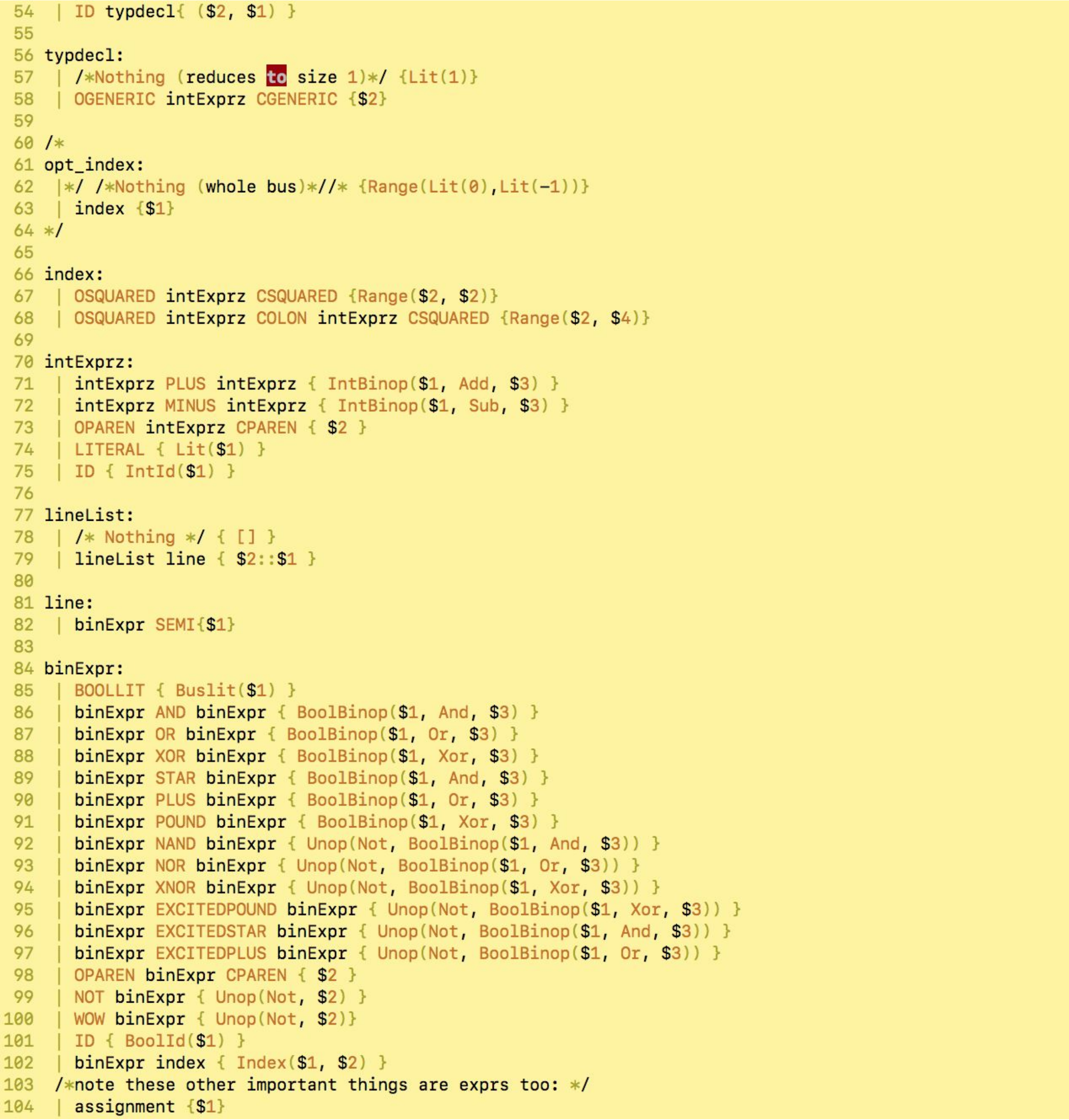


```
| _ as ch { raise (Failure("illegal character " ^ Char.escaped ch)) }
and multi = parse
    "**/" { token lexbuf }
|   _     { multi lexbuf }

and single = parse
    '\n' { token lexbuf }
|   _    { single lexbuf }
```

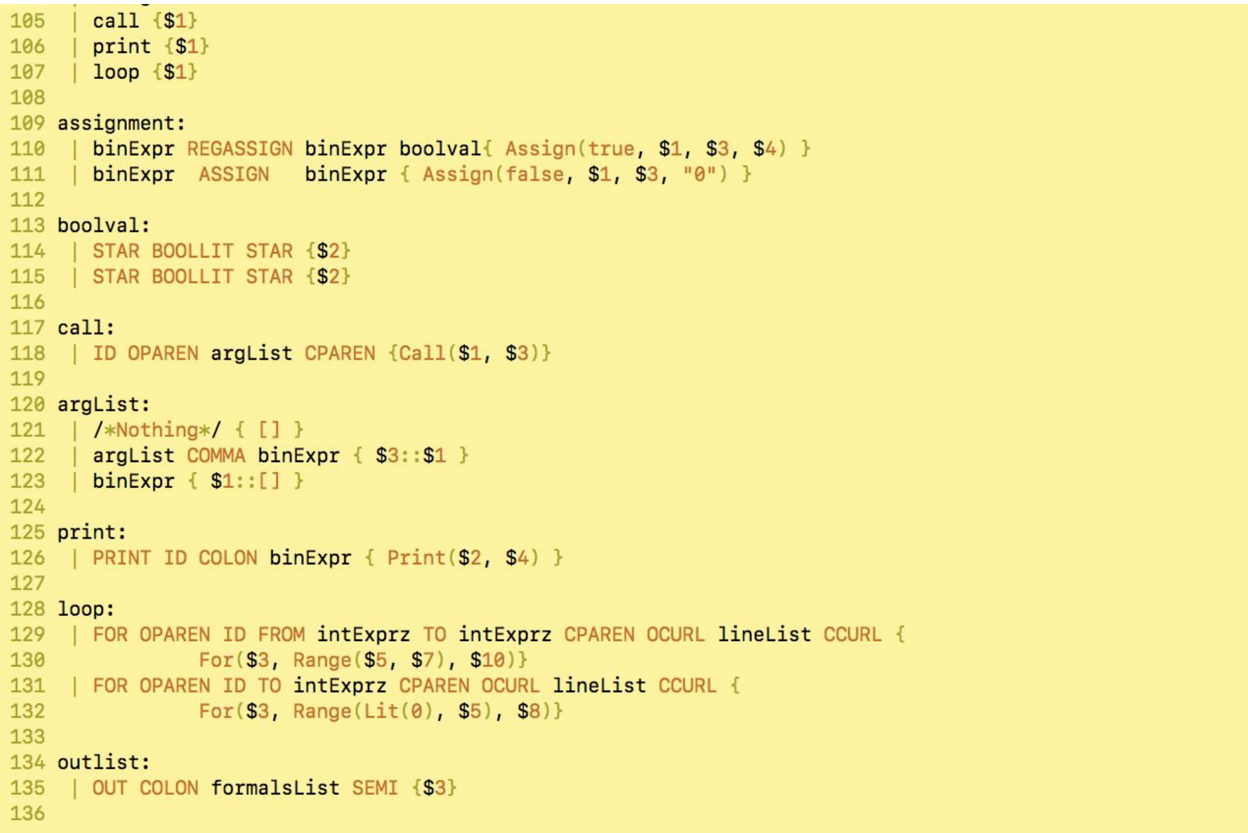
Listing: parser.mly



```
/* Ocamlyacc parser for UNI-corn*/

%{
open Ast
%}

%token OPAREN CPAREN OCURL CCURL OSQUARED CSQUARED
%token COMMA COLON
%token SEMI UNICORN UNICORN2 EOF
%token OGENERIC CGENERIC
%token ASSIGN REGASSIGN STAR
%token PLUS MINUS POUND
%token EXCITEDSTAR EXCITEDPLUS EXCITEDPOUND WOW
%token FOR TO FROM
%token OUT INIT
%token AND OR NOT NAND NOR XOR XNOR
%token PRINT MAKE
%token <int> LITERAL
%token <string> BOOLLIT
%token <string> ID

%left COLON
%right ASSIGN REGASSIGN
%left XNOR XOR POUND EXCITEDPOUND
%left OR NOR PLUS EXCITEDPLUS
%left AND NAND STAR EXCITEDSTAR
%left NOT WOW
%left MINUS
%left OSQUARED

%start program
%type <Ast.program> program

%%

program:
 | modulezList UNICORN2 EOF { $1 }
 | modulezList UNICORN  EOF { $1 }

modulezList:
 | /*Nothing*/{ [] }
 | modulez modulezList{ $1::$2 }

modulez:
 | ID OPAREN formalsList CPAREN OCURL lineList outlist CCURL
            { MD($7, $1, $3, $6) }

formalsList:
 | /* Nothing */ { [] }
 | formalsList COMMA formal{ $3::$1 }
 | formal { [$1] }

formal:
```

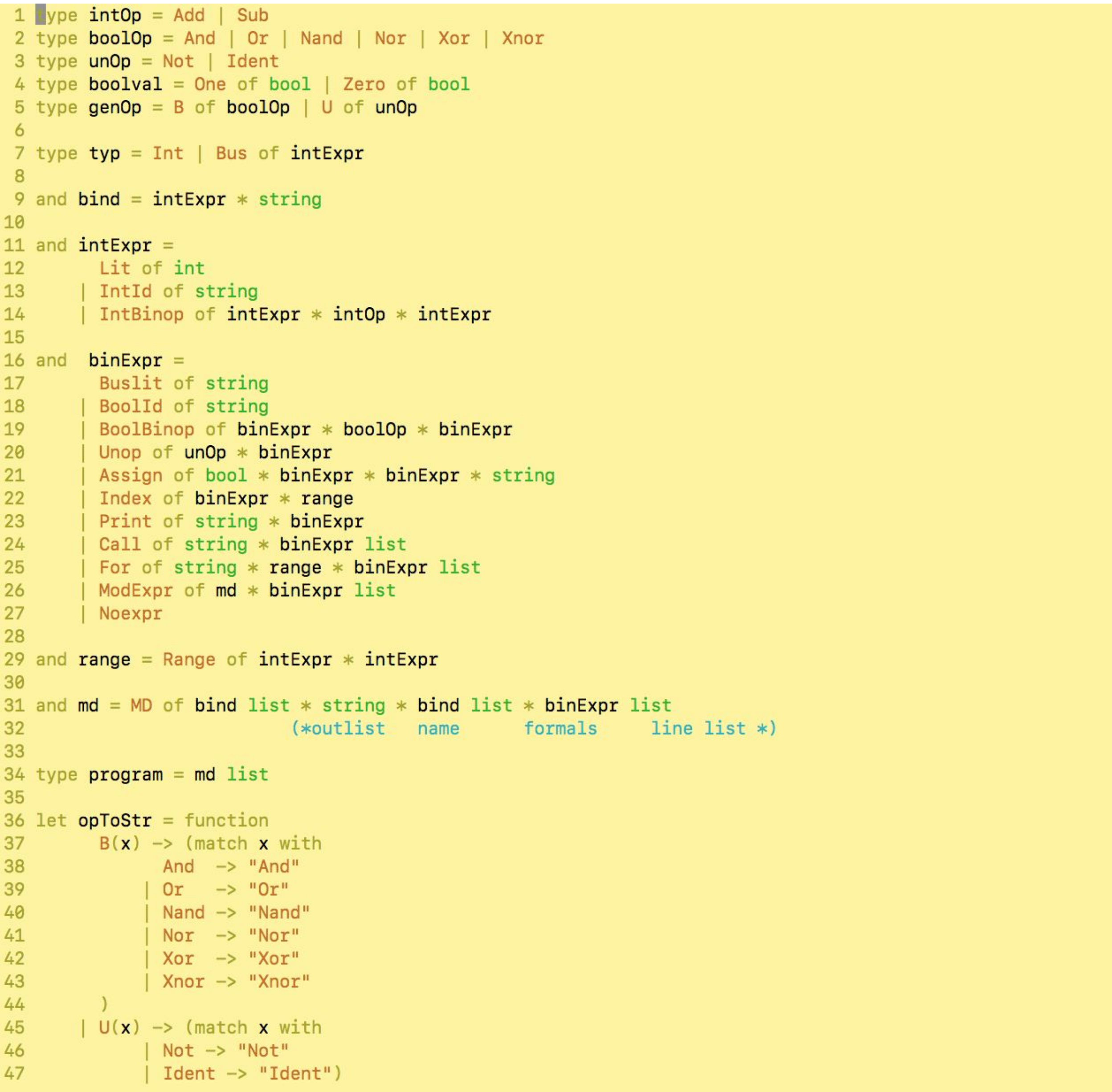


```
 | ID typdecl{ ($2, $1) }

typdecl:
 | /*Nothing (reduces to size 1)*/ {Lit(1)}
 | OGENERIC intExprz CGENERIC {$2}

/*
opt_index:
 |*/ /*Nothing (whole bus)*//* {Range(Lit(0),Lit(-1))}
 | index {$1}
*/

index:
 | OSQUARED intExprz CSQUARED {Range($2, $2)}
 | OSQUARED intExprz COLON intExprz CSQUARED {Range($2, $4)}

intExprz:
 | intExprz PLUS intExprz { IntBinop($1, Add, $3) }
 | intExprz MINUS intExprz { IntBinop($1, Sub, $3) }
 | OPAREN intExprz CPAREN { $2 }
 | LITERAL { Lit($1) }
 | ID { IntId($1) }

lineList:
 | /* Nothing */ { [] }
 | lineList line { $2::$1 }

line:
 | binExpr SEMI{$1}

binExpr:
 | BOOLLIT { Buslit($1) }
 | binExpr AND binExpr { BoolBinop($1, And, $3) }
 | binExpr OR binExpr { BoolBinop($1, Or, $3) }
 | binExpr XOR binExpr { BoolBinop($1, Xor, $3) }
 | binExpr STAR binExpr { BoolBinop($1, And, $3) }
 | binExpr PLUS binExpr { BoolBinop($1, Or, $3) }
 | binExpr POUND binExpr { BoolBinop($1, Xor, $3) }
 | binExpr NAND binExpr { Unop(Not, BoolBinop($1, And, $3)) }
 | binExpr NOR binExpr { Unop(Not, BoolBinop($1, Or, $3)) }
 | binExpr XNOR binExpr { Unop(Not, BoolBinop($1, Xor, $3)) }
 | binExpr EXCITEDPOUND binExpr { Unop(Not, BoolBinop($1, Xor, $3)) }
 | binExpr EXCITEDSTAR binExpr { Unop(Not, BoolBinop($1, And, $3)) }
 | binExpr EXCITEDPLUS binExpr { Unop(Not, BoolBinop($1, Or, $3)) }
 | OPAREN binExpr CPAREN { $2 }
 | NOT binExpr { Unop(Not, $2) }
 | WOW binExpr { Unop(Not, $2)}
 | ID { BoolId($1) }
 | binExpr index { Index($1, $2) }
 /*note these other important things are exprs too: */
 | assignment {$1}
```



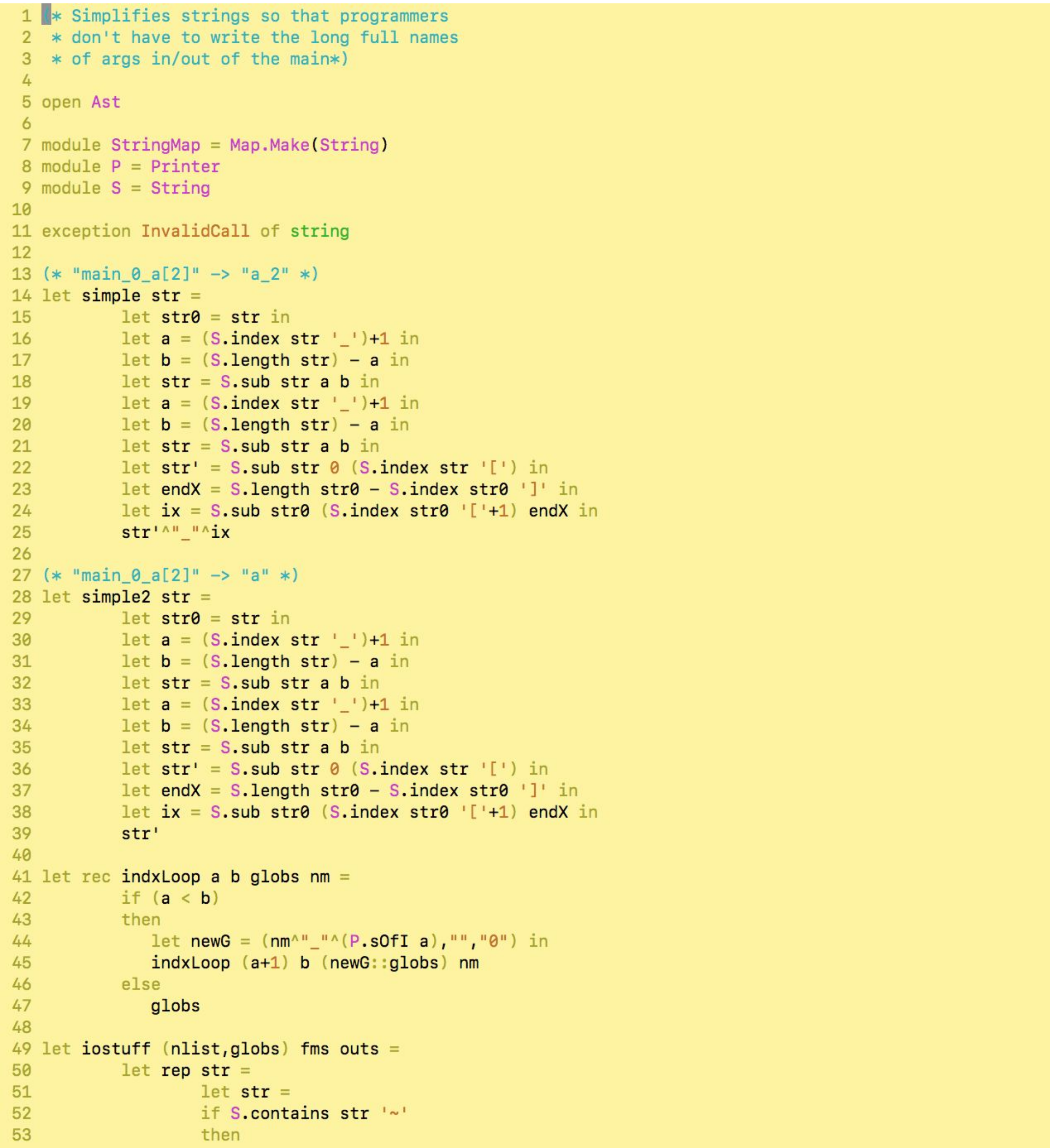
```
 | call {$1}
 | print {$1}
 | loop {$1}

assignment:
 | binExpr REGASSIGN binExpr boolval{ Assign(true, $1, $3, $4) }
 | binExpr  ASSIGN   binExpr { Assign(false, $1, $3, "0") }

boolval:
 | STAR BOOLLIT STAR {$2}
 | STAR BOOLLIT STAR {$2}

call:
 | ID OPAREN argList CPAREN {Call($1, $3)}

argList:
 | /*Nothing*/ { [] }
 | argList COMMA binExpr { $3::$1 }
 | binExpr { $1::[] }

print:
 | PRINT ID COLON binExpr { Print($2, $4) }

loop:
 | FOR OPAREN ID FROM intExprz TO intExprz CPAREN OCURL lineList CCURL {
            For($3, Range($5, $7), $10)}
 | FOR OPAREN ID TO intExprz CPAREN OCURL lineList CCURL {
            For($3, Range(Lit(0), $5), $8)}

outlist:
 | OUT COLON formalsList SEMI {$3}

```

Listing: ast.ml

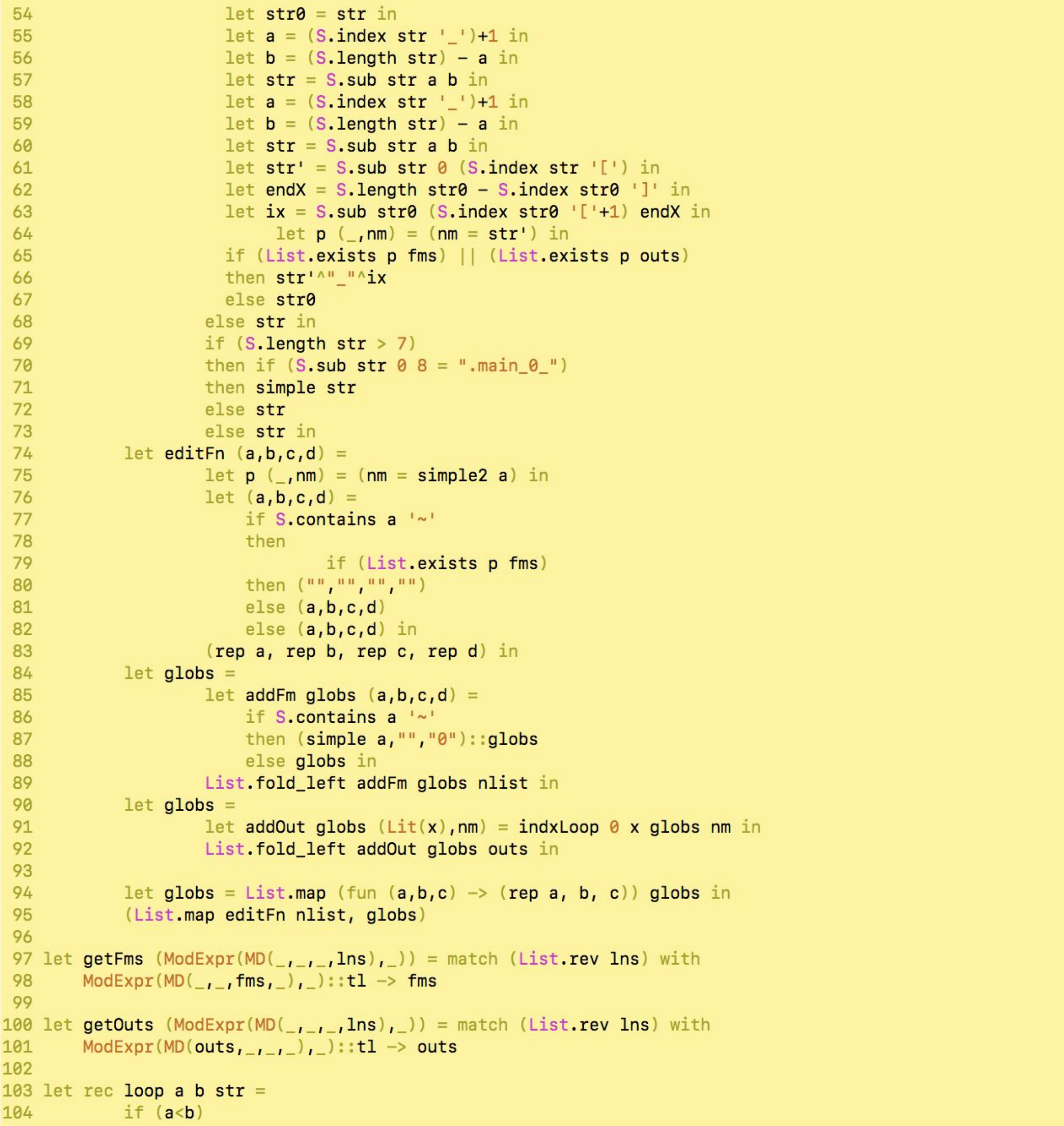


```
type intOp = Add | Sub
type boolOp = And | Or | Nand | Nor | Xor | Xnor
type unOp = Not | Ident
type boolval = One of bool | Zero of bool
type genOp = B of boolOp | U of unOp

type typ = Int | Bus of intExpr

and bind = intExpr * string

and intExpr =
      Lit of int
    | IntId of string
    | IntBinop of intExpr * intOp * intExpr

and  binExpr =
      Buslit of string
    | BoolId of string
    | BoolBinop of binExpr * boolOp * binExpr
    | Unop of unOp * binExpr
    | Assign of bool * binExpr * binExpr * string
    | Index of binExpr * range
    | Print of string * binExpr
    | Call of string * binExpr list
    | For of string * range * binExpr list
    | ModExpr of md * binExpr list
    | Noexpr

and range = Range of intExpr * intExpr

and md = MD of bind list * string * bind list * binExpr list
                        (*outlist   name      formals     line list *)

type program = md list

let opToStr = function
      B(x) -> (match x with
            And  -> "And"
          | Or   -> "Or"
          | Nand -> "Nand"
          | Nor  -> "Nor"
          | Xor  -> "Xor"
          | Xnor -> "Xnor"
      )
    | U(x) -> (match x with
          | Not -> "Not"
          | Ident -> "Ident")
```

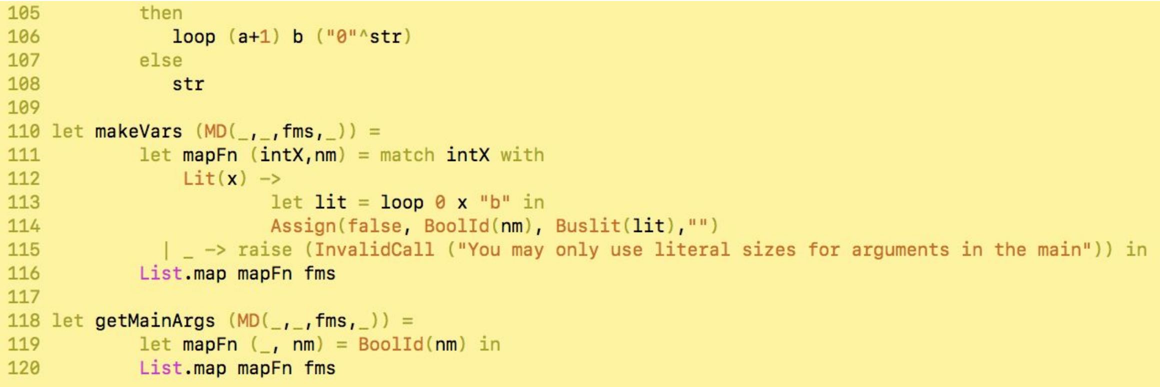
Io.ml



```
(* Simplifies strings so that programmers
 * don't have to write the long full names
 * of args in/out of the main*)

open Ast

module StringMap = Map.Make(String)
module P = Printer
module S = String

exception InvalidCall of string

(* "main_0_a[2]" -> "a_2" *)
let simple str =
        let str0 = str in
        let a = (S.index str '_')+1 in
        let b = (S.length str) - a in
        let str = S.sub str a b in
        let a = (S.index str '_')+1 in
        let b = (S.length str) - a in
        let str = S.sub str a b in
        let str' = S.sub str 0 (S.index str '[') in
        let endX = S.length str0 - S.index str0 ']' in
        let ix = S.sub str0 (S.index str0 '['+1) endX in
        str'^"_"^ix

(* "main_0_a[2]" -> "a" *)
let simple2 str =
        let str0 = str in
        let a = (S.index str '_')+1 in
        let b = (S.length str) - a in
        let str = S.sub str a b in
        let a = (S.index str '_')+1 in
        let b = (S.length str) - a in
        let str = S.sub str a b in
        let str' = S.sub str 0 (S.index str '[') in
        let endX = S.length str0 - S.index str0 ']' in
        let ix = S.sub str0 (S.index str0 '['+1) endX in
        str'

let rec indxLoop a b globs nm =
        if (a < b)
        then
           let newG = (nm^"_"^(P.sOfI a),"","0") in
           indxLoop (a+1) b (newG::globs) nm
        else
           globs

let iostuff (nlist,globs) fms outs =
        let rep str =
                let str =
                if S.contains str '~'
                then
```

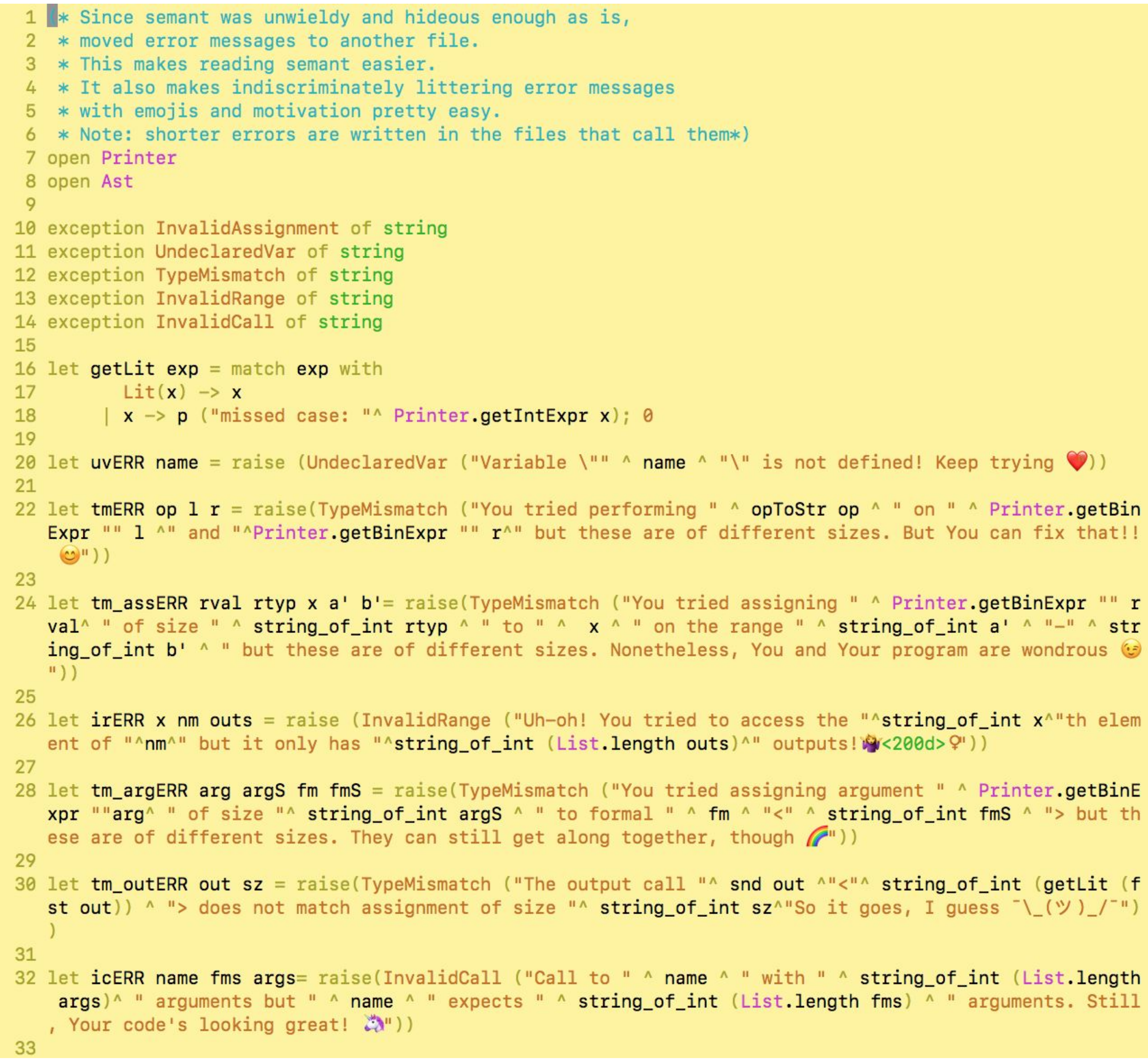


```
                  let str0 = str in
                  let a = (S.index str '_')+1 in
                  let b = (S.length str) - a in
                  let str = S.sub str a b in
                  let a = (S.index str '_')+1 in
                  let b = (S.length str) - a in
                  let str = S.sub str a b in
                  let str' = S.sub str 0 (S.index str '[') in
                  let endX = S.length str0 - S.index str0 ']' in
                  let ix = S.sub str0 (S.index str0 '['+1) endX in
                       let p (_,nm) = (nm = str') in
                  if (List.exists p fms) || (List.exists p outs)
                  then str'^"_"^ix
                  else str0
                else str in
                if (S.length str > 7)
                then if (S.sub str 0 8 = ".main_0_")
                then simple str
                else str
                else str in
        let editFn (a,b,c,d) =
                let p (_,nm) = (nm = simple2 a) in
                let (a,b,c,d) =
                    if S.contains a '~'
                    then
                            if (List.exists p fms)
                    then ("","","","")
                    else (a,b,c,d)
                    else (a,b,c,d) in
                (rep a, rep b, rep c, rep d) in
        let globs =
                let addFm globs (a,b,c,d) =
                    if S.contains a '~'
                    then (simple a,"","0")::globs
                    else globs in
                List.fold_left addFm globs nlist in
        let globs =
                let addOut globs (Lit(x),nm) = indxLoop 0 x globs nm in
                List.fold_left addOut globs outs in

        let globs = List.map (fun (a,b,c) -> (rep a, b, c)) globs in
        (List.map editFn nlist, globs)

let getFms (ModExpr(MD(_,_,_,lns),_)) = match (List.rev lns) with
    ModExpr(MD(_,_,fms,_),_)::tl -> fms

let getOuts (ModExpr(MD(_,_,_,lns),_)) = match (List.rev lns) with
    ModExpr(MD(outs,_,_,_),_)::tl -> outs

let rec loop a b str =
        if (a<b)
```



```
        then
           loop (a+1) b ("0"^str)
        else
           str

let makeVars (MD(_,_,fms,_)) =
        let mapFn (intX,nm) = match intX with
            Lit(x) ->
                    let lit = loop 0 x "b" in
                    Assign(false, BoolId(nm), Buslit(lit),"")
          | _ -> raise (InvalidCall ("You may only use literal sizes for arguments in the main")) in
        List.map mapFn fms

let getMainArgs (MD(_,_,fms,_)) =
        let mapFn (_, nm) = BoolId(nm) in
        List.map mapFn fms
```

ERR.ml



```
(* Since semant was unwieldy and hideous enough as is,
 * moved error messages to another file.
 * This makes reading semant easier.
 * It also makes indiscriminately littering error messages
 * with emojis and motivation pretty easy.
 * Note: shorter errors are written in the files that call them*)
open Printer
open Ast

exception InvalidAssignment of string
exception UndeclaredVar of string
exception TypeMismatch of string
exception InvalidRange of string
exception InvalidCall of string

let getLit exp = match exp with
       Lit(x) -> x
     | x -> p ("missed case: "^ Printer.getIntExpr x); 0

let uvERR name = raise (UndeclaredVar ("Variable \"" ^ name ^ "\" is not defined! Keep trying ❤️))

let tmERR op l r = raise(TypeMismatch ("You tried performing " ^ opToStr op ^ " on " ^ Printer.getBinExpr "" l ^" and "^Printer.getBinExpr "" r^" but these are of different sizes. But You can fix that!! 😊"))

let tm_assERR rval rtyp x a' b'= raise(TypeMismatch ("You tried assigning " ^ Printer.getBinExpr "" rval^ " of size " ^ string_of_int rtyp ^ " to " ^  x ^ " on the range " ^ string_of_int a' ^ "-" ^ string_of_int b' ^ " but these are of different sizes. Nonetheless, You and Your program are wondrous 😉"))

let irERR x nm outs = raise (InvalidRange ("Uh-oh! You tried to access the "^string_of_int x^"th element of "^nm^" but it only has "^string_of_int (List.length outs)^" outputs!🤷<200d>♀"))

let tm_argERR arg argS fm fmS = raise(TypeMismatch ("You tried assigning argument " ^ Printer.getBinExpr ""arg^ " of size "^ string_of_int argS ^ " to formal " ^ fm ^ "<" ^ string_of_int fmS ^ "> but these are of different sizes. They can still get along together, though 🌈"))

let tm_outERR out sz = raise(TypeMismatch ("The output call "^ snd out ^"<"^ string_of_int (getLit (fst out)) ^ "> does not match assignment of size "^ string_of_int sz^"So it goes, I guess ¯\_(ツ)_/¯"))

let icERR name fms args= raise(InvalidCall ("Call to " ^ name ^ " with " ^ string_of_int (List.length args)^ " arguments but " ^ name ^ " expects " ^ string_of_int (List.length fms) ^ " arguments. Still, Your code's looking great! 🦄"))

```

Test Program alu.uni and target code (LLVM)

alu.uni:

```
/**
      000 -> A+B
      001 -> A-B
      100 -> A&B
      101 -> A|B
      110 -> A xor B
      111 -> Not A
**/

main(){
      a = 00111111b;
      b = 11000100b;
      x = alu(111b,a,b);
      print x: x;
      out:;
}

alu(instr<3>,a<n>,b<n>){
      m = math(a,b,instr[0]);
      l = logic(a,b,instr[0:1]);
      o = mux(m,l,instr[2]);
      out:o<n>;
}

math(a<n>,b<n>,sel){
      bc = comp(b);
      bnew = mux(b,bc,sel);
      sum = add(a,bnew);
      out:sum<n>;
}

comp(a<n>){
      a1 = not a;
      for (i from 1 to n-1){
            one[i] = 0b;
      };
      one[0] = 1b;
      cmp = add(a1,one);
      out:cmp<n>;
}

fA(a, b, carryIn){
      axb = a xor b;
    sum = axb xor carryIn;
    carry = (axb and carryIn) or (a and b);
      out: sum, carry;
}

add(a<n>,b<n>){
      c[0] = 0b;

      for (i from 0 to n-1){
            sum[i] = fA(a[i],b[i],c[i])[sum];
            c[i+1] = fA(a[i],b[i],c[i])[carry];
      };
      out:sum<n>;
}

mux(a<n>,b<n>,s){
      for (i from 0 to n-1){
            as[i] = a[i] and not s;
            bs[i] = b[i] and s;
            o[i] = as[i] or bs[i];
      };
      out:o<n>;
}

mux2(a<n>,b<n>,c<n>,d<n>,sel<2>){
      m1 = mux(a,b,sel[0]);
      m2 = mux(c,d,sel[0]);
      m  = mux(m1,m2,sel[1]);
      out:m<n>;
}

logic(a<n>, b<n>, sel<2>){
      AND = a and b;
      OR = a or b;
      XOR = a xor b;
      NOT = not a;
      o = mux2(AND,OR,XOR,NOT,sel);
      out:o<n>;
}
```

alu.uni:

```
; ModuleID = 'Unic'
source_filename = "Unic"

@fmt = private unnamed_addr constant [4 x i8] c"%d\0A\00"
@fmt.1 = private unnamed_addr constant [4 x i8] c"%g\0A\00"
@prt = private unnamed_addr constant [10 x i8] c"x[0]: %d\0A\00"
@prt.2 = private unnamed_addr constant [10 x i8] c"x[1]: %d\0A\00"
@prt.3 = private unnamed_addr constant [10 x i8] c"x[2]: %d\0A\00"
@prt.4 = private unnamed_addr constant [10 x i8] c"x[3]: %d\0A\00"
@prt.5 = private unnamed_addr constant [10 x i8] c"x[4]: %d\0A\00"
@prt.6 = private unnamed_addr constant [10 x i8] c"x[5]: %d\0A\00"
@prt.7 = private unnamed_addr constant [10 x i8] c"x[6]: %d\0A\00"
@prt.8 = private unnamed_addr constant [10 x i8] c"x[7]: %d\0A\00"

declare i32 @printf(i8*, ...)

declare i32 @printbig(i32)

define i1 @tick() {
entry:
  %x_7 = alloca i1
  %x_6 = alloca i1
  %x_5 = alloca i1
  %x_4 = alloca i1
  %x_3 = alloca i1
  %x_2 = alloca i1
  %".alu_0_o[7]" = alloca i1
  %".alu_0_o[6]" = alloca i1
  %".alu_0_o[5]" = alloca i1
  %".alu_0_o[4]" = alloca i1
  %".alu_0_o[3]" = alloca i1
  %".alu_0_o[2]" = alloca i1
  %".mux_4_o[7]" = alloca i1
  %".mux_4_o[6]" = alloca i1
```

```
%".mux_4_o[5]" = alloca i1
%".mux_4_o[4]" = alloca i1
%".mux_4_o[3]" = alloca i1
%".mux_4_o[2]" = alloca i1
%"#582[0]" = alloca i1
%".mux_4_as[7]" = alloca i1
%"#577[0]" = alloca i1
%".alu_0_m[7]" = alloca i1
%"#575[0]" = alloca i1
%".mux_4_as[6]" = alloca i1
%"#570[0]" = alloca i1
%".alu_0_m[6]" = alloca i1
%"#568[0]" = alloca i1
%".mux_4_as[5]" = alloca i1
%"#563[0]" = alloca i1
%".alu_0_m[5]" = alloca i1
%"#561[0]" = alloca i1
%".mux_4_as[4]" = alloca i1
%"#556[0]" = alloca i1
%".alu_0_m[4]" = alloca i1
%"#554[0]" = alloca i1
%".mux_4_as[3]" = alloca i1
%"#549[0]" = alloca i1
%".alu_0_m[3]" = alloca i1
%"#547[0]" = alloca i1
%".mux_4_as[2]" = alloca i1
%"#542[0]" = alloca i1
%".alu_0_m[2]" = alloca i1
%".math_0_sum[7]" = alloca i1
%".math_0_sum[6]" = alloca i1
%".math_0_sum[5]" = alloca i1
%".math_0_sum[4]" = alloca i1
%".math_0_sum[3]" = alloca i1
%".math_0_sum[2]" = alloca i1
%".add_1_sum[7]" = alloca i1
%".add_1_sum[6]" = alloca i1
%".add_1_sum[5]" = alloca i1
%".add_1_sum[4]" = alloca i1
%".add_1_sum[3]" = alloca i1
%".add_1_sum[2]" = alloca i1
%".fA_31_sum[0]" = alloca i1
%".fA_30_sum[0]" = alloca i1
%".fA_30_carry[0]" = alloca i1
```

```
%"#300[0]" = alloca i1
%".fA_29_sum[0]" = alloca i1
%".fA_28_sum[0]" = alloca i1
%".fA_28_carry[0]" = alloca i1
%"#284[0]" = alloca i1
%".fA_27_sum[0]" = alloca i1
%".fA_26_sum[0]" = alloca i1
%".fA_26_carry[0]" = alloca i1
%"#268[0]" = alloca i1
%".fA_25_sum[0]" = alloca i1
%".fA_24_sum[0]" = alloca i1
%".fA_24_carry[0]" = alloca i1
%"#252[0]" = alloca i1
%".fA_23_sum[0]" = alloca i1
%".fA_22_sum[0]" = alloca i1
%".fA_22_carry[0]" = alloca i1
%"#236[0]" = alloca i1
%".fA_21_sum[0]" = alloca i1
%".fA_20_sum[0]" = alloca i1
%".fA_20_carry[0]" = alloca i1
%"#220[0]" = alloca i1
%".fA_19_sum[0]" = alloca i1
%"#218[0]" = alloca i1
%"#219[0]" = alloca i1
%"#226[0]" = alloca i1
%"#234[0]" = alloca i1
%"#235[0]" = alloca i1
%"#242[0]" = alloca i1
%"#250[0]" = alloca i1
%"#251[0]" = alloca i1
%"#258[0]" = alloca i1
%"#266[0]" = alloca i1
%"#267[0]" = alloca i1
%"#274[0]" = alloca i1
%"#282[0]" = alloca i1
%"#283[0]" = alloca i1
%"#290[0]" = alloca i1
%"#298[0]" = alloca i1
%"#299[0]" = alloca i1
%"#306[0]" = alloca i1
%x_1 = alloca i1
%".alu_0_o[1]" = alloca i1
%".mux_4_o[1]" = alloca i1
```

```
%"#540[0]" = alloca i1
%".mux_4_as[1]" = alloca i1
%"#535[0]" = alloca i1
%".alu_0_m[1]" = alloca i1
%".math_0_sum[1]" = alloca i1
%".add_1_sum[1]" = alloca i1
%".add_1_c[8]" = alloca i1
%".fA_31_carry[0]" = alloca i1
%"#308[0]" = alloca i1
%".fA_31_axb[0]" = alloca i1
%"#307[0]" = alloca i1
%".add_1_c[7]" = alloca i1
%"#303[0]" = alloca i1
%".fA_30_axb[0]" = alloca i1
%".fA_29_carry[0]" = alloca i1
%"#292[0]" = alloca i1
%".fA_29_axb[0]" = alloca i1
%"#291[0]" = alloca i1
%".add_1_c[6]" = alloca i1
%"#287[0]" = alloca i1
%".fA_28_axb[0]" = alloca i1
%".fA_27_carry[0]" = alloca i1
%"#276[0]" = alloca i1
%".fA_27_axb[0]" = alloca i1
%"#275[0]" = alloca i1
%".add_1_c[5]" = alloca i1
%"#271[0]" = alloca i1
%".fA_26_axb[0]" = alloca i1
%".fA_25_carry[0]" = alloca i1
%"#260[0]" = alloca i1
%".fA_25_axb[0]" = alloca i1
%"#259[0]" = alloca i1
%".add_1_c[4]" = alloca i1
%"#255[0]" = alloca i1
%".fA_24_axb[0]" = alloca i1
%".fA_23_carry[0]" = alloca i1
%"#244[0]" = alloca i1
%".fA_23_axb[0]" = alloca i1
%"#243[0]" = alloca i1
%".add_1_c[3]" = alloca i1
%"#239[0]" = alloca i1
%".fA_22_axb[0]" = alloca i1
%".fA_21_carry[0]" = alloca i1
```

```
%"#228[0]" = alloca i1
%".fA_21_axb[0]" = alloca i1
%"#227[0]" = alloca i1
%".add_1_c[2]" = alloca i1
%"#223[0]" = alloca i1
%".fA_20_axb[0]" = alloca i1
%".fA_19_carry[0]" = alloca i1
%"#212[0]" = alloca i1
%".fA_19_axb[0]" = alloca i1
%".fA_18_sum[0]" = alloca i1
%".fA_18_carry[0]" = alloca i1
%"#204[0]" = alloca i1
%"#207[0]" = alloca i1
%".fA_18_axb[0]" = alloca i1
%".fA_17_sum[0]" = alloca i1
%"#201[0]" = alloca i1
%"#202[0]" = alloca i1
%"#203[0]" = alloca i1
%"#206[0]" = alloca i1
%"#209[0]" = alloca i1
%"#210[0]" = alloca i1
%"#217[0]" = alloca i1
%"#222[0]" = alloca i1
%"#225[0]" = alloca i1
%"#233[0]" = alloca i1
%"#238[0]" = alloca i1
%"#241[0]" = alloca i1
%"#249[0]" = alloca i1
%"#254[0]" = alloca i1
%"#257[0]" = alloca i1
%"#265[0]" = alloca i1
%"#270[0]" = alloca i1
%"#273[0]" = alloca i1
%"#281[0]" = alloca i1
%"#286[0]" = alloca i1
%"#289[0]" = alloca i1
%"#297[0]" = alloca i1
%"#302[0]" = alloca i1
%"#305[0]" = alloca i1
%x_0 = alloca i1
%".alu_0_o[0]" = alloca i1
%".mux_4_o[0]" = alloca i1
%"#533[0]" = alloca i1
```

```
%".mux_4_as[0]" = alloca i1
%"#528[0]" = alloca i1
%".alu_0_m[0]" = alloca i1
%".math_0_sum[0]" = alloca i1
%".add_1_sum[0]" = alloca i1
%"#311[0]" = alloca i1
%"#310[0]" = alloca i1
%".math_0_bnew[7]" = alloca i1
%"#295[0]" = alloca i1
%"#294[0]" = alloca i1
%".math_0_bnew[6]" = alloca i1
%"#279[0]" = alloca i1
%"#278[0]" = alloca i1
%".math_0_bnew[5]" = alloca i1
%"#263[0]" = alloca i1
%"#262[0]" = alloca i1
%".math_0_bnew[4]" = alloca i1
%"#247[0]" = alloca i1
%"#246[0]" = alloca i1
%".math_0_bnew[3]" = alloca i1
%"#231[0]" = alloca i1
%"#230[0]" = alloca i1
%".math_0_bnew[2]" = alloca i1
%"#215[0]" = alloca i1
%"#214[0]" = alloca i1
%".math_0_bnew[1]" = alloca i1
%"#211[0]" = alloca i1
%".add_1_c[1]" = alloca i1
%".fA_17_carry[0]" = alloca i1
%"#196[0]" = alloca i1
%".fA_17_axb[0]" = alloca i1
%".fA_16_sum[0]" = alloca i1
%".fA_16_carry[0]" = alloca i1
%"#188[0]" = alloca i1
%"#191[0]" = alloca i1
%".fA_16_axb[0]" = alloca i1
%".mux_0_o[7]" = alloca i1
%".mux_0_o[6]" = alloca i1
%".mux_0_o[5]" = alloca i1
%".mux_0_o[4]" = alloca i1
%".mux_0_o[3]" = alloca i1
%".mux_0_o[2]" = alloca i1
%".mux_0_o[1]" = alloca i1
```

```
%"#183[0]" = alloca i1
%".mux_0_bs[7]" = alloca i1
%"#180[0]" = alloca i1
%".math_0_bc[7]" = alloca i1
%"#176[0]" = alloca i1
%".mux_0_bs[6]" = alloca i1
%"#173[0]" = alloca i1
%".math_0_bc[6]" = alloca i1
%"#169[0]" = alloca i1
%".mux_0_bs[5]" = alloca i1
%"#166[0]" = alloca i1
%".math_0_bc[5]" = alloca i1
%"#162[0]" = alloca i1
%".mux_0_bs[4]" = alloca i1
%"#159[0]" = alloca i1
%".math_0_bc[4]" = alloca i1
%"#155[0]" = alloca i1
%".mux_0_bs[3]" = alloca i1
%"#152[0]" = alloca i1
%".math_0_bc[3]" = alloca i1
%"#148[0]" = alloca i1
%".mux_0_bs[2]" = alloca i1
%"#145[0]" = alloca i1
%".math_0_bc[2]" = alloca i1
%"#141[0]" = alloca i1
%".mux_0_bs[1]" = alloca i1
%"#138[0]" = alloca i1
%".math_0_bc[1]" = alloca i1
%".comp_0_cmp[7]" = alloca i1
%".comp_0_cmp[6]" = alloca i1
%".comp_0_cmp[5]" = alloca i1
%".comp_0_cmp[4]" = alloca i1
%".comp_0_cmp[3]" = alloca i1
%".comp_0_cmp[2]" = alloca i1
%".comp_0_cmp[1]" = alloca i1
%".add_0_sum[7]" = alloca i1
%".add_0_sum[6]" = alloca i1
%".add_0_sum[5]" = alloca i1
%".add_0_sum[4]" = alloca i1
%".add_0_sum[3]" = alloca i1
%".add_0_sum[2]" = alloca i1
%".add_0_sum[1]" = alloca i1
%".fA_15_sum[0]" = alloca i1
```

```
%".fA_14_sum[0]" = alloca i1
%".fA_14_carry[0]" = alloca i1
%"#116[0]" = alloca i1
%".fA_13_sum[0]" = alloca i1
%".fA_12_sum[0]" = alloca i1
%".fA_12_carry[0]" = alloca i1
%"#100[0]" = alloca i1
%".fA_11_sum[0]" = alloca i1
%".fA_10_sum[0]" = alloca i1
%".fA_10_carry[0]" = alloca i1
%"#84[0]" = alloca i1
%".fA_9_sum[0]" = alloca i1
%".fA_8_sum[0]" = alloca i1
%".fA_8_carry[0]" = alloca i1
%"#68[0]" = alloca i1
%".fA_7_sum[0]" = alloca i1
%".fA_6_sum[0]" = alloca i1
%".fA_6_carry[0]" = alloca i1
%"#52[0]" = alloca i1
%".fA_5_sum[0]" = alloca i1
%".fA_4_sum[0]" = alloca i1
%".fA_4_carry[0]" = alloca i1
%"#36[0]" = alloca i1
%".fA_3_sum[0]" = alloca i1
%".fA_2_sum[0]" = alloca i1
%".fA_2_carry[0]" = alloca i1
%"#20[0]" = alloca i1
%".fA_1_sum[0]" = alloca i1
%"#18[0]" = alloca i1
%"#19[0]" = alloca i1
%"#26[0]" = alloca i1
%"#34[0]" = alloca i1
%"#35[0]" = alloca i1
%"#42[0]" = alloca i1
%"#50[0]" = alloca i1
%"#51[0]" = alloca i1
%"#58[0]" = alloca i1
%"#66[0]" = alloca i1
%"#67[0]" = alloca i1
%"#74[0]" = alloca i1
%"#82[0]" = alloca i1
%"#83[0]" = alloca i1
%"#90[0]" = alloca i1
```

```
%"#98[0]" = alloca i1
%"#99[0]" = alloca i1
%"#106[0]" = alloca i1
%"#114[0]" = alloca i1
%"#115[0]" = alloca i1
%"#122[0]" = alloca i1
%"#185[0]" = alloca i1
%"#190[0]" = alloca i1
%"#193[0]" = alloca i1
%"#199[0]" = alloca i1
%"#198[0]" = alloca i1
%".math_0_bnew[0]" = alloca i1
%".mux_0_o[0]" = alloca i1
%"#182[0]" = alloca i1
%".mux_0_as[7]" = alloca i1
%"#175[0]" = alloca i1
%".mux_0_as[6]" = alloca i1
%"#168[0]" = alloca i1
%".mux_0_as[5]" = alloca i1
%"#161[0]" = alloca i1
%".mux_0_as[4]" = alloca i1
%"#154[0]" = alloca i1
%".mux_0_as[3]" = alloca i1
%"#147[0]" = alloca i1
%".mux_0_as[2]" = alloca i1
%"#140[0]" = alloca i1
%".mux_0_as[1]" = alloca i1
%"#133[0]" = alloca i1
%"#134[0]" = alloca i1
%".mux_0_bs[0]" = alloca i1
%".mux_0_as[0]" = alloca i1
%"#131[0]" = alloca i1
%".math_0_bc[0]" = alloca i1
%".comp_0_cmp[0]" = alloca i1
%".add_0_sum[0]" = alloca i1
%".add_0_c[8]" = alloca i1
%".fA_15_carry[0]" = alloca i1
%"#124[0]" = alloca i1
%"#127[0]" = alloca i1
%"#125[0]" = alloca i1
%".fA_15_axb[0]" = alloca i1
%"#123[0]" = alloca i1
%".add_0_c[7]" = alloca i1
```

```
%"#119[0]" = alloca i1
%".fA_14_axb[0]" = alloca i1
%".fA_13_carry[0]" = alloca i1
%"#108[0]" = alloca i1
%"#111[0]" = alloca i1
%"#109[0]" = alloca i1
%".fA_13_axb[0]" = alloca i1
%"#107[0]" = alloca i1
%".add_0_c[6]" = alloca i1
%"#103[0]" = alloca i1
%".fA_12_axb[0]" = alloca i1
%".fA_11_carry[0]" = alloca i1
%"#92[0]" = alloca i1
%"#95[0]" = alloca i1
%"#93[0]" = alloca i1
%".fA_11_axb[0]" = alloca i1
%"#91[0]" = alloca i1
%".add_0_c[5]" = alloca i1
%"#87[0]" = alloca i1
%".fA_10_axb[0]" = alloca i1
%".fA_9_carry[0]" = alloca i1
%"#76[0]" = alloca i1
%"#79[0]" = alloca i1
%"#77[0]" = alloca i1
%".fA_9_axb[0]" = alloca i1
%"#75[0]" = alloca i1
%".add_0_c[4]" = alloca i1
%"#71[0]" = alloca i1
%".fA_8_axb[0]" = alloca i1
%".fA_7_carry[0]" = alloca i1
%"#60[0]" = alloca i1
%"#63[0]" = alloca i1
%"#61[0]" = alloca i1
%".fA_7_axb[0]" = alloca i1
%"#59[0]" = alloca i1
%".add_0_c[3]" = alloca i1
%"#55[0]" = alloca i1
%".fA_6_axb[0]" = alloca i1
%".fA_5_carry[0]" = alloca i1
%"#44[0]" = alloca i1
%"#47[0]" = alloca i1
%"#45[0]" = alloca i1
%".fA_5_axb[0]" = alloca i1
```

```
%"#43[0]" = alloca i1
%".add_0_c[2]" = alloca i1
%"#39[0]" = alloca i1
%".fA_4_axb[0]" = alloca i1
%".fA_3_carry[0]" = alloca i1
%"#28[0]" = alloca i1
%"#31[0]" = alloca i1
%"#29[0]" = alloca i1
%".fA_3_axb[0]" = alloca i1
%"#27[0]" = alloca i1
%".add_0_c[1]" = alloca i1
%"#23[0]" = alloca i1
%".fA_2_axb[0]" = alloca i1
%".fA_1_carry[0]" = alloca i1
%"#12[0]" = alloca i1
%"#15[0]" = alloca i1
%"#13[0]" = alloca i1
%".fA_1_axb[0]" = alloca i1
%".fA_0_sum[0]" = alloca i1
%".fA_0_carry[0]" = alloca i1
%"#4[0]" = alloca i1
%"#7[0]" = alloca i1
%".fA_0_axb[0]" = alloca i1
%"#1[0]" = alloca i1
%"#0[0]" = alloca i1
%"#2[0]" = alloca i1
%"#3[0]" = alloca i1
%"#6[0]" = alloca i1
%"#5[0]" = alloca i1
%"#9[0]" = alloca i1
%"#8[0]" = alloca i1
%"#10[0]" = alloca i1
%".comp_0_a1[0]" = alloca i1
%"#17[0]" = alloca i1
%"#16[0]" = alloca i1
%"#22[0]" = alloca i1
%"#21[0]" = alloca i1
%"#25[0]" = alloca i1
%"#24[0]" = alloca i1
%".comp_0_a1[1]" = alloca i1
%"#33[0]" = alloca i1
%"#32[0]" = alloca i1
%"#38[0]" = alloca i1
```

```
%"#37[0]" = alloca i1
%"#41[0]" = alloca i1
%"#40[0]" = alloca i1
%".comp_0_a1[2]" = alloca i1
%"#49[0]" = alloca i1
%"#48[0]" = alloca i1
%"#54[0]" = alloca i1
%"#53[0]" = alloca i1
%"#57[0]" = alloca i1
%"#56[0]" = alloca i1
%".comp_0_a1[3]" = alloca i1
%"#65[0]" = alloca i1
%"#64[0]" = alloca i1
%"#70[0]" = alloca i1
%"#69[0]" = alloca i1
%"#73[0]" = alloca i1
%"#72[0]" = alloca i1
%".comp_0_a1[4]" = alloca i1
%"#81[0]" = alloca i1
%"#80[0]" = alloca i1
%"#86[0]" = alloca i1
%"#85[0]" = alloca i1
%"#89[0]" = alloca i1
%"#88[0]" = alloca i1
%".comp_0_a1[5]" = alloca i1
%"#97[0]" = alloca i1
%"#96[0]" = alloca i1
%"#102[0]" = alloca i1
%"#101[0]" = alloca i1
%"#105[0]" = alloca i1
%"#104[0]" = alloca i1
%".comp_0_a1[6]" = alloca i1
%"#113[0]" = alloca i1
%"#112[0]" = alloca i1
%"#118[0]" = alloca i1
%"#117[0]" = alloca i1
%"#121[0]" = alloca i1
%"#120[0]" = alloca i1
%".comp_0_a1[7]" = alloca i1
%"#128[0]" = alloca i1
%"#135[0]" = alloca i1
%"#142[0]" = alloca i1
%"#149[0]" = alloca i1
```

```
%"#156[0]" = alloca i1
%"#163[0]" = alloca i1
%"#170[0]" = alloca i1
%"#177[0]" = alloca i1
%"#184[0]" = alloca i1
%"#186[0]" = alloca i1
%"#187[0]" = alloca i1
%"#189[0]" = alloca i1
%"#192[0]" = alloca i1
%"#194[0]" = alloca i1
%"#197[0]" = alloca i1
%"#200[0]" = alloca i1
%"#205[0]" = alloca i1
%"#208[0]" = alloca i1
%"#213[0]" = alloca i1
%"#216[0]" = alloca i1
%"#221[0]" = alloca i1
%"#224[0]" = alloca i1
%"#229[0]" = alloca i1
%"#232[0]" = alloca i1
%"#237[0]" = alloca i1
%"#240[0]" = alloca i1
%"#245[0]" = alloca i1
%"#248[0]" = alloca i1
%"#253[0]" = alloca i1
%"#256[0]" = alloca i1
%"#261[0]" = alloca i1
%"#264[0]" = alloca i1
%"#269[0]" = alloca i1
%"#272[0]" = alloca i1
%"#277[0]" = alloca i1
%"#280[0]" = alloca i1
%"#285[0]" = alloca i1
%"#288[0]" = alloca i1
%"#293[0]" = alloca i1
%"#296[0]" = alloca i1
%"#301[0]" = alloca i1
%"#304[0]" = alloca i1
%"#309[0]" = alloca i1
%"#583[0]" = alloca i1
%".mux_4_bs[7]" = alloca i1
%"#580[0]" = alloca i1
%"#581[0]" = alloca i1
```

```
%".alu_0_l[7]" = alloca i1
%"#579[0]" = alloca i1
%"#578[0]" = alloca i1
%"#576[0]" = alloca i1
%".mux_4_bs[6]" = alloca i1
%"#573[0]" = alloca i1
%"#574[0]" = alloca i1
%".alu_0_l[6]" = alloca i1
%"#572[0]" = alloca i1
%"#571[0]" = alloca i1
%"#569[0]" = alloca i1
%".mux_4_bs[5]" = alloca i1
%"#566[0]" = alloca i1
%"#567[0]" = alloca i1
%".alu_0_l[5]" = alloca i1
%"#565[0]" = alloca i1
%"#564[0]" = alloca i1
%"#562[0]" = alloca i1
%".mux_4_bs[4]" = alloca i1
%"#559[0]" = alloca i1
%"#560[0]" = alloca i1
%".alu_0_l[4]" = alloca i1
%"#558[0]" = alloca i1
%"#557[0]" = alloca i1
%"#555[0]" = alloca i1
%".mux_4_bs[3]" = alloca i1
%"#552[0]" = alloca i1
%"#553[0]" = alloca i1
%".alu_0_l[3]" = alloca i1
%"#551[0]" = alloca i1
%"#550[0]" = alloca i1
%"#548[0]" = alloca i1
%".mux_4_bs[2]" = alloca i1
%"#545[0]" = alloca i1
%"#546[0]" = alloca i1
%".alu_0_l[2]" = alloca i1
%"#544[0]" = alloca i1
%"#543[0]" = alloca i1
%"#541[0]" = alloca i1
%".mux_4_bs[1]" = alloca i1
%"#538[0]" = alloca i1
%"#539[0]" = alloca i1
%".alu_0_l[1]" = alloca i1
```

```
%"#537[0]" = alloca i1
%"#536[0]" = alloca i1
%"#534[0]" = alloca i1
%".mux_4_bs[0]" = alloca i1
%"#531[0]" = alloca i1
%"#532[0]" = alloca i1
%".alu_0_l[0]" = alloca i1
%"#530[0]" = alloca i1
%"#529[0]" = alloca i1
%".logic_0_o[7]" = alloca i1
%".logic_0_o[6]" = alloca i1
%".logic_0_o[5]" = alloca i1
%".logic_0_o[4]" = alloca i1
%".logic_0_o[3]" = alloca i1
%".logic_0_o[2]" = alloca i1
%".logic_0_o[1]" = alloca i1
%".logic_0_o[0]" = alloca i1
%".mux2_0_m[7]" = alloca i1
%".mux2_0_m[6]" = alloca i1
%".mux2_0_m[5]" = alloca i1
%".mux2_0_m[4]" = alloca i1
%".mux2_0_m[3]" = alloca i1
%".mux2_0_m[2]" = alloca i1
%".mux2_0_m[1]" = alloca i1
%".mux2_0_m[0]" = alloca i1
%".mux_3_o[7]" = alloca i1
%".mux_3_o[6]" = alloca i1
%".mux_3_o[5]" = alloca i1
%".mux_3_o[4]" = alloca i1
%".mux_3_o[3]" = alloca i1
%".mux_3_o[2]" = alloca i1
%".mux_3_o[1]" = alloca i1
%".mux_3_o[0]" = alloca i1
%"#526[0]" = alloca i1
%"#527[0]" = alloca i1
%".mux_3_bs[7]" = alloca i1
%".mux_3_as[7]" = alloca i1
%"#523[0]" = alloca i1
%"#525[0]" = alloca i1
%"#524[1]" = alloca i1
%"#524[0]" = alloca i1
%".mux2_0_m2[7]" = alloca i1
%"#519[0]" = alloca i1
```

```
%"#522[0]" = alloca i1
%"#521[0]" = alloca i1
%"#520[1]" = alloca i1
%"#520[0]" = alloca i1
%".mux2_0_m1[7]" = alloca i1
%"#517[0]" = alloca i1
%"#518[0]" = alloca i1
%".mux_3_bs[6]" = alloca i1
%".mux_3_as[6]" = alloca i1
%"#514[0]" = alloca i1
%"#516[0]" = alloca i1
%"#515[1]" = alloca i1
%"#515[0]" = alloca i1
%".mux2_0_m2[6]" = alloca i1
%"#510[0]" = alloca i1
%"#513[0]" = alloca i1
%"#512[0]" = alloca i1
%"#511[1]" = alloca i1
%"#511[0]" = alloca i1
%".mux2_0_m1[6]" = alloca i1
%"#508[0]" = alloca i1
%"#509[0]" = alloca i1
%".mux_3_bs[5]" = alloca i1
%".mux_3_as[5]" = alloca i1
%"#505[0]" = alloca i1
%"#507[0]" = alloca i1
%"#506[1]" = alloca i1
%"#506[0]" = alloca i1
%".mux2_0_m2[5]" = alloca i1
%"#501[0]" = alloca i1
%"#504[0]" = alloca i1
%"#503[0]" = alloca i1
%"#502[1]" = alloca i1
%"#502[0]" = alloca i1
%".mux2_0_m1[5]" = alloca i1
%"#499[0]" = alloca i1
%"#500[0]" = alloca i1
%".mux_3_bs[4]" = alloca i1
%".mux_3_as[4]" = alloca i1
%"#496[0]" = alloca i1
%"#498[0]" = alloca i1
%"#497[1]" = alloca i1
%"#497[0]" = alloca i1
```

```
%".mux2_0_m2[4]" = alloca i1
%"#492[0]" = alloca i1
%"#495[0]" = alloca i1
%"#494[0]" = alloca i1
%"#493[1]" = alloca i1
%"#493[0]" = alloca i1
%".mux2_0_m1[4]" = alloca i1
%"#490[0]" = alloca i1
%"#491[0]" = alloca i1
%".mux_3_bs[3]" = alloca i1
%".mux_3_as[3]" = alloca i1
%"#487[0]" = alloca i1
%"#489[0]" = alloca i1
%"#488[1]" = alloca i1
%"#488[0]" = alloca i1
%".mux2_0_m2[3]" = alloca i1
%"#483[0]" = alloca i1
%"#486[0]" = alloca i1
%"#485[0]" = alloca i1
%"#484[1]" = alloca i1
%"#484[0]" = alloca i1
%".mux2_0_m1[3]" = alloca i1
%"#481[0]" = alloca i1
%"#482[0]" = alloca i1
%".mux_3_bs[2]" = alloca i1
%".mux_3_as[2]" = alloca i1
%"#478[0]" = alloca i1
%"#480[0]" = alloca i1
%"#479[1]" = alloca i1
%"#479[0]" = alloca i1
%".mux2_0_m2[2]" = alloca i1
%"#474[0]" = alloca i1
%"#477[0]" = alloca i1
%"#476[0]" = alloca i1
%"#475[1]" = alloca i1
%"#475[0]" = alloca i1
%".mux2_0_m1[2]" = alloca i1
%"#472[0]" = alloca i1
%"#473[0]" = alloca i1
%".mux_3_bs[1]" = alloca i1
%".mux_3_as[1]" = alloca i1
%"#469[0]" = alloca i1
%"#471[0]" = alloca i1
```

```
%"#470[1]" = alloca i1
%"#470[0]" = alloca i1
%".mux2_0_m2[1]" = alloca i1
%"#465[0]" = alloca i1
%"#468[0]" = alloca i1
%"#467[0]" = alloca i1
%"#466[1]" = alloca i1
%"#466[0]" = alloca i1
%".mux2_0_m1[1]" = alloca i1
%"#463[0]" = alloca i1
%"#464[0]" = alloca i1
%".mux_3_bs[0]" = alloca i1
%".mux_3_as[0]" = alloca i1
%"#460[0]" = alloca i1
%"#462[0]" = alloca i1
%"#461[1]" = alloca i1
%"#461[0]" = alloca i1
%".mux2_0_m2[0]" = alloca i1
%"#456[0]" = alloca i1
%"#459[0]" = alloca i1
%"#458[0]" = alloca i1
%"#457[1]" = alloca i1
%"#457[0]" = alloca i1
%".mux2_0_m1[0]" = alloca i1
%".mux_2_o[7]" = alloca i1
%".mux_2_o[6]" = alloca i1
%".mux_2_o[5]" = alloca i1
%".mux_2_o[4]" = alloca i1
%".mux_2_o[3]" = alloca i1
%".mux_2_o[2]" = alloca i1
%".mux_2_o[1]" = alloca i1
%".mux_2_o[0]" = alloca i1
%"#454[0]" = alloca i1
%"#455[0]" = alloca i1
%".mux_2_bs[7]" = alloca i1
%".mux_2_as[7]" = alloca i1
%"#451[0]" = alloca i1
%"#453[0]" = alloca i1
%"#452[0]" = alloca i1
%"#452[1]" = alloca i1
%".logic_0_NOT[7]" = alloca i1
%"#447[0]" = alloca i1
%"#450[0]" = alloca i1
```

```
%"#449[0]" = alloca i1
%"#448[0]" = alloca i1
%"#448[1]" = alloca i1
%".logic_0_XOR[7]" = alloca i1
%"#445[0]" = alloca i1
%"#446[0]" = alloca i1
%".mux_2_bs[6]" = alloca i1
%".mux_2_as[6]" = alloca i1
%"#442[0]" = alloca i1
%"#444[0]" = alloca i1
%"#443[0]" = alloca i1
%"#443[1]" = alloca i1
%".logic_0_NOT[6]" = alloca i1
%"#438[0]" = alloca i1
%"#441[0]" = alloca i1
%"#440[0]" = alloca i1
%"#439[0]" = alloca i1
%"#439[1]" = alloca i1
%".logic_0_XOR[6]" = alloca i1
%"#436[0]" = alloca i1
%"#437[0]" = alloca i1
%".mux_2_bs[5]" = alloca i1
%".mux_2_as[5]" = alloca i1
%"#433[0]" = alloca i1
%"#435[0]" = alloca i1
%"#434[0]" = alloca i1
%"#434[1]" = alloca i1
%".logic_0_NOT[5]" = alloca i1
%"#429[0]" = alloca i1
%"#432[0]" = alloca i1
%"#431[0]" = alloca i1
%"#430[0]" = alloca i1
%"#430[1]" = alloca i1
%".logic_0_XOR[5]" = alloca i1
%"#427[0]" = alloca i1
%"#428[0]" = alloca i1
%".mux_2_bs[4]" = alloca i1
%".mux_2_as[4]" = alloca i1
%"#424[0]" = alloca i1
%"#426[0]" = alloca i1
%"#425[0]" = alloca i1
%"#425[1]" = alloca i1
%".logic_0_NOT[4]" = alloca i1
```

```
%"#420[0]" = alloca i1
%"#423[0]" = alloca i1
%"#422[0]" = alloca i1
%"#421[0]" = alloca i1
%"#421[1]" = alloca i1
%".logic_0_XOR[4]" = alloca i1
%"#418[0]" = alloca i1
%"#419[0]" = alloca i1
%".mux_2_bs[3]" = alloca i1
%".mux_2_as[3]" = alloca i1
%"#415[0]" = alloca i1
%"#417[0]" = alloca i1
%"#416[0]" = alloca i1
%"#416[1]" = alloca i1
%".logic_0_NOT[3]" = alloca i1
%"#411[0]" = alloca i1
%"#414[0]" = alloca i1
%"#413[0]" = alloca i1
%"#412[0]" = alloca i1
%"#412[1]" = alloca i1
%".logic_0_XOR[3]" = alloca i1
%"#409[0]" = alloca i1
%"#410[0]" = alloca i1
%".mux_2_bs[2]" = alloca i1
%".mux_2_as[2]" = alloca i1
%"#406[0]" = alloca i1
%"#408[0]" = alloca i1
%"#407[0]" = alloca i1
%"#407[1]" = alloca i1
%".logic_0_NOT[2]" = alloca i1
%"#402[0]" = alloca i1
%"#405[0]" = alloca i1
%"#404[0]" = alloca i1
%"#403[0]" = alloca i1
%"#403[1]" = alloca i1
%".logic_0_XOR[2]" = alloca i1
%"#400[0]" = alloca i1
%"#401[0]" = alloca i1
%".mux_2_bs[1]" = alloca i1
%".mux_2_as[1]" = alloca i1
%"#397[0]" = alloca i1
%"#399[0]" = alloca i1
%"#398[0]" = alloca i1
```

```
%"#398[1]" = alloca i1
%".logic_0_NOT[1]" = alloca i1
%"#393[0]" = alloca i1
%"#396[0]" = alloca i1
%"#395[0]" = alloca i1
%"#394[0]" = alloca i1
%"#394[1]" = alloca i1
%".logic_0_XOR[1]" = alloca i1
%"#391[0]" = alloca i1
%"#392[0]" = alloca i1
%".mux_2_bs[0]" = alloca i1
%".mux_2_as[0]" = alloca i1
%"#388[0]" = alloca i1
%"#390[0]" = alloca i1
%"#389[0]" = alloca i1
%"#389[1]" = alloca i1
%".logic_0_NOT[0]" = alloca i1
%"#384[0]" = alloca i1
%"#387[0]" = alloca i1
%"#386[0]" = alloca i1
%"#385[0]" = alloca i1
%"#385[1]" = alloca i1
%".logic_0_XOR[0]" = alloca i1
%".mux_1_o[7]" = alloca i1
%".mux_1_o[6]" = alloca i1
%".mux_1_o[5]" = alloca i1
%".mux_1_o[4]" = alloca i1
%".mux_1_o[3]" = alloca i1
%".mux_1_o[2]" = alloca i1
%".mux_1_o[1]" = alloca i1
%".mux_1_o[0]" = alloca i1
%"#382[0]" = alloca i1
%"#383[0]" = alloca i1
%".mux_1_bs[7]" = alloca i1
%".mux_1_as[7]" = alloca i1
%"#379[0]" = alloca i1
%"#381[0]" = alloca i1
%"#380[0]" = alloca i1
%"#380[1]" = alloca i1
%".logic_0_OR[7]" = alloca i1
%"#375[0]" = alloca i1
%"#378[0]" = alloca i1
%"#377[0]" = alloca i1
```

```
%"#376[0]" = alloca i1
%"#376[1]" = alloca i1
%".logic_0_AND[7]" = alloca i1
%"#373[0]" = alloca i1
%"#374[0]" = alloca i1
%".mux_1_bs[6]" = alloca i1
%".mux_1_as[6]" = alloca i1
%"#370[0]" = alloca i1
%"#372[0]" = alloca i1
%"#371[0]" = alloca i1
%"#371[1]" = alloca i1
%".logic_0_OR[6]" = alloca i1
%"#366[0]" = alloca i1
%"#369[0]" = alloca i1
%"#368[0]" = alloca i1
%"#367[0]" = alloca i1
%"#367[1]" = alloca i1
%".logic_0_AND[6]" = alloca i1
%"#364[0]" = alloca i1
%"#365[0]" = alloca i1
%".mux_1_bs[5]" = alloca i1
%".mux_1_as[5]" = alloca i1
%"#361[0]" = alloca i1
%"#363[0]" = alloca i1
%"#362[0]" = alloca i1
%"#362[1]" = alloca i1
%".logic_0_OR[5]" = alloca i1
%"#357[0]" = alloca i1
%"#360[0]" = alloca i1
%"#359[0]" = alloca i1
%"#358[0]" = alloca i1
%"#358[1]" = alloca i1
%".logic_0_AND[5]" = alloca i1
%"#355[0]" = alloca i1
%"#356[0]" = alloca i1
%".mux_1_bs[4]" = alloca i1
%".mux_1_as[4]" = alloca i1
%"#352[0]" = alloca i1
%"#354[0]" = alloca i1
%"#353[0]" = alloca i1
%"#353[1]" = alloca i1
%".logic_0_OR[4]" = alloca i1
%"#348[0]" = alloca i1
```

```
%"#351[0]" = alloca i1
%"#350[0]" = alloca i1
%"#349[0]" = alloca i1
%"#349[1]" = alloca i1
%".logic_0_AND[4]" = alloca i1
%"#346[0]" = alloca i1
%"#347[0]" = alloca i1
%".mux_1_bs[3]" = alloca i1
%".mux_1_as[3]" = alloca i1
%"#343[0]" = alloca i1
%"#345[0]" = alloca i1
%"#344[0]" = alloca i1
%"#344[1]" = alloca i1
%".logic_0_OR[3]" = alloca i1
%"#339[0]" = alloca i1
%"#342[0]" = alloca i1
%"#341[0]" = alloca i1
%"#340[0]" = alloca i1
%"#340[1]" = alloca i1
%".logic_0_AND[3]" = alloca i1
%"#337[0]" = alloca i1
%"#338[0]" = alloca i1
%".mux_1_bs[2]" = alloca i1
%".mux_1_as[2]" = alloca i1
%"#334[0]" = alloca i1
%"#336[0]" = alloca i1
%"#335[0]" = alloca i1
%"#335[1]" = alloca i1
%".logic_0_OR[2]" = alloca i1
%"#330[0]" = alloca i1
%"#333[0]" = alloca i1
%"#332[0]" = alloca i1
%"#331[0]" = alloca i1
%"#331[1]" = alloca i1
%".logic_0_AND[2]" = alloca i1
%"#328[0]" = alloca i1
%"#329[0]" = alloca i1
%".mux_1_bs[1]" = alloca i1
%".mux_1_as[1]" = alloca i1
%"#325[0]" = alloca i1
%"#327[0]" = alloca i1
%"#326[0]" = alloca i1
%"#326[1]" = alloca i1
```

```
%".logic_0_OR[1]" = alloca i1
%"#321[0]" = alloca i1
%"#324[0]" = alloca i1
%"#323[0]" = alloca i1
%"#322[0]" = alloca i1
%"#322[1]" = alloca i1
%".logic_0_AND[1]" = alloca i1
%"#319[0]" = alloca i1
%"#320[0]" = alloca i1
%".mux_1_bs[0]" = alloca i1
%".mux_1_as[0]" = alloca i1
%"#316[0]" = alloca i1
%"#318[0]" = alloca i1
%"#317[0]" = alloca i1
%"#317[1]" = alloca i1
%".logic_0_OR[0]" = alloca i1
%"#312[0]" = alloca i1
%"#315[0]" = alloca i1
%"#314[0]" = alloca i1
%"#313[0]" = alloca i1
%"#313[1]" = alloca i1
%".logic_0_AND[0]" = alloca i1
%a_7 = alloca i1
%a_6 = alloca i1
%a_5 = alloca i1
%a_4 = alloca i1
%a_3 = alloca i1
%a_2 = alloca i1
%a_1 = alloca i1
%a_0 = alloca i1
%b_7 = alloca i1
%b_6 = alloca i1
%b_5 = alloca i1
%b_4 = alloca i1
%b_3 = alloca i1
%b_2 = alloca i1
%b_1 = alloca i1
%b_0 = alloca i1
%"#195[0]" = alloca i1
%".add_1_c[0]" = alloca i1
%"#181[0]" = alloca i1
%"#179[0]" = alloca i1
%"#178[0]" = alloca i1
```

```
%"#174[0]" = alloca i1
%"#172[0]" = alloca i1
%"#171[0]" = alloca i1
%"#167[0]" = alloca i1
%"#165[0]" = alloca i1
%"#164[0]" = alloca i1
%"#160[0]" = alloca i1
%"#158[0]" = alloca i1
%"#157[0]" = alloca i1
%"#153[0]" = alloca i1
%"#151[0]" = alloca i1
%"#150[0]" = alloca i1
%"#146[0]" = alloca i1
%"#144[0]" = alloca i1
%"#143[0]" = alloca i1
%"#139[0]" = alloca i1
%"#137[0]" = alloca i1
%"#136[0]" = alloca i1
%"#132[0]" = alloca i1
%"#130[0]" = alloca i1
%"#129[0]" = alloca i1
%"#126[0]" = alloca i1
%".comp_0_one[7]" = alloca i1
%"#110[0]" = alloca i1
%".comp_0_one[6]" = alloca i1
%"#94[0]" = alloca i1
%".comp_0_one[5]" = alloca i1
%"#78[0]" = alloca i1
%".comp_0_one[4]" = alloca i1
%"#62[0]" = alloca i1
%".comp_0_one[3]" = alloca i1
%"#46[0]" = alloca i1
%".comp_0_one[2]" = alloca i1
%"#30[0]" = alloca i1
%".comp_0_one[1]" = alloca i1
%"#14[0]" = alloca i1
%".comp_0_one[0]" = alloca i1
%"#11[0]" = alloca i1
%".add_0_c[0]" = alloca i1
%0 = alloca i1
store i1 false, i1* %".add_0_c[0]"
%".add_0_c[0]1" = load i1, i1* %".add_0_c[0]"
store i1 %".add_0_c[0]1", i1* %"#11[0]"
store i1 true, i1* %".comp_0_one[0]"
%".comp_0_one[0]2" = load i1, i1* %".comp_0_one[0]"
store i1 %".comp_0_one[0]2", i1* %"#14[0]"
store i1 false, i1* %".comp_0_one[1]"
%".comp_0_one[1]3" = load i1, i1* %".comp_0_one[1]"
store i1 %".comp_0_one[1]3", i1* %"#30[0]"
store i1 false, i1* %".comp_0_one[2]"
%".comp_0_one[2]4" = load i1, i1* %".comp_0_one[2]"
store i1 %".comp_0_one[2]4", i1* %"#46[0]"
store i1 false, i1* %".comp_0_one[3]"
%".comp_0_one[3]5" = load i1, i1* %".comp_0_one[3]"
store i1 %".comp_0_one[3]5", i1* %"#62[0]"
store i1 false, i1* %".comp_0_one[4]"
%".comp_0_one[4]6" = load i1, i1* %".comp_0_one[4]"
store i1 %".comp_0_one[4]6", i1* %"#78[0]"
store i1 false, i1* %".comp_0_one[5]"
%".comp_0_one[5]7" = load i1, i1* %".comp_0_one[5]"
store i1 %".comp_0_one[5]7", i1* %"#94[0]"
store i1 false, i1* %".comp_0_one[6]"
%".comp_0_one[6]8" = load i1, i1* %".comp_0_one[6]"
store i1 %".comp_0_one[6]8", i1* %"#110[0]"
store i1 false, i1* %".comp_0_one[7]"
%".comp_0_one[7]9" = load i1, i1* %".comp_0_one[7]"
store i1 %".comp_0_one[7]9", i1* %"#126[0]"
store i1 true, i1* %"#129[0]"
%"#129[0]10" = load i1, i1* %"#129[0]"
%"#130[0]'svalue" = xor i1 %"#129[0]10", true
store i1 %"#130[0]'svalue", i1* %"#130[0]"
store i1 true, i1* %"#132[0]"
store i1 true, i1* %"#136[0]"
%"#136[0]11" = load i1, i1* %"#136[0]"
%"#137[0]'svalue" = xor i1 %"#136[0]11", true
store i1 %"#137[0]'svalue", i1* %"#137[0]"
store i1 true, i1* %"#139[0]"
store i1 true, i1* %"#143[0]"
%"#143[0]12" = load i1, i1* %"#143[0]"
%"#144[0]'svalue" = xor i1 %"#143[0]12", true
store i1 %"#144[0]'svalue", i1* %"#144[0]"
store i1 true, i1* %"#146[0]"
store i1 true, i1* %"#150[0]"
%"#150[0]13" = load i1, i1* %"#150[0]"
%"#151[0]'svalue" = xor i1 %"#150[0]13", true
store i1 %"#151[0]'svalue", i1* %"#151[0]"
store i1 true, i1* %"#153[0]"
store i1 true, i1* %"#157[0]"
%"#157[0]14" = load i1, i1* %"#157[0]"
%"#158[0]'svalue" = xor i1 %"#157[0]14", true
store i1 %"#158[0]'svalue", i1* %"#158[0]"
store i1 true, i1* %"#160[0]"
store i1 true, i1* %"#164[0]"
%"#164[0]15" = load i1, i1* %"#164[0]"
%"#165[0]'svalue" = xor i1 %"#164[0]15", true
store i1 %"#165[0]'svalue", i1* %"#165[0]"
store i1 true, i1* %"#167[0]"
store i1 true, i1* %"#171[0]"
%"#171[0]16" = load i1, i1* %"#171[0]"
%"#172[0]'svalue" = xor i1 %"#171[0]16", true
store i1 %"#172[0]'svalue", i1* %"#172[0]"
store i1 true, i1* %"#174[0]"
store i1 true, i1* %"#178[0]"
%"#178[0]17" = load i1, i1* %"#178[0]"
%"#179[0]'svalue" = xor i1 %"#178[0]17", true
store i1 %"#179[0]'svalue", i1* %"#179[0]"
store i1 true, i1* %"#181[0]"
store i1 false, i1* %".add_1_c[0]"
%".add_1_c[0]18" = load i1, i1* %".add_1_c[0]"
store i1 %".add_1_c[0]18", i1* %"#195[0]"
store i1 false, i1* %b_0
store i1 false, i1* %b_1
store i1 true, i1* %b_2
store i1 false, i1* %b_3
store i1 false, i1* %b_4
store i1 false, i1* %b_5
store i1 true, i1* %b_6
store i1 true, i1* %b_7
store i1 true, i1* %a_0
store i1 true, i1* %a_1
store i1 true, i1* %a_2
store i1 true, i1* %a_3
store i1 true, i1* %a_4
store i1 true, i1* %a_5
store i1 false, i1* %a_6
store i1 false, i1* %a_7
%b_019 = load i1, i1* %b_0
%a_020 = load i1, i1* %a_0
%".logic_0_AND[0]'svalue" = and i1 %a_020, %b_019
```

```
store i1 %".logic_0_AND[0]'svalue", i1* %".logic_0_AND[0]"
store i1 true, i1* %"#313[1]"
store i1 true, i1* %"#313[0]"
%"#313[0]21" = load i1, i1* %"#313[0]"
store i1 %"#313[0]21", i1* %"#314[0]"
%"#314[0]22" = load i1, i1* %"#314[0]"
%"#315[0]'svalue" = xor i1 %"#314[0]22", true
store i1 %"#315[0]'svalue", i1* %"#315[0]"
%".logic_0_AND[0]23" = load i1, i1* %".logic_0_AND[0]"
store i1 %".logic_0_AND[0]23", i1* %"#312[0]"
%b_024 = load i1, i1* %b_0
%a_025 = load i1, i1* %a_0
%".logic_0_OR[0]'svalue" = or i1 %a_025, %b_024
store i1 %".logic_0_OR[0]'svalue", i1* %".logic_0_OR[0]"
store i1 true, i1* %"#317[1]"
store i1 true, i1* %"#317[0]"
%"#317[0]26" = load i1, i1* %"#317[0]"
store i1 %"#317[0]26", i1* %"#318[0]"
%".logic_0_OR[0]27" = load i1, i1* %".logic_0_OR[0]"
store i1 %".logic_0_OR[0]27", i1* %"#316[0]"
%"#315[0]28" = load i1, i1* %"#315[0]"
%"#312[0]29" = load i1, i1* %"#312[0]"
%".mux_1_as[0]'svalue" = and i1 %"#312[0]29", %"#315[0]28"
store i1 %".mux_1_as[0]'svalue", i1* %".mux_1_as[0]"
%"#318[0]30" = load i1, i1* %"#318[0]"
%"#316[0]31" = load i1, i1* %"#316[0]"
%".mux_1_bs[0]'svalue" = and i1 %"#316[0]31", %"#318[0]30"
store i1 %".mux_1_bs[0]'svalue", i1* %".mux_1_bs[0]"
%".mux_1_bs[0]32" = load i1, i1* %".mux_1_bs[0]"
store i1 %".mux_1_bs[0]32", i1* %"#320[0]"
%".mux_1_as[0]33" = load i1, i1* %".mux_1_as[0]"
store i1 %".mux_1_as[0]33", i1* %"#319[0]"
%b_134 = load i1, i1* %b_1
%a_135 = load i1, i1* %a_1
%".logic_0_AND[1]'svalue" = and i1 %a_135, %b_134
store i1 %".logic_0_AND[1]'svalue", i1* %".logic_0_AND[1]"
store i1 true, i1* %"#322[1]"
store i1 true, i1* %"#322[0]"
%"#322[0]36" = load i1, i1* %"#322[0]"
store i1 %"#322[0]36", i1* %"#323[0]"
%"#323[0]37" = load i1, i1* %"#323[0]"
%"#324[0]'svalue" = xor i1 %"#323[0]37", true
store i1 %"#324[0]'svalue", i1* %"#324[0]"
```

```
%".logic_0_AND[1]38" = load i1, i1* %".logic_0_AND[1]"
store i1 %".logic_0_AND[1]38", i1* %"#321[0]"
%b_139 = load i1, i1* %b_1
%a_140 = load i1, i1* %a_1
%".logic_0_OR[1]'svalue" = or i1 %a_140, %b_139
store i1 %".logic_0_OR[1]'svalue", i1* %".logic_0_OR[1]"
store i1 true, i1* %"#326[1]"
store i1 true, i1* %"#326[0]"
%"#326[0]41" = load i1, i1* %"#326[0]"
store i1 %"#326[0]41", i1* %"#327[0]"
%".logic_0_OR[1]42" = load i1, i1* %".logic_0_OR[1]"
store i1 %".logic_0_OR[1]42", i1* %"#325[0]"
%"#324[0]43" = load i1, i1* %"#324[0]"
%"#321[0]44" = load i1, i1* %"#321[0]"
%".mux_1_as[1]'svalue" = and i1 %"#321[0]44", %"#324[0]43"
store i1 %".mux_1_as[1]'svalue", i1* %".mux_1_as[1]"
%"#327[0]45" = load i1, i1* %"#327[0]"
%"#325[0]46" = load i1, i1* %"#325[0]"
%".mux_1_bs[1]'svalue" = and i1 %"#325[0]46", %"#327[0]45"
store i1 %".mux_1_bs[1]'svalue", i1* %".mux_1_bs[1]"
%".mux_1_bs[1]47" = load i1, i1* %".mux_1_bs[1]"
store i1 %".mux_1_bs[1]47", i1* %"#329[0]"
%".mux_1_as[1]48" = load i1, i1* %".mux_1_as[1]"
store i1 %".mux_1_as[1]48", i1* %"#328[0]"
%b_249 = load i1, i1* %b_2
%a_250 = load i1, i1* %a_2
%".logic_0_AND[2]'svalue" = and i1 %a_250, %b_249
store i1 %".logic_0_AND[2]'svalue", i1* %".logic_0_AND[2]"
store i1 true, i1* %"#331[1]"
store i1 true, i1* %"#331[0]"
%"#331[0]51" = load i1, i1* %"#331[0]"
store i1 %"#331[0]51", i1* %"#332[0]"
%"#332[0]52" = load i1, i1* %"#332[0]"
%"#333[0]'svalue" = xor i1 %"#332[0]52", true
store i1 %"#333[0]'svalue", i1* %"#333[0]"
%".logic_0_AND[2]53" = load i1, i1* %".logic_0_AND[2]"
store i1 %".logic_0_AND[2]53", i1* %"#330[0]"
%b_254 = load i1, i1* %b_2
%a_255 = load i1, i1* %a_2
%".logic_0_OR[2]'svalue" = or i1 %a_255, %b_254
store i1 %".logic_0_OR[2]'svalue", i1* %".logic_0_OR[2]"
store i1 true, i1* %"#335[1]"
store i1 true, i1* %"#335[0]"
```

```
%"#335[0]56" = load i1, i1* %"#335[0]"
store i1 %"#335[0]56", i1* %"#336[0]"
%".logic_0_OR[2]57" = load i1, i1* %".logic_0_OR[2]"
store i1 %".logic_0_OR[2]57", i1* %"#334[0]"
%"#333[0]58" = load i1, i1* %"#333[0]"
%"#330[0]59" = load i1, i1* %"#330[0]"
%".mux_1_as[2]'svalue" = and i1 %"#330[0]59", %"#333[0]58"
store i1 %".mux_1_as[2]'svalue", i1* %".mux_1_as[2]"
%"#336[0]60" = load i1, i1* %"#336[0]"
%"#334[0]61" = load i1, i1* %"#334[0]"
%".mux_1_bs[2]'svalue" = and i1 %"#334[0]61", %"#336[0]60"
store i1 %".mux_1_bs[2]'svalue", i1* %".mux_1_bs[2]"
%".mux_1_bs[2]62" = load i1, i1* %".mux_1_bs[2]"
store i1 %".mux_1_bs[2]62", i1* %"#338[0]"
%".mux_1_as[2]63" = load i1, i1* %".mux_1_as[2]"
store i1 %".mux_1_as[2]63", i1* %"#337[0]"
%b_364 = load i1, i1* %b_3
%a_365 = load i1, i1* %a_3
%".logic_0_AND[3]'svalue" = and i1 %a_365, %b_364
store i1 %".logic_0_AND[3]'svalue", i1* %".logic_0_AND[3]"
store i1 true, i1* %"#340[1]"
store i1 true, i1* %"#340[0]"
%"#340[0]66" = load i1, i1* %"#340[0]"
store i1 %"#340[0]66", i1* %"#341[0]"
%"#341[0]67" = load i1, i1* %"#341[0]"
%"#342[0]'svalue" = xor i1 %"#341[0]67", true
store i1 %"#342[0]'svalue", i1* %"#342[0]"
%".logic_0_AND[3]68" = load i1, i1* %".logic_0_AND[3]"
store i1 %".logic_0_AND[3]68", i1* %"#339[0]"
%b_369 = load i1, i1* %b_3
%a_370 = load i1, i1* %a_3
%".logic_0_OR[3]'svalue" = or i1 %a_370, %b_369
store i1 %".logic_0_OR[3]'svalue", i1* %".logic_0_OR[3]"
store i1 true, i1* %"#344[1]"
store i1 true, i1* %"#344[0]"
%"#344[0]71" = load i1, i1* %"#344[0]"
store i1 %"#344[0]71", i1* %"#345[0]"
%".logic_0_OR[3]72" = load i1, i1* %".logic_0_OR[3]"
store i1 %".logic_0_OR[3]72", i1* %"#343[0]"
%"#342[0]73" = load i1, i1* %"#342[0]"
%"#339[0]74" = load i1, i1* %"#339[0]"
%".mux_1_as[3]'svalue" = and i1 %"#339[0]74", %"#342[0]73"
store i1 %".mux_1_as[3]'svalue", i1* %".mux_1_as[3]"
```

```
%"#345[0]75" = load i1, i1* %"#345[0]"
%"#343[0]76" = load i1, i1* %"#343[0]"
%".mux_1_bs[3]'svalue" = and i1 %"#343[0]76", %"#345[0]75"
store i1 %".mux_1_bs[3]'svalue", i1* %".mux_1_bs[3]"
%".mux_1_bs[3]77" = load i1, i1* %".mux_1_bs[3]"
store i1 %".mux_1_bs[3]77", i1* %"#347[0]"
%".mux_1_as[3]78" = load i1, i1* %".mux_1_as[3]"
store i1 %".mux_1_as[3]78", i1* %"#346[0]"
%b_479 = load i1, i1* %b_4
%a_480 = load i1, i1* %a_4
%".logic_0_AND[4]'svalue" = and i1 %a_480, %b_479
store i1 %".logic_0_AND[4]'svalue", i1* %".logic_0_AND[4]"
store i1 true, i1* %"#349[1]"
store i1 true, i1* %"#349[0]"
%"#349[0]81" = load i1, i1* %"#349[0]"
store i1 %"#349[0]81", i1* %"#350[0]"
%"#350[0]82" = load i1, i1* %"#350[0]"
%"#351[0]'svalue" = xor i1 %"#350[0]82", true
store i1 %"#351[0]'svalue", i1* %"#351[0]"
%".logic_0_AND[4]83" = load i1, i1* %".logic_0_AND[4]"
store i1 %".logic_0_AND[4]83", i1* %"#348[0]"
%b_484 = load i1, i1* %b_4
%a_485 = load i1, i1* %a_4
%".logic_0_OR[4]'svalue" = or i1 %a_485, %b_484
store i1 %".logic_0_OR[4]'svalue", i1* %".logic_0_OR[4]"
store i1 true, i1* %"#353[1]"
store i1 true, i1* %"#353[0]"
%"#353[0]86" = load i1, i1* %"#353[0]"
store i1 %"#353[0]86", i1* %"#354[0]"
%".logic_0_OR[4]87" = load i1, i1* %".logic_0_OR[4]"
store i1 %".logic_0_OR[4]87", i1* %"#352[0]"
%"#351[0]88" = load i1, i1* %"#351[0]"
%"#348[0]89" = load i1, i1* %"#348[0]"
%".mux_1_as[4]'svalue" = and i1 %"#348[0]89", %"#351[0]88"
store i1 %".mux_1_as[4]'svalue", i1* %".mux_1_as[4]"
%"#354[0]90" = load i1, i1* %"#354[0]"
%"#352[0]91" = load i1, i1* %"#352[0]"
%".mux_1_bs[4]'svalue" = and i1 %"#352[0]91", %"#354[0]90"
store i1 %".mux_1_bs[4]'svalue", i1* %".mux_1_bs[4]"
%".mux_1_bs[4]92" = load i1, i1* %".mux_1_bs[4]"
store i1 %".mux_1_bs[4]92", i1* %"#356[0]"
%".mux_1_as[4]93" = load i1, i1* %".mux_1_as[4]"
store i1 %".mux_1_as[4]93", i1* %"#355[0]"
```

```
%b_594 = load i1, i1* %b_5
%a_595 = load i1, i1* %a_5
%".logic_0_AND[5]'svalue" = and i1 %a_595, %b_594
store i1 %".logic_0_AND[5]'svalue", i1* %".logic_0_AND[5]"
store i1 true, i1* %"#358[1]"
store i1 true, i1* %"#358[0]"
%"#358[0]96" = load i1, i1* %"#358[0]"
store i1 %"#358[0]96", i1* %"#359[0]"
%"#359[0]97" = load i1, i1* %"#359[0]"
%"#360[0]'svalue" = xor i1 %"#359[0]97", true
store i1 %"#360[0]'svalue", i1* %"#360[0]"
%".logic_0_AND[5]98" = load i1, i1* %".logic_0_AND[5]"
store i1 %".logic_0_AND[5]98", i1* %"#357[0]"
%b_599 = load i1, i1* %b_5
%a_5100 = load i1, i1* %a_5
%".logic_0_OR[5]'svalue" = or i1 %a_5100, %b_599
store i1 %".logic_0_OR[5]'svalue", i1* %".logic_0_OR[5]"
store i1 true, i1* %"#362[1]"
store i1 true, i1* %"#362[0]"
%"#362[0]101" = load i1, i1* %"#362[0]"
store i1 %"#362[0]101", i1* %"#363[0]"
%".logic_0_OR[5]102" = load i1, i1* %".logic_0_OR[5]"
store i1 %".logic_0_OR[5]102", i1* %"#361[0]"
%"#360[0]103" = load i1, i1* %"#360[0]"
%"#357[0]104" = load i1, i1* %"#357[0]"
%".mux_1_as[5]'svalue" = and i1 %"#357[0]104", %"#360[0]103"
store i1 %".mux_1_as[5]'svalue", i1* %".mux_1_as[5]"
%"#363[0]105" = load i1, i1* %"#363[0]"
%"#361[0]106" = load i1, i1* %"#361[0]"
%".mux_1_bs[5]'svalue" = and i1 %"#361[0]106", %"#363[0]105"
store i1 %".mux_1_bs[5]'svalue", i1* %".mux_1_bs[5]"
%".mux_1_bs[5]107" = load i1, i1* %".mux_1_bs[5]"
store i1 %".mux_1_bs[5]107", i1* %"#365[0]"
%".mux_1_as[5]108" = load i1, i1* %".mux_1_as[5]"
store i1 %".mux_1_as[5]108", i1* %"#364[0]"
%b_6109 = load i1, i1* %b_6
%a_6110 = load i1, i1* %a_6
%".logic_0_AND[6]'svalue" = and i1 %a_6110, %b_6109
store i1 %".logic_0_AND[6]'svalue", i1* %".logic_0_AND[6]"
store i1 true, i1* %"#367[1]"
store i1 true, i1* %"#367[0]"
%"#367[0]111" = load i1, i1* %"#367[0]"
store i1 %"#367[0]111", i1* %"#368[0]"
```

```
%"#368[0]112" = load i1, i1* %"#368[0]"
%"#369[0]'svalue" = xor i1 %"#368[0]112", true
store i1 %"#369[0]'svalue", i1* %"#369[0]"
%".logic_0_AND[6]113" = load i1, i1* %".logic_0_AND[6]"
store i1 %".logic_0_AND[6]113", i1* %"#366[0]"
%b_6114 = load i1, i1* %b_6
%a_6115 = load i1, i1* %a_6
%".logic_0_OR[6]'svalue" = or i1 %a_6115, %b_6114
store i1 %".logic_0_OR[6]'svalue", i1* %".logic_0_OR[6]"
store i1 true, i1* %"#371[1]"
store i1 true, i1* %"#371[0]"
%"#371[0]116" = load i1, i1* %"#371[0]"
store i1 %"#371[0]116", i1* %"#372[0]"
%".logic_0_OR[6]117" = load i1, i1* %".logic_0_OR[6]"
store i1 %".logic_0_OR[6]117", i1* %"#370[0]"
%"#369[0]118" = load i1, i1* %"#369[0]"
%"#366[0]119" = load i1, i1* %"#366[0]"
%".mux_1_as[6]'svalue" = and i1 %"#366[0]119", %"#369[0]118"
store i1 %".mux_1_as[6]'svalue", i1* %".mux_1_as[6]"
%"#372[0]120" = load i1, i1* %"#372[0]"
%"#370[0]121" = load i1, i1* %"#370[0]"
%".mux_1_bs[6]'svalue" = and i1 %"#370[0]121", %"#372[0]120"
store i1 %".mux_1_bs[6]'svalue", i1* %".mux_1_bs[6]"
%".mux_1_bs[6]122" = load i1, i1* %".mux_1_bs[6]"
store i1 %".mux_1_bs[6]122", i1* %"#374[0]"
%".mux_1_as[6]123" = load i1, i1* %".mux_1_as[6]"
store i1 %".mux_1_as[6]123", i1* %"#373[0]"
%b_7124 = load i1, i1* %b_7
%a_7125 = load i1, i1* %a_7
%".logic_0_AND[7]'svalue" = and i1 %a_7125, %b_7124
store i1 %".logic_0_AND[7]'svalue", i1* %".logic_0_AND[7]"
store i1 true, i1* %"#376[1]"
store i1 true, i1* %"#376[0]"
%"#376[0]126" = load i1, i1* %"#376[0]"
store i1 %"#376[0]126", i1* %"#377[0]"
%"#377[0]127" = load i1, i1* %"#377[0]"
%"#378[0]'svalue" = xor i1 %"#377[0]127", true
store i1 %"#378[0]'svalue", i1* %"#378[0]"
%".logic_0_AND[7]128" = load i1, i1* %".logic_0_AND[7]"
store i1 %".logic_0_AND[7]128", i1* %"#375[0]"
%b_7129 = load i1, i1* %b_7
%a_7130 = load i1, i1* %a_7
%".logic_0_OR[7]'svalue" = or i1 %a_7130, %b_7129
```

```
store i1 %".logic_0_OR[7]'svalue", i1* %".logic_0_OR[7]"
store i1 true, i1* %"#380[1]"
store i1 true, i1* %"#380[0]"
%"#380[0]131" = load i1, i1* %"#380[0]"
store i1 %"#380[0]131", i1* %"#381[0]"
%".logic_0_OR[7]132" = load i1, i1* %".logic_0_OR[7]"
store i1 %".logic_0_OR[7]132", i1* %"#379[0]"
%"#378[0]133" = load i1, i1* %"#378[0]"
%"#375[0]134" = load i1, i1* %"#375[0]"
%".mux_1_as[7]'svalue" = and i1 %"#375[0]134", %"#378[0]133"
store i1 %".mux_1_as[7]'svalue", i1* %".mux_1_as[7]"
%"#381[0]135" = load i1, i1* %"#381[0]"
%"#379[0]136" = load i1, i1* %"#379[0]"
%".mux_1_bs[7]'svalue" = and i1 %"#379[0]136", %"#381[0]135"
store i1 %".mux_1_bs[7]'svalue", i1* %".mux_1_bs[7]"
%".mux_1_bs[7]137" = load i1, i1* %".mux_1_bs[7]"
store i1 %".mux_1_bs[7]137", i1* %"#383[0]"
%".mux_1_as[7]138" = load i1, i1* %".mux_1_as[7]"
store i1 %".mux_1_as[7]138", i1* %"#382[0]"
%"#320[0]139" = load i1, i1* %"#320[0]"
%"#319[0]140" = load i1, i1* %"#319[0]"
%".mux_1_o[0]'svalue" = or i1 %"#319[0]140", %"#320[0]139"
store i1 %".mux_1_o[0]'svalue", i1* %".mux_1_o[0]"
%"#329[0]141" = load i1, i1* %"#329[0]"
%"#328[0]142" = load i1, i1* %"#328[0]"
%".mux_1_o[1]'svalue" = or i1 %"#328[0]142", %"#329[0]141"
store i1 %".mux_1_o[1]'svalue", i1* %".mux_1_o[1]"
%"#338[0]143" = load i1, i1* %"#338[0]"
%"#337[0]144" = load i1, i1* %"#337[0]"
%".mux_1_o[2]'svalue" = or i1 %"#337[0]144", %"#338[0]143"
store i1 %".mux_1_o[2]'svalue", i1* %".mux_1_o[2]"
%"#347[0]145" = load i1, i1* %"#347[0]"
%"#346[0]146" = load i1, i1* %"#346[0]"
%".mux_1_o[3]'svalue" = or i1 %"#346[0]146", %"#347[0]145"
store i1 %".mux_1_o[3]'svalue", i1* %".mux_1_o[3]"
%"#356[0]147" = load i1, i1* %"#356[0]"
%"#355[0]148" = load i1, i1* %"#355[0]"
%".mux_1_o[4]'svalue" = or i1 %"#355[0]148", %"#356[0]147"
store i1 %".mux_1_o[4]'svalue", i1* %".mux_1_o[4]"
%"#365[0]149" = load i1, i1* %"#365[0]"
%"#364[0]150" = load i1, i1* %"#364[0]"
%".mux_1_o[5]'svalue" = or i1 %"#364[0]150", %"#365[0]149"
store i1 %".mux_1_o[5]'svalue", i1* %".mux_1_o[5]"
```

```
%"#374[0]151" = load i1, i1* %"#374[0]"
%"#373[0]152" = load i1, i1* %"#373[0]"
%".mux_1_o[6]'svalue" = or i1 %"#373[0]152", %"#374[0]151"
store i1 %".mux_1_o[6]'svalue", i1* %".mux_1_o[6]"
%"#383[0]153" = load i1, i1* %"#383[0]"
%"#382[0]154" = load i1, i1* %"#382[0]"
%".mux_1_o[7]'svalue" = or i1 %"#382[0]154", %"#383[0]153"
store i1 %".mux_1_o[7]'svalue", i1* %".mux_1_o[7]"
%b_0155 = load i1, i1* %b_0
%a_0156 = load i1, i1* %a_0
%".logic_0_XOR[0]'svalue" = xor i1 %a_0156, %b_0155
store i1 %".logic_0_XOR[0]'svalue", i1* %".logic_0_XOR[0]"
store i1 true, i1* %"#385[1]"
store i1 true, i1* %"#385[0]"
%"#385[0]157" = load i1, i1* %"#385[0]"
store i1 %"#385[0]157", i1* %"#386[0]"
%"#386[0]158" = load i1, i1* %"#386[0]"
%"#387[0]'svalue" = xor i1 %"#386[0]158", true
store i1 %"#387[0]'svalue", i1* %"#387[0]"
%".logic_0_XOR[0]159" = load i1, i1* %".logic_0_XOR[0]"
store i1 %".logic_0_XOR[0]159", i1* %"#384[0]"
%a_0160 = load i1, i1* %a_0
%".logic_0_NOT[0]'svalue" = xor i1 %a_0160, true
store i1 %".logic_0_NOT[0]'svalue", i1* %".logic_0_NOT[0]"
store i1 true, i1* %"#389[1]"
store i1 true, i1* %"#389[0]"
%"#389[0]161" = load i1, i1* %"#389[0]"
store i1 %"#389[0]161", i1* %"#390[0]"
%".logic_0_NOT[0]162" = load i1, i1* %".logic_0_NOT[0]"
store i1 %".logic_0_NOT[0]162", i1* %"#388[0]"
%"#387[0]163" = load i1, i1* %"#387[0]"
%"#384[0]164" = load i1, i1* %"#384[0]"
%".mux_2_as[0]'svalue" = and i1 %"#384[0]164", %"#387[0]163"
store i1 %".mux_2_as[0]'svalue", i1* %".mux_2_as[0]"
%"#390[0]165" = load i1, i1* %"#390[0]"
%"#388[0]166" = load i1, i1* %"#388[0]"
%".mux_2_bs[0]'svalue" = and i1 %"#388[0]166", %"#390[0]165"
store i1 %".mux_2_bs[0]'svalue", i1* %".mux_2_bs[0]"
%".mux_2_bs[0]167" = load i1, i1* %".mux_2_bs[0]"
store i1 %".mux_2_bs[0]167", i1* %"#392[0]"
%".mux_2_as[0]168" = load i1, i1* %".mux_2_as[0]"
store i1 %".mux_2_as[0]168", i1* %"#391[0]"
%b_1169 = load i1, i1* %b_1
```

```
%a_1170 = load i1, i1* %a_1
%".logic_0_XOR[1]'svalue" = xor i1 %a_1170, %b_1169
store i1 %".logic_0_XOR[1]'svalue", i1* %".logic_0_XOR[1]"
store i1 true, i1* %"#394[1]"
store i1 true, i1* %"#394[0]"
%"#394[0]171" = load i1, i1* %"#394[0]"
store i1 %"#394[0]171", i1* %"#395[0]"
%"#395[0]172" = load i1, i1* %"#395[0]"
%"#396[0]'svalue" = xor i1 %"#395[0]172", true
store i1 %"#396[0]'svalue", i1* %"#396[0]"
%".logic_0_XOR[1]173" = load i1, i1* %".logic_0_XOR[1]"
store i1 %".logic_0_XOR[1]173", i1* %"#393[0]"
%a_1174 = load i1, i1* %a_1
%".logic_0_NOT[1]'svalue" = xor i1 %a_1174, true
store i1 %".logic_0_NOT[1]'svalue", i1* %".logic_0_NOT[1]"
store i1 true, i1* %"#398[1]"
store i1 true, i1* %"#398[0]"
%"#398[0]175" = load i1, i1* %"#398[0]"
store i1 %"#398[0]175", i1* %"#399[0]"
%".logic_0_NOT[1]176" = load i1, i1* %".logic_0_NOT[1]"
store i1 %".logic_0_NOT[1]176", i1* %"#397[0]"
%"#396[0]177" = load i1, i1* %"#396[0]"
%"#393[0]178" = load i1, i1* %"#393[0]"
%".mux_2_as[1]'svalue" = and i1 %"#393[0]178", %"#396[0]177"
store i1 %".mux_2_as[1]'svalue", i1* %".mux_2_as[1]"
%"#399[0]179" = load i1, i1* %"#399[0]"
%"#397[0]180" = load i1, i1* %"#397[0]"
%".mux_2_bs[1]'svalue" = and i1 %"#397[0]180", %"#399[0]179"
store i1 %".mux_2_bs[1]'svalue", i1* %".mux_2_bs[1]"
%".mux_2_bs[1]181" = load i1, i1* %".mux_2_bs[1]"
store i1 %".mux_2_bs[1]181", i1* %"#401[0]"
%".mux_2_as[1]182" = load i1, i1* %".mux_2_as[1]"
store i1 %".mux_2_as[1]182", i1* %"#400[0]"
%b_2183 = load i1, i1* %b_2
%a_2184 = load i1, i1* %a_2
%".logic_0_XOR[2]'svalue" = xor i1 %a_2184, %b_2183
store i1 %".logic_0_XOR[2]'svalue", i1* %".logic_0_XOR[2]"
store i1 true, i1* %"#403[1]"
store i1 true, i1* %"#403[0]"
%"#403[0]185" = load i1, i1* %"#403[0]"
store i1 %"#403[0]185", i1* %"#404[0]"
%"#404[0]186" = load i1, i1* %"#404[0]"
%"#405[0]'svalue" = xor i1 %"#404[0]186", true
```

```
store i1 %"#405[0]'svalue", i1* %"#405[0]"
%".logic_0_XOR[2]187" = load i1, i1* %".logic_0_XOR[2]"
store i1 %".logic_0_XOR[2]187", i1* %"#402[0]"
%a_2188 = load i1, i1* %a_2
%".logic_0_NOT[2]'svalue" = xor i1 %a_2188, true
store i1 %".logic_0_NOT[2]'svalue", i1* %".logic_0_NOT[2]"
store i1 true, i1* %"#407[1]"
store i1 true, i1* %"#407[0]"
%"#407[0]189" = load i1, i1* %"#407[0]"
store i1 %"#407[0]189", i1* %"#408[0]"
%".logic_0_NOT[2]190" = load i1, i1* %".logic_0_NOT[2]"
store i1 %".logic_0_NOT[2]190", i1* %"#406[0]"
%"#405[0]191" = load i1, i1* %"#405[0]"
%"#402[0]192" = load i1, i1* %"#402[0]"
%".mux_2_as[2]'svalue" = and i1 %"#402[0]192", %"#405[0]191"
store i1 %".mux_2_as[2]'svalue", i1* %".mux_2_as[2]"
%"#408[0]193" = load i1, i1* %"#408[0]"
%"#406[0]194" = load i1, i1* %"#406[0]"
%".mux_2_bs[2]'svalue" = and i1 %"#406[0]194", %"#408[0]193"
store i1 %".mux_2_bs[2]'svalue", i1* %".mux_2_bs[2]"
%".mux_2_bs[2]195" = load i1, i1* %".mux_2_bs[2]"
store i1 %".mux_2_bs[2]195", i1* %"#410[0]"
%".mux_2_as[2]196" = load i1, i1* %".mux_2_as[2]"
store i1 %".mux_2_as[2]196", i1* %"#409[0]"
%b_3197 = load i1, i1* %b_3
%a_3198 = load i1, i1* %a_3
%".logic_0_XOR[3]'svalue" = xor i1 %a_3198, %b_3197
store i1 %".logic_0_XOR[3]'svalue", i1* %".logic_0_XOR[3]"
store i1 true, i1* %"#412[1]"
store i1 true, i1* %"#412[0]"
%"#412[0]199" = load i1, i1* %"#412[0]"
store i1 %"#412[0]199", i1* %"#413[0]"
%"#413[0]200" = load i1, i1* %"#413[0]"
%"#414[0]'svalue" = xor i1 %"#413[0]200", true
store i1 %"#414[0]'svalue", i1* %"#414[0]"
%".logic_0_XOR[3]201" = load i1, i1* %".logic_0_XOR[3]"
store i1 %".logic_0_XOR[3]201", i1* %"#411[0]"
%a_3202 = load i1, i1* %a_3
%".logic_0_NOT[3]'svalue" = xor i1 %a_3202, true
store i1 %".logic_0_NOT[3]'svalue", i1* %".logic_0_NOT[3]"
store i1 true, i1* %"#416[1]"
store i1 true, i1* %"#416[0]"
%"#416[0]203" = load i1, i1* %"#416[0]"
```

```
store i1 %"#416[0]203", i1* %"#417[0]"
%".logic_0_NOT[3]204" = load i1, i1* %".logic_0_NOT[3]"
store i1 %".logic_0_NOT[3]204", i1* %"#415[0]"
%"#414[0]205" = load i1, i1* %"#414[0]"
%"#411[0]206" = load i1, i1* %"#411[0]"
%".mux_2_as[3]'svalue" = and i1 %"#411[0]206", %"#414[0]205"
store i1 %".mux_2_as[3]'svalue", i1* %".mux_2_as[3]"
%"#417[0]207" = load i1, i1* %"#417[0]"
%"#415[0]208" = load i1, i1* %"#415[0]"
%".mux_2_bs[3]'svalue" = and i1 %"#415[0]208", %"#417[0]207"
store i1 %".mux_2_bs[3]'svalue", i1* %".mux_2_bs[3]"
%".mux_2_bs[3]209" = load i1, i1* %".mux_2_bs[3]"
store i1 %".mux_2_bs[3]209", i1* %"#419[0]"
%".mux_2_as[3]210" = load i1, i1* %".mux_2_as[3]"
store i1 %".mux_2_as[3]210", i1* %"#418[0]"
%b_4211 = load i1, i1* %b_4
%a_4212 = load i1, i1* %a_4
%".logic_0_XOR[4]'svalue" = xor i1 %a_4212, %b_4211
store i1 %".logic_0_XOR[4]'svalue", i1* %".logic_0_XOR[4]"
store i1 true, i1* %"#421[1]"
store i1 true, i1* %"#421[0]"
%"#421[0]213" = load i1, i1* %"#421[0]"
store i1 %"#421[0]213", i1* %"#422[0]"
%"#422[0]214" = load i1, i1* %"#422[0]"
%"#423[0]'svalue" = xor i1 %"#422[0]214", true
store i1 %"#423[0]'svalue", i1* %"#423[0]"
%".logic_0_XOR[4]215" = load i1, i1* %".logic_0_XOR[4]"
store i1 %".logic_0_XOR[4]215", i1* %"#420[0]"
%a_4216 = load i1, i1* %a_4
%".logic_0_NOT[4]'svalue" = xor i1 %a_4216, true
store i1 %".logic_0_NOT[4]'svalue", i1* %".logic_0_NOT[4]"
store i1 true, i1* %"#425[1]"
store i1 true, i1* %"#425[0]"
%"#425[0]217" = load i1, i1* %"#425[0]"
store i1 %"#425[0]217", i1* %"#426[0]"
%".logic_0_NOT[4]218" = load i1, i1* %".logic_0_NOT[4]"
store i1 %".logic_0_NOT[4]218", i1* %"#424[0]"
%"#423[0]219" = load i1, i1* %"#423[0]"
%"#420[0]220" = load i1, i1* %"#420[0]"
%".mux_2_as[4]'svalue" = and i1 %"#420[0]220", %"#423[0]219"
store i1 %".mux_2_as[4]'svalue", i1* %".mux_2_as[4]"
%"#426[0]221" = load i1, i1* %"#426[0]"
%"#424[0]222" = load i1, i1* %"#424[0]"
```

```
%".mux_2_bs[4]'svalue" = and i1 %"#424[0]222", %"#426[0]221"
store i1 %".mux_2_bs[4]'svalue", i1* %".mux_2_bs[4]"
%".mux_2_bs[4]223" = load i1, i1* %".mux_2_bs[4]"
store i1 %".mux_2_bs[4]223", i1* %"#428[0]"
%".mux_2_as[4]224" = load i1, i1* %".mux_2_as[4]"
store i1 %".mux_2_as[4]224", i1* %"#427[0]"
%b_5225 = load i1, i1* %b_5
%a_5226 = load i1, i1* %a_5
%".logic_0_XOR[5]'svalue" = xor i1 %a_5226, %b_5225
store i1 %".logic_0_XOR[5]'svalue", i1* %".logic_0_XOR[5]"
store i1 true, i1* %"#430[1]"
store i1 true, i1* %"#430[0]"
%"#430[0]227" = load i1, i1* %"#430[0]"
store i1 %"#430[0]227", i1* %"#431[0]"
%"#431[0]228" = load i1, i1* %"#431[0]"
%"#432[0]'svalue" = xor i1 %"#431[0]228", true
store i1 %"#432[0]'svalue", i1* %"#432[0]"
%".logic_0_XOR[5]229" = load i1, i1* %".logic_0_XOR[5]"
store i1 %".logic_0_XOR[5]229", i1* %"#429[0]"
%a_5230 = load i1, i1* %a_5
%".logic_0_NOT[5]'svalue" = xor i1 %a_5230, true
store i1 %".logic_0_NOT[5]'svalue", i1* %".logic_0_NOT[5]"
store i1 true, i1* %"#434[1]"
store i1 true, i1* %"#434[0]"
%"#434[0]231" = load i1, i1* %"#434[0]"
store i1 %"#434[0]231", i1* %"#435[0]"
%".logic_0_NOT[5]232" = load i1, i1* %".logic_0_NOT[5]"
store i1 %".logic_0_NOT[5]232", i1* %"#433[0]"
%"#432[0]233" = load i1, i1* %"#432[0]"
%"#429[0]234" = load i1, i1* %"#429[0]"
%".mux_2_as[5]'svalue" = and i1 %"#429[0]234", %"#432[0]233"
store i1 %".mux_2_as[5]'svalue", i1* %".mux_2_as[5]"
%"#435[0]235" = load i1, i1* %"#435[0]"
%"#433[0]236" = load i1, i1* %"#433[0]"
%".mux_2_bs[5]'svalue" = and i1 %"#433[0]236", %"#435[0]235"
store i1 %".mux_2_bs[5]'svalue", i1* %".mux_2_bs[5]"
%".mux_2_bs[5]237" = load i1, i1* %".mux_2_bs[5]"
store i1 %".mux_2_bs[5]237", i1* %"#437[0]"
%".mux_2_as[5]238" = load i1, i1* %".mux_2_as[5]"
store i1 %".mux_2_as[5]238", i1* %"#436[0]"
%b_6239 = load i1, i1* %b_6
%a_6240 = load i1, i1* %a_6
%".logic_0_XOR[6]'svalue" = xor i1 %a_6240, %b_6239
```

```
store i1 %".logic_0_XOR[6]'svalue", i1* %".logic_0_XOR[6]"
store i1 true, i1* %"#439[1]"
store i1 true, i1* %"#439[0]"
%"#439[0]241" = load i1, i1* %"#439[0]"
store i1 %"#439[0]241", i1* %"#440[0]"
%"#440[0]242" = load i1, i1* %"#440[0]"
%"#441[0]'svalue" = xor i1 %"#440[0]242", true
store i1 %"#441[0]'svalue", i1* %"#441[0]"
%".logic_0_XOR[6]243" = load i1, i1* %".logic_0_XOR[6]"
store i1 %".logic_0_XOR[6]243", i1* %"#438[0]"
%a_6244 = load i1, i1* %a_6
%".logic_0_NOT[6]'svalue" = xor i1 %a_6244, true
store i1 %".logic_0_NOT[6]'svalue", i1* %".logic_0_NOT[6]"
store i1 true, i1* %"#443[1]"
store i1 true, i1* %"#443[0]"
%"#443[0]245" = load i1, i1* %"#443[0]"
store i1 %"#443[0]245", i1* %"#444[0]"
%".logic_0_NOT[6]246" = load i1, i1* %".logic_0_NOT[6]"
store i1 %".logic_0_NOT[6]246", i1* %"#442[0]"
%"#441[0]247" = load i1, i1* %"#441[0]"
%"#438[0]248" = load i1, i1* %"#438[0]"
%".mux_2_as[6]'svalue" = and i1 %"#438[0]248", %"#441[0]247"
store i1 %".mux_2_as[6]'svalue", i1* %".mux_2_as[6]"
%"#444[0]249" = load i1, i1* %"#444[0]"
%"#442[0]250" = load i1, i1* %"#442[0]"
%".mux_2_bs[6]'svalue" = and i1 %"#442[0]250", %"#444[0]249"
store i1 %".mux_2_bs[6]'svalue", i1* %".mux_2_bs[6]"
%".mux_2_bs[6]251" = load i1, i1* %".mux_2_bs[6]"
store i1 %".mux_2_bs[6]251", i1* %"#446[0]"
%".mux_2_as[6]252" = load i1, i1* %".mux_2_as[6]"
store i1 %".mux_2_as[6]252", i1* %"#445[0]"
%b_7253 = load i1, i1* %b_7
%a_7254 = load i1, i1* %a_7
%".logic_0_XOR[7]'svalue" = xor i1 %a_7254, %b_7253
store i1 %".logic_0_XOR[7]'svalue", i1* %".logic_0_XOR[7]"
store i1 true, i1* %"#448[1]"
store i1 true, i1* %"#448[0]"
%"#448[0]255" = load i1, i1* %"#448[0]"
store i1 %"#448[0]255", i1* %"#449[0]"
%"#449[0]256" = load i1, i1* %"#449[0]"
%"#450[0]'svalue" = xor i1 %"#449[0]256", true
store i1 %"#450[0]'svalue", i1* %"#450[0]"
%".logic_0_XOR[7]257" = load i1, i1* %".logic_0_XOR[7]"
```

```
store i1 %".logic_0_XOR[7]257", i1* %"#447[0]"
%a_7258 = load i1, i1* %a_7
%".logic_0_NOT[7]'svalue" = xor i1 %a_7258, true
store i1 %".logic_0_NOT[7]'svalue", i1* %".logic_0_NOT[7]"
store i1 true, i1* %"#452[1]"
store i1 true, i1* %"#452[0]"
%"#452[0]259" = load i1, i1* %"#452[0]"
store i1 %"#452[0]259", i1* %"#453[0]"
%".logic_0_NOT[7]260" = load i1, i1* %".logic_0_NOT[7]"
store i1 %".logic_0_NOT[7]260", i1* %"#451[0]"
%"#450[0]261" = load i1, i1* %"#450[0]"
%"#447[0]262" = load i1, i1* %"#447[0]"
%".mux_2_as[7]'svalue" = and i1 %"#447[0]262", %"#450[0]261"
store i1 %".mux_2_as[7]'svalue", i1* %".mux_2_as[7]"
%"#453[0]263" = load i1, i1* %"#453[0]"
%"#451[0]264" = load i1, i1* %"#451[0]"
%".mux_2_bs[7]'svalue" = and i1 %"#451[0]264", %"#453[0]263"
store i1 %".mux_2_bs[7]'svalue", i1* %".mux_2_bs[7]"
%".mux_2_bs[7]265" = load i1, i1* %".mux_2_bs[7]"
store i1 %".mux_2_bs[7]265", i1* %"#455[0]"
%".mux_2_as[7]266" = load i1, i1* %".mux_2_as[7]"
store i1 %".mux_2_as[7]266", i1* %"#454[0]"
%"#392[0]267" = load i1, i1* %"#392[0]"
%"#391[0]268" = load i1, i1* %"#391[0]"
%".mux_2_o[0]'svalue" = or i1 %"#391[0]268", %"#392[0]267"
store i1 %".mux_2_o[0]'svalue", i1* %".mux_2_o[0]"
%"#401[0]269" = load i1, i1* %"#401[0]"
%"#400[0]270" = load i1, i1* %"#400[0]"
%".mux_2_o[1]'svalue" = or i1 %"#400[0]270", %"#401[0]269"
store i1 %".mux_2_o[1]'svalue", i1* %".mux_2_o[1]"
%"#410[0]271" = load i1, i1* %"#410[0]"
%"#409[0]272" = load i1, i1* %"#409[0]"
%".mux_2_o[2]'svalue" = or i1 %"#409[0]272", %"#410[0]271"
store i1 %".mux_2_o[2]'svalue", i1* %".mux_2_o[2]"
%"#419[0]273" = load i1, i1* %"#419[0]"
%"#418[0]274" = load i1, i1* %"#418[0]"
%".mux_2_o[3]'svalue" = or i1 %"#418[0]274", %"#419[0]273"
store i1 %".mux_2_o[3]'svalue", i1* %".mux_2_o[3]"
%"#428[0]275" = load i1, i1* %"#428[0]"
%"#427[0]276" = load i1, i1* %"#427[0]"
%".mux_2_o[4]'svalue" = or i1 %"#427[0]276", %"#428[0]275"
store i1 %".mux_2_o[4]'svalue", i1* %".mux_2_o[4]"
%"#437[0]277" = load i1, i1* %"#437[0]"
```

```
%"#436[0]278" = load i1, i1* %"#436[0]"
%".mux_2_o[5]'svalue" = or i1 %"#436[0]278", %"#437[0]277"
store i1 %".mux_2_o[5]'svalue", i1* %".mux_2_o[5]"
%"#446[0]279" = load i1, i1* %"#446[0]"
%"#445[0]280" = load i1, i1* %"#445[0]"
%".mux_2_o[6]'svalue" = or i1 %"#445[0]280", %"#446[0]279"
store i1 %".mux_2_o[6]'svalue", i1* %".mux_2_o[6]"
%"#455[0]281" = load i1, i1* %"#455[0]"
%"#454[0]282" = load i1, i1* %"#454[0]"
%".mux_2_o[7]'svalue" = or i1 %"#454[0]282", %"#455[0]281"
store i1 %".mux_2_o[7]'svalue", i1* %".mux_2_o[7]"
%".mux_1_o[0]283" = load i1, i1* %".mux_1_o[0]"
store i1 %".mux_1_o[0]283", i1* %".mux2_0_m1[0]"
store i1 true, i1* %"#457[0]"
store i1 true, i1* %"#457[1]"
%"#457[1]284" = load i1, i1* %"#457[1]"
store i1 %"#457[1]284", i1* %"#458[0]"
%"#458[0]285" = load i1, i1* %"#458[0]"
%"#459[0]'svalue" = xor i1 %"#458[0]285", true
store i1 %"#459[0]'svalue", i1* %"#459[0]"
%".mux2_0_m1[0]286" = load i1, i1* %".mux2_0_m1[0]"
store i1 %".mux2_0_m1[0]286", i1* %"#456[0]"
%".mux_2_o[0]287" = load i1, i1* %".mux_2_o[0]"
store i1 %".mux_2_o[0]287", i1* %".mux2_0_m2[0]"
store i1 true, i1* %"#461[0]"
store i1 true, i1* %"#461[1]"
%"#461[1]288" = load i1, i1* %"#461[1]"
store i1 %"#461[1]288", i1* %"#462[0]"
%".mux2_0_m2[0]289" = load i1, i1* %".mux2_0_m2[0]"
store i1 %".mux2_0_m2[0]289", i1* %"#460[0]"
%"#459[0]290" = load i1, i1* %"#459[0]"
%"#456[0]291" = load i1, i1* %"#456[0]"
%".mux_3_as[0]'svalue" = and i1 %"#456[0]291", %"#459[0]290"
store i1 %".mux_3_as[0]'svalue", i1* %".mux_3_as[0]"
%"#462[0]292" = load i1, i1* %"#462[0]"
%"#460[0]293" = load i1, i1* %"#460[0]"
%".mux_3_bs[0]'svalue" = and i1 %"#460[0]293", %"#462[0]292"
store i1 %".mux_3_bs[0]'svalue", i1* %".mux_3_bs[0]"
%".mux_3_bs[0]294" = load i1, i1* %".mux_3_bs[0]"
store i1 %".mux_3_bs[0]294", i1* %"#464[0]"
%".mux_3_as[0]295" = load i1, i1* %".mux_3_as[0]"
store i1 %".mux_3_as[0]295", i1* %"#463[0]"
%".mux_1_o[1]296" = load i1, i1* %".mux_1_o[1]"
```

```
store i1 %".mux_1_o[1]296", i1* %".mux2_0_m1[1]"
store i1 true, i1* %"#466[0]"
store i1 true, i1* %"#466[1]"
%"#466[1]297" = load i1, i1* %"#466[1]"
store i1 %"#466[1]297", i1* %"#467[0]"
%"#467[0]298" = load i1, i1* %"#467[0]"
%"#468[0]'svalue" = xor i1 %"#467[0]298", true
store i1 %"#468[0]'svalue", i1* %"#468[0]"
%".mux2_0_m1[1]299" = load i1, i1* %".mux2_0_m1[1]"
store i1 %".mux2_0_m1[1]299", i1* %"#465[0]"
%".mux_2_o[1]300" = load i1, i1* %".mux_2_o[1]"
store i1 %".mux_2_o[1]300", i1* %".mux2_0_m2[1]"
store i1 true, i1* %"#470[0]"
store i1 true, i1* %"#470[1]"
%"#470[1]301" = load i1, i1* %"#470[1]"
store i1 %"#470[1]301", i1* %"#471[0]"
%".mux2_0_m2[1]302" = load i1, i1* %".mux2_0_m2[1]"
store i1 %".mux2_0_m2[1]302", i1* %"#469[0]"
%"#468[0]303" = load i1, i1* %"#468[0]"
%"#465[0]304" = load i1, i1* %"#465[0]"
%".mux_3_as[1]'svalue" = and i1 %"#465[0]304", %"#468[0]303"
store i1 %".mux_3_as[1]'svalue", i1* %".mux_3_as[1]"
%"#471[0]305" = load i1, i1* %"#471[0]"
%"#469[0]306" = load i1, i1* %"#469[0]"
%".mux_3_bs[1]'svalue" = and i1 %"#469[0]306", %"#471[0]305"
store i1 %".mux_3_bs[1]'svalue", i1* %".mux_3_bs[1]"
%".mux_3_bs[1]307" = load i1, i1* %".mux_3_bs[1]"
store i1 %".mux_3_bs[1]307", i1* %"#473[0]"
%".mux_3_as[1]308" = load i1, i1* %".mux_3_as[1]"
store i1 %".mux_3_as[1]308", i1* %"#472[0]"
%".mux_1_o[2]309" = load i1, i1* %".mux_1_o[2]"
store i1 %".mux_1_o[2]309", i1* %".mux2_0_m1[2]"
store i1 true, i1* %"#475[0]"
store i1 true, i1* %"#475[1]"
%"#475[1]310" = load i1, i1* %"#475[1]"
store i1 %"#475[1]310", i1* %"#476[0]"
%"#476[0]311" = load i1, i1* %"#476[0]"
%"#477[0]'svalue" = xor i1 %"#476[0]311", true
store i1 %"#477[0]'svalue", i1* %"#477[0]"
%".mux2_0_m1[2]312" = load i1, i1* %".mux2_0_m1[2]"
store i1 %".mux2_0_m1[2]312", i1* %"#474[0]"
%".mux_2_o[2]313" = load i1, i1* %".mux_2_o[2]"
store i1 %".mux_2_o[2]313", i1* %".mux2_0_m2[2]"
store i1 true, i1* %"#479[0]"
store i1 true, i1* %"#479[1]"
%"#479[1]314" = load i1, i1* %"#479[1]"
store i1 %"#479[1]314", i1* %"#480[0]"
%".mux2_0_m2[2]315" = load i1, i1* %".mux2_0_m2[2]"
store i1 %".mux2_0_m2[2]315", i1* %"#478[0]"
%"#477[0]316" = load i1, i1* %"#477[0]"
%"#474[0]317" = load i1, i1* %"#474[0]"
%".mux_3_as[2]'svalue" = and i1 %"#474[0]317", %"#477[0]316"
store i1 %".mux_3_as[2]'svalue", i1* %".mux_3_as[2]"
%"#480[0]318" = load i1, i1* %"#480[0]"
%"#478[0]319" = load i1, i1* %"#478[0]"
%".mux_3_bs[2]'svalue" = and i1 %"#478[0]319", %"#480[0]318"
store i1 %".mux_3_bs[2]'svalue", i1* %".mux_3_bs[2]"
%".mux_3_bs[2]320" = load i1, i1* %".mux_3_bs[2]"
store i1 %".mux_3_bs[2]320", i1* %"#482[0]"
%".mux_3_as[2]321" = load i1, i1* %".mux_3_as[2]"
store i1 %".mux_3_as[2]321", i1* %"#481[0]"
%".mux_1_o[3]322" = load i1, i1* %".mux_1_o[3]"
store i1 %".mux_1_o[3]322", i1* %".mux2_0_m1[3]"
store i1 true, i1* %"#484[0]"
store i1 true, i1* %"#484[1]"
%"#484[1]323" = load i1, i1* %"#484[1]"
store i1 %"#484[1]323", i1* %"#485[0]"
%"#485[0]324" = load i1, i1* %"#485[0]"
%"#486[0]'svalue" = xor i1 %"#485[0]324", true
store i1 %"#486[0]'svalue", i1* %"#486[0]"
%".mux2_0_m1[3]325" = load i1, i1* %".mux2_0_m1[3]"
store i1 %".mux2_0_m1[3]325", i1* %"#483[0]"
%".mux_2_o[3]326" = load i1, i1* %".mux_2_o[3]"
store i1 %".mux_2_o[3]326", i1* %".mux2_0_m2[3]"
store i1 true, i1* %"#488[0]"
store i1 true, i1* %"#488[1]"
%"#488[1]327" = load i1, i1* %"#488[1]"
store i1 %"#488[1]327", i1* %"#489[0]"
%".mux2_0_m2[3]328" = load i1, i1* %".mux2_0_m2[3]"
store i1 %".mux2_0_m2[3]328", i1* %"#487[0]"
%"#486[0]329" = load i1, i1* %"#486[0]"
%"#483[0]330" = load i1, i1* %"#483[0]"
%".mux_3_as[3]'svalue" = and i1 %"#483[0]330", %"#486[0]329"
store i1 %".mux_3_as[3]'svalue", i1* %".mux_3_as[3]"
%"#489[0]331" = load i1, i1* %"#489[0]"
%"#487[0]332" = load i1, i1* %"#487[0]"
```

```
%".mux_3_bs[3]'svalue" = and i1 %"#487[0]332", %"#489[0]331"
store i1 %".mux_3_bs[3]'svalue", i1* %".mux_3_bs[3]"
%".mux_3_bs[3]333" = load i1, i1* %".mux_3_bs[3]"
store i1 %".mux_3_bs[3]333", i1* %"#491[0]"
%".mux_3_as[3]334" = load i1, i1* %".mux_3_as[3]"
store i1 %".mux_3_as[3]334", i1* %"#490[0]"
%".mux_1_o[4]335" = load i1, i1* %".mux_1_o[4]"
store i1 %".mux_1_o[4]335", i1* %".mux2_0_m1[4]"
store i1 true, i1* %"#493[0]"
store i1 true, i1* %"#493[1]"
%"#493[1]336" = load i1, i1* %"#493[1]"
store i1 %"#493[1]336", i1* %"#494[0]"
%"#494[0]337" = load i1, i1* %"#494[0]"
%"#495[0]'svalue" = xor i1 %"#494[0]337", true
store i1 %"#495[0]'svalue", i1* %"#495[0]"
%".mux2_0_m1[4]338" = load i1, i1* %".mux2_0_m1[4]"
store i1 %".mux2_0_m1[4]338", i1* %"#492[0]"
%".mux_2_o[4]339" = load i1, i1* %".mux_2_o[4]"
store i1 %".mux_2_o[4]339", i1* %".mux2_0_m2[4]"
store i1 true, i1* %"#497[0]"
store i1 true, i1* %"#497[1]"
%"#497[1]340" = load i1, i1* %"#497[1]"
store i1 %"#497[1]340", i1* %"#498[0]"
%".mux2_0_m2[4]341" = load i1, i1* %".mux2_0_m2[4]"
store i1 %".mux2_0_m2[4]341", i1* %"#496[0]"
%"#495[0]342" = load i1, i1* %"#495[0]"
%"#492[0]343" = load i1, i1* %"#492[0]"
%".mux_3_as[4]'svalue" = and i1 %"#492[0]343", %"#495[0]342"
store i1 %".mux_3_as[4]'svalue", i1* %".mux_3_as[4]"
%"#498[0]344" = load i1, i1* %"#498[0]"
%"#496[0]345" = load i1, i1* %"#496[0]"
%".mux_3_bs[4]'svalue" = and i1 %"#496[0]345", %"#498[0]344"
store i1 %".mux_3_bs[4]'svalue", i1* %".mux_3_bs[4]"
%".mux_3_bs[4]346" = load i1, i1* %".mux_3_bs[4]"
store i1 %".mux_3_bs[4]346", i1* %"#500[0]"
%".mux_3_as[4]347" = load i1, i1* %".mux_3_as[4]"
store i1 %".mux_3_as[4]347", i1* %"#499[0]"
%".mux_1_o[5]348" = load i1, i1* %".mux_1_o[5]"
store i1 %".mux_1_o[5]348", i1* %".mux2_0_m1[5]"
store i1 true, i1* %"#502[0]"
store i1 true, i1* %"#502[1]"
%"#502[1]349" = load i1, i1* %"#502[1]"
store i1 %"#502[1]349", i1* %"#503[0]"
```

```
%"#503[0]350" = load i1, i1* %"#503[0]"
%"#504[0]'svalue" = xor i1 %"#503[0]350", true
store i1 %"#504[0]'svalue", i1* %"#504[0]"
%".mux2_0_m1[5]351" = load i1, i1* %".mux2_0_m1[5]"
store i1 %".mux2_0_m1[5]351", i1* %"#501[0]"
%".mux_2_o[5]352" = load i1, i1* %".mux_2_o[5]"
store i1 %".mux_2_o[5]352", i1* %".mux2_0_m2[5]"
store i1 true, i1* %"#506[0]"
store i1 true, i1* %"#506[1]"
%"#506[1]353" = load i1, i1* %"#506[1]"
store i1 %"#506[1]353", i1* %"#507[0]"
%".mux2_0_m2[5]354" = load i1, i1* %".mux2_0_m2[5]"
store i1 %".mux2_0_m2[5]354", i1* %"#505[0]"
%"#504[0]355" = load i1, i1* %"#504[0]"
%"#501[0]356" = load i1, i1* %"#501[0]"
%".mux_3_as[5]'svalue" = and i1 %"#501[0]356", %"#504[0]355"
store i1 %".mux_3_as[5]'svalue", i1* %".mux_3_as[5]"
%"#507[0]357" = load i1, i1* %"#507[0]"
%"#505[0]358" = load i1, i1* %"#505[0]"
%".mux_3_bs[5]'svalue" = and i1 %"#505[0]358", %"#507[0]357"
store i1 %".mux_3_bs[5]'svalue", i1* %".mux_3_bs[5]"
%".mux_3_bs[5]359" = load i1, i1* %".mux_3_bs[5]"
store i1 %".mux_3_bs[5]359", i1* %"#509[0]"
%".mux_3_as[5]360" = load i1, i1* %".mux_3_as[5]"
store i1 %".mux_3_as[5]360", i1* %"#508[0]"
%".mux_1_o[6]361" = load i1, i1* %".mux_1_o[6]"
store i1 %".mux_1_o[6]361", i1* %".mux2_0_m1[6]"
store i1 true, i1* %"#511[0]"
store i1 true, i1* %"#511[1]"
%"#511[1]362" = load i1, i1* %"#511[1]"
store i1 %"#511[1]362", i1* %"#512[0]"
%"#512[0]363" = load i1, i1* %"#512[0]"
%"#513[0]'svalue" = xor i1 %"#512[0]363", true
store i1 %"#513[0]'svalue", i1* %"#513[0]"
%".mux2_0_m1[6]364" = load i1, i1* %".mux2_0_m1[6]"
store i1 %".mux2_0_m1[6]364", i1* %"#510[0]"
%".mux_2_o[6]365" = load i1, i1* %".mux_2_o[6]"
store i1 %".mux_2_o[6]365", i1* %".mux2_0_m2[6]"
store i1 true, i1* %"#515[0]"
store i1 true, i1* %"#515[1]"
%"#515[1]366" = load i1, i1* %"#515[1]"
store i1 %"#515[1]366", i1* %"#516[0]"
%".mux2_0_m2[6]367" = load i1, i1* %".mux2_0_m2[6]"
```

```
store i1 %".mux2_0_m2[6]367", i1* %"#514[0]"
%"#513[0]368" = load i1, i1* %"#513[0]"
%"#510[0]369" = load i1, i1* %"#510[0]"
%".mux_3_as[6]'svalue" = and i1 %"#510[0]369", %"#513[0]368"
store i1 %".mux_3_as[6]'svalue", i1* %".mux_3_as[6]"
%"#516[0]370" = load i1, i1* %"#516[0]"
%"#514[0]371" = load i1, i1* %"#514[0]"
%".mux_3_bs[6]'svalue" = and i1 %"#514[0]371", %"#516[0]370"
store i1 %".mux_3_bs[6]'svalue", i1* %".mux_3_bs[6]"
%".mux_3_bs[6]372" = load i1, i1* %".mux_3_bs[6]"
store i1 %".mux_3_bs[6]372", i1* %"#518[0]"
%".mux_3_as[6]373" = load i1, i1* %".mux_3_as[6]"
store i1 %".mux_3_as[6]373", i1* %"#517[0]"
%".mux_1_o[7]374" = load i1, i1* %".mux_1_o[7]"
store i1 %".mux_1_o[7]374", i1* %".mux2_0_m1[7]"
store i1 true, i1* %"#520[0]"
store i1 true, i1* %"#520[1]"
%"#520[1]375" = load i1, i1* %"#520[1]"
store i1 %"#520[1]375", i1* %"#521[0]"
%"#521[0]376" = load i1, i1* %"#521[0]"
%"#522[0]'svalue" = xor i1 %"#521[0]376", true
store i1 %"#522[0]'svalue", i1* %"#522[0]"
%".mux2_0_m1[7]377" = load i1, i1* %".mux2_0_m1[7]"
store i1 %".mux2_0_m1[7]377", i1* %"#519[0]"
%".mux_2_o[7]378" = load i1, i1* %".mux_2_o[7]"
store i1 %".mux_2_o[7]378", i1* %".mux2_0_m2[7]"
store i1 true, i1* %"#524[0]"
store i1 true, i1* %"#524[1]"
%"#524[1]379" = load i1, i1* %"#524[1]"
store i1 %"#524[1]379", i1* %"#525[0]"
%".mux2_0_m2[7]380" = load i1, i1* %".mux2_0_m2[7]"
store i1 %".mux2_0_m2[7]380", i1* %"#523[0]"
%"#522[0]381" = load i1, i1* %"#522[0]"
%"#519[0]382" = load i1, i1* %"#519[0]"
%".mux_3_as[7]'svalue" = and i1 %"#519[0]382", %"#522[0]381"
store i1 %".mux_3_as[7]'svalue", i1* %".mux_3_as[7]"
%"#525[0]383" = load i1, i1* %"#525[0]"
%"#523[0]384" = load i1, i1* %"#523[0]"
%".mux_3_bs[7]'svalue" = and i1 %"#523[0]384", %"#525[0]383"
store i1 %".mux_3_bs[7]'svalue", i1* %".mux_3_bs[7]"
%".mux_3_bs[7]385" = load i1, i1* %".mux_3_bs[7]"
store i1 %".mux_3_bs[7]385", i1* %"#527[0]"
%".mux_3_as[7]386" = load i1, i1* %".mux_3_as[7]"
```

```
store i1 %".mux_3_as[7]386", i1* %"#526[0]"
%"#464[0]387" = load i1, i1* %"#464[0]"
%"#463[0]388" = load i1, i1* %"#463[0]"
%".mux_3_o[0]'svalue" = or i1 %"#463[0]388", %"#464[0]387"
store i1 %".mux_3_o[0]'svalue", i1* %".mux_3_o[0]"
%"#473[0]389" = load i1, i1* %"#473[0]"
%"#472[0]390" = load i1, i1* %"#472[0]"
%".mux_3_o[1]'svalue" = or i1 %"#472[0]390", %"#473[0]389"
store i1 %".mux_3_o[1]'svalue", i1* %".mux_3_o[1]"
%"#482[0]391" = load i1, i1* %"#482[0]"
%"#481[0]392" = load i1, i1* %"#481[0]"
%".mux_3_o[2]'svalue" = or i1 %"#481[0]392", %"#482[0]391"
store i1 %".mux_3_o[2]'svalue", i1* %".mux_3_o[2]"
%"#491[0]393" = load i1, i1* %"#491[0]"
%"#490[0]394" = load i1, i1* %"#490[0]"
%".mux_3_o[3]'svalue" = or i1 %"#490[0]394", %"#491[0]393"
store i1 %".mux_3_o[3]'svalue", i1* %".mux_3_o[3]"
%"#500[0]395" = load i1, i1* %"#500[0]"
%"#499[0]396" = load i1, i1* %"#499[0]"
%".mux_3_o[4]'svalue" = or i1 %"#499[0]396", %"#500[0]395"
store i1 %".mux_3_o[4]'svalue", i1* %".mux_3_o[4]"
%"#509[0]397" = load i1, i1* %"#509[0]"
%"#508[0]398" = load i1, i1* %"#508[0]"
%".mux_3_o[5]'svalue" = or i1 %"#508[0]398", %"#509[0]397"
store i1 %".mux_3_o[5]'svalue", i1* %".mux_3_o[5]"
%"#518[0]399" = load i1, i1* %"#518[0]"
%"#517[0]400" = load i1, i1* %"#517[0]"
%".mux_3_o[6]'svalue" = or i1 %"#517[0]400", %"#518[0]399"
store i1 %".mux_3_o[6]'svalue", i1* %".mux_3_o[6]"
%"#527[0]401" = load i1, i1* %"#527[0]"
%"#526[0]402" = load i1, i1* %"#526[0]"
%".mux_3_o[7]'svalue" = or i1 %"#526[0]402", %"#527[0]401"
store i1 %".mux_3_o[7]'svalue", i1* %".mux_3_o[7]"
%".mux_3_o[0]403" = load i1, i1* %".mux_3_o[0]"
store i1 %".mux_3_o[0]403", i1* %".mux2_0_m[0]"
%".mux_3_o[1]404" = load i1, i1* %".mux_3_o[1]"
store i1 %".mux_3_o[1]404", i1* %".mux2_0_m[1]"
%".mux_3_o[2]405" = load i1, i1* %".mux_3_o[2]"
store i1 %".mux_3_o[2]405", i1* %".mux2_0_m[2]"
%".mux_3_o[3]406" = load i1, i1* %".mux_3_o[3]"
store i1 %".mux_3_o[3]406", i1* %".mux2_0_m[3]"
%".mux_3_o[4]407" = load i1, i1* %".mux_3_o[4]"
store i1 %".mux_3_o[4]407", i1* %".mux2_0_m[4]"
```

```
%".mux_3_o[5]408" = load i1, i1* %".mux_3_o[5]"
store i1 %".mux_3_o[5]408", i1* %".mux2_0_m[5]"
%".mux_3_o[6]409" = load i1, i1* %".mux_3_o[6]"
store i1 %".mux_3_o[6]409", i1* %".mux2_0_m[6]"
%".mux_3_o[7]410" = load i1, i1* %".mux_3_o[7]"
store i1 %".mux_3_o[7]410", i1* %".mux2_0_m[7]"
%".mux2_0_m[0]411" = load i1, i1* %".mux2_0_m[0]"
store i1 %".mux2_0_m[0]411", i1* %".logic_0_o[0]"
%".mux2_0_m[1]412" = load i1, i1* %".mux2_0_m[1]"
store i1 %".mux2_0_m[1]412", i1* %".logic_0_o[1]"
%".mux2_0_m[2]413" = load i1, i1* %".mux2_0_m[2]"
store i1 %".mux2_0_m[2]413", i1* %".logic_0_o[2]"
%".mux2_0_m[3]414" = load i1, i1* %".mux2_0_m[3]"
store i1 %".mux2_0_m[3]414", i1* %".logic_0_o[3]"
%".mux2_0_m[4]415" = load i1, i1* %".mux2_0_m[4]"
store i1 %".mux2_0_m[4]415", i1* %".logic_0_o[4]"
%".mux2_0_m[5]416" = load i1, i1* %".mux2_0_m[5]"
store i1 %".mux2_0_m[5]416", i1* %".logic_0_o[5]"
%".mux2_0_m[6]417" = load i1, i1* %".mux2_0_m[6]"
store i1 %".mux2_0_m[6]417", i1* %".logic_0_o[6]"
%".mux2_0_m[7]418" = load i1, i1* %".mux2_0_m[7]"
store i1 %".mux2_0_m[7]418", i1* %".logic_0_o[7]"
store i1 true, i1* %"#529[0]"
%"#529[0]419" = load i1, i1* %"#529[0]"
%"#530[0]'svalue" = xor i1 %"#529[0]419", true
store i1 %"#530[0]'svalue", i1* %"#530[0]"
%".logic_0_o[0]420" = load i1, i1* %".logic_0_o[0]"
store i1 %".logic_0_o[0]420", i1* %".alu_0_l[0]"
store i1 true, i1* %"#532[0]"
%".alu_0_l[0]421" = load i1, i1* %".alu_0_l[0]"
store i1 %".alu_0_l[0]421", i1* %"#531[0]"
%"#532[0]422" = load i1, i1* %"#532[0]"
%"#531[0]423" = load i1, i1* %"#531[0]"
%".mux_4_bs[0]'svalue" = and i1 %"#531[0]423", %"#532[0]422"
store i1 %".mux_4_bs[0]'svalue", i1* %".mux_4_bs[0]"
%".mux_4_bs[0]424" = load i1, i1* %".mux_4_bs[0]"
store i1 %".mux_4_bs[0]424", i1* %"#534[0]"
store i1 true, i1* %"#536[0]"
%"#536[0]425" = load i1, i1* %"#536[0]"
%"#537[0]'svalue" = xor i1 %"#536[0]425", true
store i1 %"#537[0]'svalue", i1* %"#537[0]"
%".logic_0_o[1]426" = load i1, i1* %".logic_0_o[1]"
store i1 %".logic_0_o[1]426", i1* %".alu_0_l[1]"
store i1 true, i1* %"#539[0]"
%".alu_0_l[1]427" = load i1, i1* %".alu_0_l[1]"
store i1 %".alu_0_l[1]427", i1* %"#538[0]"
%"#539[0]428" = load i1, i1* %"#539[0]"
%"#538[0]429" = load i1, i1* %"#538[0]"
%".mux_4_bs[1]'svalue" = and i1 %"#538[0]429", %"#539[0]428"
store i1 %".mux_4_bs[1]'svalue", i1* %".mux_4_bs[1]"
%".mux_4_bs[1]430" = load i1, i1* %".mux_4_bs[1]"
store i1 %".mux_4_bs[1]430", i1* %"#541[0]"
store i1 true, i1* %"#543[0]"
%"#543[0]431" = load i1, i1* %"#543[0]"
%"#544[0]'svalue" = xor i1 %"#543[0]431", true
store i1 %"#544[0]'svalue", i1* %"#544[0]"
%".logic_0_o[2]432" = load i1, i1* %".logic_0_o[2]"
store i1 %".logic_0_o[2]432", i1* %".alu_0_l[2]"
store i1 true, i1* %"#546[0]"
%".alu_0_l[2]433" = load i1, i1* %".alu_0_l[2]"
store i1 %".alu_0_l[2]433", i1* %"#545[0]"
%"#546[0]434" = load i1, i1* %"#546[0]"
%"#545[0]435" = load i1, i1* %"#545[0]"
%".mux_4_bs[2]'svalue" = and i1 %"#545[0]435", %"#546[0]434"
store i1 %".mux_4_bs[2]'svalue", i1* %".mux_4_bs[2]"
%".mux_4_bs[2]436" = load i1, i1* %".mux_4_bs[2]"
store i1 %".mux_4_bs[2]436", i1* %"#548[0]"
store i1 true, i1* %"#550[0]"
%"#550[0]437" = load i1, i1* %"#550[0]"
%"#551[0]'svalue" = xor i1 %"#550[0]437", true
store i1 %"#551[0]'svalue", i1* %"#551[0]"
%".logic_0_o[3]438" = load i1, i1* %".logic_0_o[3]"
store i1 %".logic_0_o[3]438", i1* %".alu_0_l[3]"
store i1 true, i1* %"#553[0]"
%".alu_0_l[3]439" = load i1, i1* %".alu_0_l[3]"
store i1 %".alu_0_l[3]439", i1* %"#552[0]"
%"#553[0]440" = load i1, i1* %"#553[0]"
%"#552[0]441" = load i1, i1* %"#552[0]"
%".mux_4_bs[3]'svalue" = and i1 %"#552[0]441", %"#553[0]440"
store i1 %".mux_4_bs[3]'svalue", i1* %".mux_4_bs[3]"
%".mux_4_bs[3]442" = load i1, i1* %".mux_4_bs[3]"
store i1 %".mux_4_bs[3]442", i1* %"#555[0]"
store i1 true, i1* %"#557[0]"
%"#557[0]443" = load i1, i1* %"#557[0]"
%"#558[0]'svalue" = xor i1 %"#557[0]443", true
store i1 %"#558[0]'svalue", i1* %"#558[0]"
```

```
%".logic_0_o[4]444" = load i1, i1* %".logic_0_o[4]"
store i1 %".logic_0_o[4]444", i1* %".alu_0_l[4]"
store i1 true, i1* %"#560[0]"
%".alu_0_l[4]445" = load i1, i1* %".alu_0_l[4]"
store i1 %".alu_0_l[4]445", i1* %"#559[0]"
%"#560[0]446" = load i1, i1* %"#560[0]"
%"#559[0]447" = load i1, i1* %"#559[0]"
%".mux_4_bs[4]'svalue" = and i1 %"#559[0]447", %"#560[0]446"
store i1 %".mux_4_bs[4]'svalue", i1* %".mux_4_bs[4]"
%".mux_4_bs[4]448" = load i1, i1* %".mux_4_bs[4]"
store i1 %".mux_4_bs[4]448", i1* %"#562[0]"
store i1 true, i1* %"#564[0]"
%"#564[0]449" = load i1, i1* %"#564[0]"
%"#565[0]'svalue" = xor i1 %"#564[0]449", true
store i1 %"#565[0]'svalue", i1* %"#565[0]"
%".logic_0_o[5]450" = load i1, i1* %".logic_0_o[5]"
store i1 %".logic_0_o[5]450", i1* %".alu_0_l[5]"
store i1 true, i1* %"#567[0]"
%".alu_0_l[5]451" = load i1, i1* %".alu_0_l[5]"
store i1 %".alu_0_l[5]451", i1* %"#566[0]"
%"#567[0]452" = load i1, i1* %"#567[0]"
%"#566[0]453" = load i1, i1* %"#566[0]"
%".mux_4_bs[5]'svalue" = and i1 %"#566[0]453", %"#567[0]452"
store i1 %".mux_4_bs[5]'svalue", i1* %".mux_4_bs[5]"
%".mux_4_bs[5]454" = load i1, i1* %".mux_4_bs[5]"
store i1 %".mux_4_bs[5]454", i1* %"#569[0]"
store i1 true, i1* %"#571[0]"
%"#571[0]455" = load i1, i1* %"#571[0]"
%"#572[0]'svalue" = xor i1 %"#571[0]455", true
store i1 %"#572[0]'svalue", i1* %"#572[0]"
%".logic_0_o[6]456" = load i1, i1* %".logic_0_o[6]"
store i1 %".logic_0_o[6]456", i1* %".alu_0_l[6]"
store i1 true, i1* %"#574[0]"
%".alu_0_l[6]457" = load i1, i1* %".alu_0_l[6]"
store i1 %".alu_0_l[6]457", i1* %"#573[0]"
%"#574[0]458" = load i1, i1* %"#574[0]"
%"#573[0]459" = load i1, i1* %"#573[0]"
%".mux_4_bs[6]'svalue" = and i1 %"#573[0]459", %"#574[0]458"
store i1 %".mux_4_bs[6]'svalue", i1* %".mux_4_bs[6]"
%".mux_4_bs[6]460" = load i1, i1* %".mux_4_bs[6]"
store i1 %".mux_4_bs[6]460", i1* %"#576[0]"
store i1 true, i1* %"#578[0]"
%"#578[0]461" = load i1, i1* %"#578[0]"
```

```
%"#579[0]'svalue" = xor i1 %"#578[0]461", true
store i1 %"#579[0]'svalue", i1* %"#579[0]"
%".logic_0_o[7]462" = load i1, i1* %".logic_0_o[7]"
store i1 %".logic_0_o[7]462", i1* %".alu_0_l[7]"
store i1 true, i1* %"#581[0]"
%".alu_0_l[7]463" = load i1, i1* %".alu_0_l[7]"
store i1 %".alu_0_l[7]463", i1* %"#580[0]"
%"#581[0]464" = load i1, i1* %"#581[0]"
%"#580[0]465" = load i1, i1* %"#580[0]"
%".mux_4_bs[7]'svalue" = and i1 %"#580[0]465", %"#581[0]464"
store i1 %".mux_4_bs[7]'svalue", i1* %".mux_4_bs[7]"
%".mux_4_bs[7]466" = load i1, i1* %".mux_4_bs[7]"
store i1 %".mux_4_bs[7]466", i1* %"#583[0]"
%a_7467 = load i1, i1* %a_7
store i1 %a_7467, i1* %"#309[0]"
%a_7468 = load i1, i1* %a_7
store i1 %a_7468, i1* %"#304[0]"
%a_7469 = load i1, i1* %a_7
store i1 %a_7469, i1* %"#301[0]"
%a_7470 = load i1, i1* %a_7
store i1 %a_7470, i1* %"#296[0]"
%a_6471 = load i1, i1* %a_6
store i1 %a_6471, i1* %"#293[0]"
%a_6472 = load i1, i1* %a_6
store i1 %a_6472, i1* %"#288[0]"
%a_6473 = load i1, i1* %a_6
store i1 %a_6473, i1* %"#285[0]"
%a_6474 = load i1, i1* %a_6
store i1 %a_6474, i1* %"#280[0]"
%a_5475 = load i1, i1* %a_5
store i1 %a_5475, i1* %"#277[0]"
%a_5476 = load i1, i1* %a_5
store i1 %a_5476, i1* %"#272[0]"
%a_5477 = load i1, i1* %a_5
store i1 %a_5477, i1* %"#269[0]"
%a_5478 = load i1, i1* %a_5
store i1 %a_5478, i1* %"#264[0]"
%a_4479 = load i1, i1* %a_4
store i1 %a_4479, i1* %"#261[0]"
%a_4480 = load i1, i1* %a_4
store i1 %a_4480, i1* %"#256[0]"
%a_4481 = load i1, i1* %a_4
store i1 %a_4481, i1* %"#253[0]"
```

```
%a_4482 = load i1, i1* %a_4
store i1 %a_4482, i1* %"#248[0]"
%a_3483 = load i1, i1* %a_3
store i1 %a_3483, i1* %"#245[0]"
%a_3484 = load i1, i1* %a_3
store i1 %a_3484, i1* %"#240[0]"
%a_3485 = load i1, i1* %a_3
store i1 %a_3485, i1* %"#237[0]"
%a_3486 = load i1, i1* %a_3
store i1 %a_3486, i1* %"#232[0]"
%a_2487 = load i1, i1* %a_2
store i1 %a_2487, i1* %"#229[0]"
%a_2488 = load i1, i1* %a_2
store i1 %a_2488, i1* %"#224[0]"
%a_2489 = load i1, i1* %a_2
store i1 %a_2489, i1* %"#221[0]"
%a_2490 = load i1, i1* %a_2
store i1 %a_2490, i1* %"#216[0]"
%a_1491 = load i1, i1* %a_1
store i1 %a_1491, i1* %"#213[0]"
%a_1492 = load i1, i1* %a_1
store i1 %a_1492, i1* %"#208[0]"
%a_1493 = load i1, i1* %a_1
store i1 %a_1493, i1* %"#205[0]"
%a_1494 = load i1, i1* %a_1
store i1 %a_1494, i1* %"#200[0]"
%a_0495 = load i1, i1* %a_0
store i1 %a_0495, i1* %"#197[0]"
%".add_1_c[0]496" = load i1, i1* %".add_1_c[0]"
store i1 %".add_1_c[0]496", i1* %"#194[0]"
%a_0497 = load i1, i1* %a_0
store i1 %a_0497, i1* %"#192[0]"
%a_0498 = load i1, i1* %a_0
store i1 %a_0498, i1* %"#189[0]"
%".add_1_c[0]499" = load i1, i1* %".add_1_c[0]"
store i1 %".add_1_c[0]499", i1* %"#187[0]"
%".add_1_c[0]500" = load i1, i1* %".add_1_c[0]"
store i1 %".add_1_c[0]500", i1* %"#186[0]"
%a_0501 = load i1, i1* %a_0
store i1 %a_0501, i1* %"#184[0]"
%b_7502 = load i1, i1* %b_7
store i1 %b_7502, i1* %"#177[0]"
%b_6503 = load i1, i1* %b_6
```

```
store i1 %b_6503, i1* %"#170[0]"
%b_5504 = load i1, i1* %b_5
store i1 %b_5504, i1* %"#163[0]"
%b_4505 = load i1, i1* %b_4
store i1 %b_4505, i1* %"#156[0]"
%b_3506 = load i1, i1* %b_3
store i1 %b_3506, i1* %"#149[0]"
%b_2507 = load i1, i1* %b_2
store i1 %b_2507, i1* %"#142[0]"
%b_1508 = load i1, i1* %b_1
store i1 %b_1508, i1* %"#135[0]"
%b_0509 = load i1, i1* %b_0
store i1 %b_0509, i1* %"#128[0]"
%b_7510 = load i1, i1* %b_7
%".comp_0_a1[7]'svalue" = xor i1 %b_7510, true
store i1 %".comp_0_a1[7]'svalue", i1* %".comp_0_a1[7]"
%".comp_0_a1[7]511" = load i1, i1* %".comp_0_a1[7]"
store i1 %".comp_0_a1[7]511", i1* %"#120[0]"
%".comp_0_one[7]512" = load i1, i1* %".comp_0_one[7]"
store i1 %".comp_0_one[7]512", i1* %"#121[0]"
%".comp_0_a1[7]513" = load i1, i1* %".comp_0_a1[7]"
store i1 %".comp_0_a1[7]513", i1* %"#117[0]"
%".comp_0_one[7]514" = load i1, i1* %".comp_0_one[7]"
store i1 %".comp_0_one[7]514", i1* %"#118[0]"
%".comp_0_a1[7]515" = load i1, i1* %".comp_0_a1[7]"
store i1 %".comp_0_a1[7]515", i1* %"#112[0]"
%".comp_0_one[7]516" = load i1, i1* %".comp_0_one[7]"
store i1 %".comp_0_one[7]516", i1* %"#113[0]"
%b_6517 = load i1, i1* %b_6
%".comp_0_a1[6]'svalue" = xor i1 %b_6517, true
store i1 %".comp_0_a1[6]'svalue", i1* %".comp_0_a1[6]"
%".comp_0_a1[6]518" = load i1, i1* %".comp_0_a1[6]"
store i1 %".comp_0_a1[6]518", i1* %"#104[0]"
%".comp_0_one[6]519" = load i1, i1* %".comp_0_one[6]"
store i1 %".comp_0_one[6]519", i1* %"#105[0]"
%".comp_0_a1[6]520" = load i1, i1* %".comp_0_a1[6]"
store i1 %".comp_0_a1[6]520", i1* %"#101[0]"
%".comp_0_one[6]521" = load i1, i1* %".comp_0_one[6]"
store i1 %".comp_0_one[6]521", i1* %"#102[0]"
%".comp_0_a1[6]522" = load i1, i1* %".comp_0_a1[6]"
store i1 %".comp_0_a1[6]522", i1* %"#96[0]"
%".comp_0_one[6]523" = load i1, i1* %".comp_0_one[6]"
store i1 %".comp_0_one[6]523", i1* %"#97[0]"
```

```
%b_5524 = load i1, i1* %b_5
%".comp_0_a1[5]'svalue" = xor i1 %b_5524, true
store i1 %".comp_0_a1[5]'svalue", i1* %".comp_0_a1[5]"
%".comp_0_a1[5]525" = load i1, i1* %".comp_0_a1[5]"
store i1 %".comp_0_a1[5]525", i1* %"#88[0]"
%".comp_0_one[5]526" = load i1, i1* %".comp_0_one[5]"
store i1 %".comp_0_one[5]526", i1* %"#89[0]"
%".comp_0_a1[5]527" = load i1, i1* %".comp_0_a1[5]"
store i1 %".comp_0_a1[5]527", i1* %"#85[0]"
%".comp_0_one[5]528" = load i1, i1* %".comp_0_one[5]"
store i1 %".comp_0_one[5]528", i1* %"#86[0]"
%".comp_0_a1[5]529" = load i1, i1* %".comp_0_a1[5]"
store i1 %".comp_0_a1[5]529", i1* %"#80[0]"
%".comp_0_one[5]530" = load i1, i1* %".comp_0_one[5]"
store i1 %".comp_0_one[5]530", i1* %"#81[0]"
%b_4531 = load i1, i1* %b_4
%".comp_0_a1[4]'svalue" = xor i1 %b_4531, true
store i1 %".comp_0_a1[4]'svalue", i1* %".comp_0_a1[4]"
%".comp_0_a1[4]532" = load i1, i1* %".comp_0_a1[4]"
store i1 %".comp_0_a1[4]532", i1* %"#72[0]"
%".comp_0_one[4]533" = load i1, i1* %".comp_0_one[4]"
store i1 %".comp_0_one[4]533", i1* %"#73[0]"
%".comp_0_a1[4]534" = load i1, i1* %".comp_0_a1[4]"
store i1 %".comp_0_a1[4]534", i1* %"#69[0]"
%".comp_0_one[4]535" = load i1, i1* %".comp_0_one[4]"
store i1 %".comp_0_one[4]535", i1* %"#70[0]"
%".comp_0_a1[4]536" = load i1, i1* %".comp_0_a1[4]"
store i1 %".comp_0_a1[4]536", i1* %"#64[0]"
%".comp_0_one[4]537" = load i1, i1* %".comp_0_one[4]"
store i1 %".comp_0_one[4]537", i1* %"#65[0]"
%b_3538 = load i1, i1* %b_3
%".comp_0_a1[3]'svalue" = xor i1 %b_3538, true
store i1 %".comp_0_a1[3]'svalue", i1* %".comp_0_a1[3]"
%".comp_0_a1[3]539" = load i1, i1* %".comp_0_a1[3]"
store i1 %".comp_0_a1[3]539", i1* %"#56[0]"
%".comp_0_one[3]540" = load i1, i1* %".comp_0_one[3]"
store i1 %".comp_0_one[3]540", i1* %"#57[0]"
%".comp_0_a1[3]541" = load i1, i1* %".comp_0_a1[3]"
store i1 %".comp_0_a1[3]541", i1* %"#53[0]"
%".comp_0_one[3]542" = load i1, i1* %".comp_0_one[3]"
store i1 %".comp_0_one[3]542", i1* %"#54[0]"
%".comp_0_a1[3]543" = load i1, i1* %".comp_0_a1[3]"
store i1 %".comp_0_a1[3]543", i1* %"#48[0]"
```

```
%".comp_0_one[3]544" = load i1, i1* %".comp_0_one[3]"
store i1 %".comp_0_one[3]544", i1* %"#49[0]"
%b_2545 = load i1, i1* %b_2
%".comp_0_a1[2]'svalue" = xor i1 %b_2545, true
store i1 %".comp_0_a1[2]'svalue", i1* %".comp_0_a1[2]"
%".comp_0_a1[2]546" = load i1, i1* %".comp_0_a1[2]"
store i1 %".comp_0_a1[2]546", i1* %"#40[0]"
%".comp_0_one[2]547" = load i1, i1* %".comp_0_one[2]"
store i1 %".comp_0_one[2]547", i1* %"#41[0]"
%".comp_0_a1[2]548" = load i1, i1* %".comp_0_a1[2]"
store i1 %".comp_0_a1[2]548", i1* %"#37[0]"
%".comp_0_one[2]549" = load i1, i1* %".comp_0_one[2]"
store i1 %".comp_0_one[2]549", i1* %"#38[0]"
%".comp_0_a1[2]550" = load i1, i1* %".comp_0_a1[2]"
store i1 %".comp_0_a1[2]550", i1* %"#32[0]"
%".comp_0_one[2]551" = load i1, i1* %".comp_0_one[2]"
store i1 %".comp_0_one[2]551", i1* %"#33[0]"
%b_1552 = load i1, i1* %b_1
%".comp_0_a1[1]'svalue" = xor i1 %b_1552, true
store i1 %".comp_0_a1[1]'svalue", i1* %".comp_0_a1[1]"
%".comp_0_a1[1]553" = load i1, i1* %".comp_0_a1[1]"
store i1 %".comp_0_a1[1]553", i1* %"#24[0]"
%".comp_0_one[1]554" = load i1, i1* %".comp_0_one[1]"
store i1 %".comp_0_one[1]554", i1* %"#25[0]"
%".comp_0_a1[1]555" = load i1, i1* %".comp_0_a1[1]"
store i1 %".comp_0_a1[1]555", i1* %"#21[0]"
%".comp_0_one[1]556" = load i1, i1* %".comp_0_one[1]"
store i1 %".comp_0_one[1]556", i1* %"#22[0]"
%".comp_0_a1[1]557" = load i1, i1* %".comp_0_a1[1]"
store i1 %".comp_0_a1[1]557", i1* %"#16[0]"
%".comp_0_one[1]558" = load i1, i1* %".comp_0_one[1]"
store i1 %".comp_0_one[1]558", i1* %"#17[0]"
%b_0559 = load i1, i1* %b_0
%".comp_0_a1[0]'svalue" = xor i1 %b_0559, true
store i1 %".comp_0_a1[0]'svalue", i1* %".comp_0_a1[0]"
%".add_0_c[0]560" = load i1, i1* %".add_0_c[0]"
store i1 %".add_0_c[0]560", i1* %"#10[0]"
%".comp_0_a1[0]561" = load i1, i1* %".comp_0_a1[0]"
store i1 %".comp_0_a1[0]561", i1* %"#8[0]"
%".comp_0_one[0]562" = load i1, i1* %".comp_0_one[0]"
store i1 %".comp_0_one[0]562", i1* %"#9[0]"
%".comp_0_a1[0]563" = load i1, i1* %".comp_0_a1[0]"
store i1 %".comp_0_a1[0]563", i1* %"#5[0]"
```

```
%".comp_0_one[0]564" = load i1, i1* %".comp_0_one[0]"
store i1 %".comp_0_one[0]564", i1* %"#6[0]"
%".add_0_c[0]565" = load i1, i1* %".add_0_c[0]"
store i1 %".add_0_c[0]565", i1* %"#3[0]"
%".add_0_c[0]566" = load i1, i1* %".add_0_c[0]"
store i1 %".add_0_c[0]566", i1* %"#2[0]"
%".comp_0_a1[0]567" = load i1, i1* %".comp_0_a1[0]"
store i1 %".comp_0_a1[0]567", i1* %"#0[0]"
%".comp_0_one[0]568" = load i1, i1* %".comp_0_one[0]"
store i1 %".comp_0_one[0]568", i1* %"#1[0]"
%"#1[0]569" = load i1, i1* %"#1[0]"
%"#0[0]570" = load i1, i1* %"#0[0]"
%".fA_0_axb[0]'svalue" = xor i1 %"#0[0]570", %"#1[0]569"
store i1 %".fA_0_axb[0]'svalue", i1* %".fA_0_axb[0]"
%"#6[0]571" = load i1, i1* %"#6[0]"
%"#5[0]572" = load i1, i1* %"#5[0]"
%"#7[0]'svalue" = and i1 %"#5[0]572", %"#6[0]571"
store i1 %"#7[0]'svalue", i1* %"#7[0]"
%"#3[0]573" = load i1, i1* %"#3[0]"
%".fA_0_axb[0]574" = load i1, i1* %".fA_0_axb[0]"
%"#4[0]'svalue" = and i1 %".fA_0_axb[0]574", %"#3[0]573"
store i1 %"#4[0]'svalue", i1* %"#4[0]"
%"#7[0]575" = load i1, i1* %"#7[0]"
%"#4[0]576" = load i1, i1* %"#4[0]"
%".fA_0_carry[0]'svalue" = or i1 %"#4[0]576", %"#7[0]575"
store i1 %".fA_0_carry[0]'svalue", i1* %".fA_0_carry[0]"
%"#2[0]577" = load i1, i1* %"#2[0]"
%".fA_0_axb[0]578" = load i1, i1* %".fA_0_axb[0]"
%".fA_0_sum[0]'svalue" = xor i1 %".fA_0_axb[0]578", %"#2[0]577"
store i1 %".fA_0_sum[0]'svalue", i1* %".fA_0_sum[0]"
%"#9[0]579" = load i1, i1* %"#9[0]"
%"#8[0]580" = load i1, i1* %"#8[0]"
%".fA_1_axb[0]'svalue" = xor i1 %"#8[0]580", %"#9[0]579"
store i1 %".fA_1_axb[0]'svalue", i1* %".fA_1_axb[0]"
%".comp_0_a1[0]581" = load i1, i1* %".comp_0_a1[0]"
store i1 %".comp_0_a1[0]581", i1* %"#13[0]"
%"#14[0]582" = load i1, i1* %"#14[0]"
%"#13[0]583" = load i1, i1* %"#13[0]"
%"#15[0]'svalue" = and i1 %"#13[0]583", %"#14[0]582"
store i1 %"#15[0]'svalue", i1* %"#15[0]"
%"#11[0]584" = load i1, i1* %"#11[0]"
%".fA_1_axb[0]585" = load i1, i1* %".fA_1_axb[0]"
%"#12[0]'svalue" = and i1 %".fA_1_axb[0]585", %"#11[0]584"
```

```
store i1 %"#12[0]'svalue", i1* %"#12[0]"
%"#15[0]586" = load i1, i1* %"#15[0]"
%"#12[0]587" = load i1, i1* %"#12[0]"
%".fA_1_carry[0]'svalue" = or i1 %"#12[0]587", %"#15[0]586"
store i1 %".fA_1_carry[0]'svalue", i1* %".fA_1_carry[0]"
%"#17[0]588" = load i1, i1* %"#17[0]"
%"#16[0]589" = load i1, i1* %"#16[0]"
%".fA_2_axb[0]'svalue" = xor i1 %"#16[0]589", %"#17[0]588"
store i1 %".fA_2_axb[0]'svalue", i1* %".fA_2_axb[0]"
%"#22[0]590" = load i1, i1* %"#22[0]"
%"#21[0]591" = load i1, i1* %"#21[0]"
%"#23[0]'svalue" = and i1 %"#21[0]591", %"#22[0]590"
store i1 %"#23[0]'svalue", i1* %"#23[0]"
%".fA_1_carry[0]592" = load i1, i1* %".fA_1_carry[0]"
store i1 %".fA_1_carry[0]592", i1* %".add_0_c[1]"
%".add_0_c[1]593" = load i1, i1* %".add_0_c[1]"
store i1 %".add_0_c[1]593", i1* %"#27[0]"
%"#25[0]594" = load i1, i1* %"#25[0]"
%"#24[0]595" = load i1, i1* %"#24[0]"
%".fA_3_axb[0]'svalue" = xor i1 %"#24[0]595", %"#25[0]594"
store i1 %".fA_3_axb[0]'svalue", i1* %".fA_3_axb[0]"
%".comp_0_a1[1]596" = load i1, i1* %".comp_0_a1[1]"
store i1 %".comp_0_a1[1]596", i1* %"#29[0]"
%"#30[0]597" = load i1, i1* %"#30[0]"
%"#29[0]598" = load i1, i1* %"#29[0]"
%"#31[0]'svalue" = and i1 %"#29[0]598", %"#30[0]597"
store i1 %"#31[0]'svalue", i1* %"#31[0]"
%"#27[0]599" = load i1, i1* %"#27[0]"
%".fA_3_axb[0]600" = load i1, i1* %".fA_3_axb[0]"
%"#28[0]'svalue" = and i1 %".fA_3_axb[0]600", %"#27[0]599"
store i1 %"#28[0]'svalue", i1* %"#28[0]"
%"#31[0]601" = load i1, i1* %"#31[0]"
%"#28[0]602" = load i1, i1* %"#28[0]"
%".fA_3_carry[0]'svalue" = or i1 %"#28[0]602", %"#31[0]601"
store i1 %".fA_3_carry[0]'svalue", i1* %".fA_3_carry[0]"
%"#33[0]603" = load i1, i1* %"#33[0]"
%"#32[0]604" = load i1, i1* %"#32[0]"
%".fA_4_axb[0]'svalue" = xor i1 %"#32[0]604", %"#33[0]603"
store i1 %".fA_4_axb[0]'svalue", i1* %".fA_4_axb[0]"
%"#38[0]605" = load i1, i1* %"#38[0]"
%"#37[0]606" = load i1, i1* %"#37[0]"
%"#39[0]'svalue" = and i1 %"#37[0]606", %"#38[0]605"
store i1 %"#39[0]'svalue", i1* %"#39[0]"
```

```
%".fA_3_carry[0]607" = load i1, i1* %".fA_3_carry[0]"
store i1 %".fA_3_carry[0]607", i1* %".add_0_c[2]"
%".add_0_c[2]608" = load i1, i1* %".add_0_c[2]"
store i1 %".add_0_c[2]608", i1* %"#43[0]"
%"#41[0]609" = load i1, i1* %"#41[0]"
%"#40[0]610" = load i1, i1* %"#40[0]"
%".fA_5_axb[0]'svalue" = xor i1 %"#40[0]610", %"#41[0]609"
store i1 %".fA_5_axb[0]'svalue", i1* %".fA_5_axb[0]"
%".comp_0_a1[2]611" = load i1, i1* %".comp_0_a1[2]"
store i1 %".comp_0_a1[2]611", i1* %"#45[0]"
%"#46[0]612" = load i1, i1* %"#46[0]"
%"#45[0]613" = load i1, i1* %"#45[0]"
%"#47[0]'svalue" = and i1 %"#45[0]613", %"#46[0]612"
store i1 %"#47[0]'svalue", i1* %"#47[0]"
%"#43[0]614" = load i1, i1* %"#43[0]"
%".fA_5_axb[0]615" = load i1, i1* %".fA_5_axb[0]"
%"#44[0]'svalue" = and i1 %".fA_5_axb[0]615", %"#43[0]614"
store i1 %"#44[0]'svalue", i1* %"#44[0]"
%"#47[0]616" = load i1, i1* %"#47[0]"
%"#44[0]617" = load i1, i1* %"#44[0]"
%".fA_5_carry[0]'svalue" = or i1 %"#44[0]617", %"#47[0]616"
store i1 %".fA_5_carry[0]'svalue", i1* %".fA_5_carry[0]"
%"#49[0]618" = load i1, i1* %"#49[0]"
%"#48[0]619" = load i1, i1* %"#48[0]"
%".fA_6_axb[0]'svalue" = xor i1 %"#48[0]619", %"#49[0]618"
store i1 %".fA_6_axb[0]'svalue", i1* %".fA_6_axb[0]"
%"#54[0]620" = load i1, i1* %"#54[0]"
%"#53[0]621" = load i1, i1* %"#53[0]"
%"#55[0]'svalue" = and i1 %"#53[0]621", %"#54[0]620"
store i1 %"#55[0]'svalue", i1* %"#55[0]"
%".fA_5_carry[0]622" = load i1, i1* %".fA_5_carry[0]"
store i1 %".fA_5_carry[0]622", i1* %".add_0_c[3]"
%".add_0_c[3]623" = load i1, i1* %".add_0_c[3]"
store i1 %".add_0_c[3]623", i1* %"#59[0]"
%"#57[0]624" = load i1, i1* %"#57[0]"
%"#56[0]625" = load i1, i1* %"#56[0]"
%".fA_7_axb[0]'svalue" = xor i1 %"#56[0]625", %"#57[0]624"
store i1 %".fA_7_axb[0]'svalue", i1* %".fA_7_axb[0]"
%".comp_0_a1[3]626" = load i1, i1* %".comp_0_a1[3]"
store i1 %".comp_0_a1[3]626", i1* %"#61[0]"
%"#62[0]627" = load i1, i1* %"#62[0]"
%"#61[0]628" = load i1, i1* %"#61[0]"
%"#63[0]'svalue" = and i1 %"#61[0]628", %"#62[0]627"
```

```
store i1 %"#63[0]'svalue", i1* %"#63[0]"
%"#59[0]629" = load i1, i1* %"#59[0]"
%".fA_7_axb[0]630" = load i1, i1* %".fA_7_axb[0]"
%"#60[0]'svalue" = and i1 %".fA_7_axb[0]630", %"#59[0]629"
store i1 %"#60[0]'svalue", i1* %"#60[0]"
%"#63[0]631" = load i1, i1* %"#63[0]"
%"#60[0]632" = load i1, i1* %"#60[0]"
%".fA_7_carry[0]'svalue" = or i1 %"#60[0]632", %"#63[0]631"
store i1 %".fA_7_carry[0]'svalue", i1* %".fA_7_carry[0]"
%"#65[0]633" = load i1, i1* %"#65[0]"
%"#64[0]634" = load i1, i1* %"#64[0]"
%".fA_8_axb[0]'svalue" = xor i1 %"#64[0]634", %"#65[0]633"
store i1 %".fA_8_axb[0]'svalue", i1* %".fA_8_axb[0]"
%"#70[0]635" = load i1, i1* %"#70[0]"
%"#69[0]636" = load i1, i1* %"#69[0]"
%"#71[0]'svalue" = and i1 %"#69[0]636", %"#70[0]635"
store i1 %"#71[0]'svalue", i1* %"#71[0]"
%".fA_7_carry[0]637" = load i1, i1* %".fA_7_carry[0]"
store i1 %".fA_7_carry[0]637", i1* %".add_0_c[4]"
%".add_0_c[4]638" = load i1, i1* %".add_0_c[4]"
store i1 %".add_0_c[4]638", i1* %"#75[0]"
%"#73[0]639" = load i1, i1* %"#73[0]"
%"#72[0]640" = load i1, i1* %"#72[0]"
%".fA_9_axb[0]'svalue" = xor i1 %"#72[0]640", %"#73[0]639"
store i1 %".fA_9_axb[0]'svalue", i1* %".fA_9_axb[0]"
%".comp_0_a1[4]641" = load i1, i1* %".comp_0_a1[4]"
store i1 %".comp_0_a1[4]641", i1* %"#77[0]"
%"#78[0]642" = load i1, i1* %"#78[0]"
%"#77[0]643" = load i1, i1* %"#77[0]"
%"#79[0]'svalue" = and i1 %"#77[0]643", %"#78[0]642"
store i1 %"#79[0]'svalue", i1* %"#79[0]"
%"#75[0]644" = load i1, i1* %"#75[0]"
%".fA_9_axb[0]645" = load i1, i1* %".fA_9_axb[0]"
%"#76[0]'svalue" = and i1 %".fA_9_axb[0]645", %"#75[0]644"
store i1 %"#76[0]'svalue", i1* %"#76[0]"
%"#79[0]646" = load i1, i1* %"#79[0]"
%"#76[0]647" = load i1, i1* %"#76[0]"
%".fA_9_carry[0]'svalue" = or i1 %"#76[0]647", %"#79[0]646"
store i1 %".fA_9_carry[0]'svalue", i1* %".fA_9_carry[0]"
%"#81[0]648" = load i1, i1* %"#81[0]"
%"#80[0]649" = load i1, i1* %"#80[0]"
%".fA_10_axb[0]'svalue" = xor i1 %"#80[0]649", %"#81[0]648"
store i1 %".fA_10_axb[0]'svalue", i1* %".fA_10_axb[0]"
```

```
%"#86[0]650" = load i1, i1* %"#86[0]"
%"#85[0]651" = load i1, i1* %"#85[0]"
%"#87[0]'svalue" = and i1 %"#85[0]651", %"#86[0]650"
store i1 %"#87[0]'svalue", i1* %"#87[0]"
%".fA_9_carry[0]652" = load i1, i1* %".fA_9_carry[0]"
store i1 %".fA_9_carry[0]652", i1* %".add_0_c[5]"
%".add_0_c[5]653" = load i1, i1* %".add_0_c[5]"
store i1 %".add_0_c[5]653", i1* %"#91[0]"
%"#89[0]654" = load i1, i1* %"#89[0]"
%"#88[0]655" = load i1, i1* %"#88[0]"
%".fA_11_axb[0]'svalue" = xor i1 %"#88[0]655", %"#89[0]654"
store i1 %".fA_11_axb[0]'svalue", i1* %".fA_11_axb[0]"
%".comp_0_a1[5]656" = load i1, i1* %".comp_0_a1[5]"
store i1 %".comp_0_a1[5]656", i1* %"#93[0]"
%"#94[0]657" = load i1, i1* %"#94[0]"
%"#93[0]658" = load i1, i1* %"#93[0]"
%"#95[0]'svalue" = and i1 %"#93[0]658", %"#94[0]657"
store i1 %"#95[0]'svalue", i1* %"#95[0]"
%"#91[0]659" = load i1, i1* %"#91[0]"
%".fA_11_axb[0]660" = load i1, i1* %".fA_11_axb[0]"
%"#92[0]'svalue" = and i1 %".fA_11_axb[0]660", %"#91[0]659"
store i1 %"#92[0]'svalue", i1* %"#92[0]"
%"#95[0]661" = load i1, i1* %"#95[0]"
%"#92[0]662" = load i1, i1* %"#92[0]"
%".fA_11_carry[0]'svalue" = or i1 %"#92[0]662", %"#95[0]661"
store i1 %".fA_11_carry[0]'svalue", i1* %".fA_11_carry[0]"
%"#97[0]663" = load i1, i1* %"#97[0]"
%"#96[0]664" = load i1, i1* %"#96[0]"
%".fA_12_axb[0]'svalue" = xor i1 %"#96[0]664", %"#97[0]663"
store i1 %".fA_12_axb[0]'svalue", i1* %".fA_12_axb[0]"
%"#102[0]665" = load i1, i1* %"#102[0]"
%"#101[0]666" = load i1, i1* %"#101[0]"
%"#103[0]'svalue" = and i1 %"#101[0]666", %"#102[0]665"
store i1 %"#103[0]'svalue", i1* %"#103[0]"
%".fA_11_carry[0]667" = load i1, i1* %".fA_11_carry[0]"
store i1 %".fA_11_carry[0]667", i1* %".add_0_c[6]"
%".add_0_c[6]668" = load i1, i1* %".add_0_c[6]"
store i1 %".add_0_c[6]668", i1* %"#107[0]"
%"#105[0]669" = load i1, i1* %"#105[0]"
%"#104[0]670" = load i1, i1* %"#104[0]"
%".fA_13_axb[0]'svalue" = xor i1 %"#104[0]670", %"#105[0]669"
store i1 %".fA_13_axb[0]'svalue", i1* %".fA_13_axb[0]"
%".comp_0_a1[6]671" = load i1, i1* %".comp_0_a1[6]"
```

```
store i1 %".comp_0_a1[6]671", i1* %"#109[0]"
%"#110[0]672" = load i1, i1* %"#110[0]"
%"#109[0]673" = load i1, i1* %"#109[0]"
%"#111[0]'svalue" = and i1 %"#109[0]673", %"#110[0]672"
store i1 %"#111[0]'svalue", i1* %"#111[0]"
%"#107[0]674" = load i1, i1* %"#107[0]"
%".fA_13_axb[0]675" = load i1, i1* %".fA_13_axb[0]"
%"#108[0]'svalue" = and i1 %".fA_13_axb[0]675", %"#107[0]674"
store i1 %"#108[0]'svalue", i1* %"#108[0]"
%"#111[0]676" = load i1, i1* %"#111[0]"
%"#108[0]677" = load i1, i1* %"#108[0]"
%".fA_13_carry[0]'svalue" = or i1 %"#108[0]677", %"#111[0]676"
store i1 %".fA_13_carry[0]'svalue", i1* %".fA_13_carry[0]"
%"#113[0]678" = load i1, i1* %"#113[0]"
%"#112[0]679" = load i1, i1* %"#112[0]"
%".fA_14_axb[0]'svalue" = xor i1 %"#112[0]679", %"#113[0]678"
store i1 %".fA_14_axb[0]'svalue", i1* %".fA_14_axb[0]"
%"#118[0]680" = load i1, i1* %"#118[0]"
%"#117[0]681" = load i1, i1* %"#117[0]"
%"#119[0]'svalue" = and i1 %"#117[0]681", %"#118[0]680"
store i1 %"#119[0]'svalue", i1* %"#119[0]"
%".fA_13_carry[0]682" = load i1, i1* %".fA_13_carry[0]"
store i1 %".fA_13_carry[0]682", i1* %".add_0_c[7]"
%".add_0_c[7]683" = load i1, i1* %".add_0_c[7]"
store i1 %".add_0_c[7]683", i1* %"#123[0]"
%"#121[0]684" = load i1, i1* %"#121[0]"
%"#120[0]685" = load i1, i1* %"#120[0]"
%".fA_15_axb[0]'svalue" = xor i1 %"#120[0]685", %"#121[0]684"
store i1 %".fA_15_axb[0]'svalue", i1* %".fA_15_axb[0]"
%".comp_0_a1[7]686" = load i1, i1* %".comp_0_a1[7]"
store i1 %".comp_0_a1[7]686", i1* %"#125[0]"
%"#126[0]687" = load i1, i1* %"#126[0]"
%"#125[0]688" = load i1, i1* %"#125[0]"
%"#127[0]'svalue" = and i1 %"#125[0]688", %"#126[0]687"
store i1 %"#127[0]'svalue", i1* %"#127[0]"
%"#123[0]689" = load i1, i1* %"#123[0]"
%".fA_15_axb[0]690" = load i1, i1* %".fA_15_axb[0]"
%"#124[0]'svalue" = and i1 %".fA_15_axb[0]690", %"#123[0]689"
store i1 %"#124[0]'svalue", i1* %"#124[0]"
%"#127[0]691" = load i1, i1* %"#127[0]"
%"#124[0]692" = load i1, i1* %"#124[0]"
%".fA_15_carry[0]'svalue" = or i1 %"#124[0]692", %"#127[0]691"
store i1 %".fA_15_carry[0]'svalue", i1* %".fA_15_carry[0]"
```

```
%".fA_15_carry[0]693" = load i1, i1* %".fA_15_carry[0]"
store i1 %".fA_15_carry[0]693", i1* %".add_0_c[8]"
%".fA_0_sum[0]694" = load i1, i1* %".fA_0_sum[0]"
store i1 %".fA_0_sum[0]694", i1* %".add_0_sum[0]"
%".add_0_sum[0]695" = load i1, i1* %".add_0_sum[0]"
store i1 %".add_0_sum[0]695", i1* %".comp_0_cmp[0]"
%".comp_0_cmp[0]696" = load i1, i1* %".comp_0_cmp[0]"
store i1 %".comp_0_cmp[0]696", i1* %".math_0_bc[0]"
%".math_0_bc[0]697" = load i1, i1* %".math_0_bc[0]"
store i1 %".math_0_bc[0]697", i1* %"#131[0]"
%"#130[0]698" = load i1, i1* %"#130[0]"
%"#128[0]699" = load i1, i1* %"#128[0]"
%".mux_0_as[0]'svalue" = and i1 %"#128[0]699", %"#130[0]698"
store i1 %".mux_0_as[0]'svalue", i1* %".mux_0_as[0]"
%"#132[0]700" = load i1, i1* %"#132[0]"
%"#131[0]701" = load i1, i1* %"#131[0]"
%".mux_0_bs[0]'svalue" = and i1 %"#131[0]701", %"#132[0]700"
store i1 %".mux_0_bs[0]'svalue", i1* %".mux_0_bs[0]"
%".mux_0_bs[0]702" = load i1, i1* %".mux_0_bs[0]"
store i1 %".mux_0_bs[0]702", i1* %"#134[0]"
%".mux_0_as[0]703" = load i1, i1* %".mux_0_as[0]"
store i1 %".mux_0_as[0]703", i1* %"#133[0]"
%"#137[0]704" = load i1, i1* %"#137[0]"
%"#135[0]705" = load i1, i1* %"#135[0]"
%".mux_0_as[1]'svalue" = and i1 %"#135[0]705", %"#137[0]704"
store i1 %".mux_0_as[1]'svalue", i1* %".mux_0_as[1]"
%".mux_0_as[1]706" = load i1, i1* %".mux_0_as[1]"
store i1 %".mux_0_as[1]706", i1* %"#140[0]"
%"#144[0]707" = load i1, i1* %"#144[0]"
%"#142[0]708" = load i1, i1* %"#142[0]"
%".mux_0_as[2]'svalue" = and i1 %"#142[0]708", %"#144[0]707"
store i1 %".mux_0_as[2]'svalue", i1* %".mux_0_as[2]"
%".mux_0_as[2]709" = load i1, i1* %".mux_0_as[2]"
store i1 %".mux_0_as[2]709", i1* %"#147[0]"
%"#151[0]710" = load i1, i1* %"#151[0]"
%"#149[0]711" = load i1, i1* %"#149[0]"
%".mux_0_as[3]'svalue" = and i1 %"#149[0]711", %"#151[0]710"
store i1 %".mux_0_as[3]'svalue", i1* %".mux_0_as[3]"
%".mux_0_as[3]712" = load i1, i1* %".mux_0_as[3]"
store i1 %".mux_0_as[3]712", i1* %"#154[0]"
%"#158[0]713" = load i1, i1* %"#158[0]"
%"#156[0]714" = load i1, i1* %"#156[0]"
%".mux_0_as[4]'svalue" = and i1 %"#156[0]714", %"#158[0]713"
```

```
store i1 %".mux_0_as[4]'svalue", i1* %".mux_0_as[4]"
%".mux_0_as[4]715" = load i1, i1* %".mux_0_as[4]"
store i1 %".mux_0_as[4]715", i1* %"#161[0]"
%"#165[0]716" = load i1, i1* %"#165[0]"
%"#163[0]717" = load i1, i1* %"#163[0]"
%".mux_0_as[5]'svalue" = and i1 %"#163[0]717", %"#165[0]716"
store i1 %".mux_0_as[5]'svalue", i1* %".mux_0_as[5]"
%".mux_0_as[5]718" = load i1, i1* %".mux_0_as[5]"
store i1 %".mux_0_as[5]718", i1* %"#168[0]"
%"#172[0]719" = load i1, i1* %"#172[0]"
%"#170[0]720" = load i1, i1* %"#170[0]"
%".mux_0_as[6]'svalue" = and i1 %"#170[0]720", %"#172[0]719"
store i1 %".mux_0_as[6]'svalue", i1* %".mux_0_as[6]"
%".mux_0_as[6]721" = load i1, i1* %".mux_0_as[6]"
store i1 %".mux_0_as[6]721", i1* %"#175[0]"
%"#179[0]722" = load i1, i1* %"#179[0]"
%"#177[0]723" = load i1, i1* %"#177[0]"
%".mux_0_as[7]'svalue" = and i1 %"#177[0]723", %"#179[0]722"
store i1 %".mux_0_as[7]'svalue", i1* %".mux_0_as[7]"
%".mux_0_as[7]724" = load i1, i1* %".mux_0_as[7]"
store i1 %".mux_0_as[7]724", i1* %"#182[0]"
%"#134[0]725" = load i1, i1* %"#134[0]"
%"#133[0]726" = load i1, i1* %"#133[0]"
%".mux_0_o[0]'svalue" = or i1 %"#133[0]726", %"#134[0]725"
store i1 %".mux_0_o[0]'svalue", i1* %".mux_0_o[0]"
%".mux_0_o[0]727" = load i1, i1* %".mux_0_o[0]"
store i1 %".mux_0_o[0]727", i1* %".math_0_bnew[0]"
%".math_0_bnew[0]728" = load i1, i1* %".math_0_bnew[0]"
store i1 %".math_0_bnew[0]728", i1* %"#198[0]"
%"#198[0]729" = load i1, i1* %"#198[0]"
%"#197[0]730" = load i1, i1* %"#197[0]"
%"#199[0]'svalue" = and i1 %"#197[0]730", %"#198[0]729"
store i1 %"#199[0]'svalue", i1* %"#199[0]"
%".math_0_bnew[0]731" = load i1, i1* %".math_0_bnew[0]"
store i1 %".math_0_bnew[0]731", i1* %"#193[0]"
%".math_0_bnew[0]732" = load i1, i1* %".math_0_bnew[0]"
store i1 %".math_0_bnew[0]732", i1* %"#190[0]"
%".math_0_bnew[0]733" = load i1, i1* %".math_0_bnew[0]"
store i1 %".math_0_bnew[0]733", i1* %"#185[0]"
%".add_0_c[7]734" = load i1, i1* %".add_0_c[7]"
store i1 %".add_0_c[7]734", i1* %"#122[0]"
%".add_0_c[7]735" = load i1, i1* %".add_0_c[7]"
store i1 %".add_0_c[7]735", i1* %"#115[0]"
```

```
%".add_0_c[7]736" = load i1, i1* %".add_0_c[7]"
store i1 %".add_0_c[7]736", i1* %"#114[0]"
%".add_0_c[6]737" = load i1, i1* %".add_0_c[6]"
store i1 %".add_0_c[6]737", i1* %"#106[0]"
%".add_0_c[6]738" = load i1, i1* %".add_0_c[6]"
store i1 %".add_0_c[6]738", i1* %"#99[0]"
%".add_0_c[6]739" = load i1, i1* %".add_0_c[6]"
store i1 %".add_0_c[6]739", i1* %"#98[0]"
%".add_0_c[5]740" = load i1, i1* %".add_0_c[5]"
store i1 %".add_0_c[5]740", i1* %"#90[0]"
%".add_0_c[5]741" = load i1, i1* %".add_0_c[5]"
store i1 %".add_0_c[5]741", i1* %"#83[0]"
%".add_0_c[5]742" = load i1, i1* %".add_0_c[5]"
store i1 %".add_0_c[5]742", i1* %"#82[0]"
%".add_0_c[4]743" = load i1, i1* %".add_0_c[4]"
store i1 %".add_0_c[4]743", i1* %"#74[0]"
%".add_0_c[4]744" = load i1, i1* %".add_0_c[4]"
store i1 %".add_0_c[4]744", i1* %"#67[0]"
%".add_0_c[4]745" = load i1, i1* %".add_0_c[4]"
store i1 %".add_0_c[4]745", i1* %"#66[0]"
%".add_0_c[3]746" = load i1, i1* %".add_0_c[3]"
store i1 %".add_0_c[3]746", i1* %"#58[0]"
%".add_0_c[3]747" = load i1, i1* %".add_0_c[3]"
store i1 %".add_0_c[3]747", i1* %"#51[0]"
%".add_0_c[3]748" = load i1, i1* %".add_0_c[3]"
store i1 %".add_0_c[3]748", i1* %"#50[0]"
%".add_0_c[2]749" = load i1, i1* %".add_0_c[2]"
store i1 %".add_0_c[2]749", i1* %"#42[0]"
%".add_0_c[2]750" = load i1, i1* %".add_0_c[2]"
store i1 %".add_0_c[2]750", i1* %"#35[0]"
%".add_0_c[2]751" = load i1, i1* %".add_0_c[2]"
store i1 %".add_0_c[2]751", i1* %"#34[0]"
%".add_0_c[1]752" = load i1, i1* %".add_0_c[1]"
store i1 %".add_0_c[1]752", i1* %"#26[0]"
%".add_0_c[1]753" = load i1, i1* %".add_0_c[1]"
store i1 %".add_0_c[1]753", i1* %"#19[0]"
%".add_0_c[1]754" = load i1, i1* %".add_0_c[1]"
store i1 %".add_0_c[1]754", i1* %"#18[0]"
%"#10[0]755" = load i1, i1* %"#10[0]"
%".fA_1_axb[0]756" = load i1, i1* %".fA_1_axb[0]"
%".fA_1_sum[0]'svalue" = xor i1 %".fA_1_axb[0]756", %"#10[0]755"
store i1 %".fA_1_sum[0]'svalue", i1* %".fA_1_sum[0]"
%"#19[0]757" = load i1, i1* %"#19[0]"
```

```
%".fA_2_axb[0]758" = load i1, i1* %".fA_2_axb[0]"
%"#20[0]'svalue" = and i1 %".fA_2_axb[0]758", %"#19[0]757"
store i1 %"#20[0]'svalue", i1* %"#20[0]"
%"#23[0]759" = load i1, i1* %"#23[0]"
%"#20[0]760" = load i1, i1* %"#20[0]"
%".fA_2_carry[0]'svalue" = or i1 %"#20[0]760", %"#23[0]759"
store i1 %".fA_2_carry[0]'svalue", i1* %".fA_2_carry[0]"
%"#18[0]761" = load i1, i1* %"#18[0]"
%".fA_2_axb[0]762" = load i1, i1* %".fA_2_axb[0]"
%".fA_2_sum[0]'svalue" = xor i1 %".fA_2_axb[0]762", %"#18[0]761"
store i1 %".fA_2_sum[0]'svalue", i1* %".fA_2_sum[0]"
%"#26[0]763" = load i1, i1* %"#26[0]"
%".fA_3_axb[0]764" = load i1, i1* %".fA_3_axb[0]"
%".fA_3_sum[0]'svalue" = xor i1 %".fA_3_axb[0]764", %"#26[0]763"
store i1 %".fA_3_sum[0]'svalue", i1* %".fA_3_sum[0]"
%"#35[0]765" = load i1, i1* %"#35[0]"
%".fA_4_axb[0]766" = load i1, i1* %".fA_4_axb[0]"
%"#36[0]'svalue" = and i1 %".fA_4_axb[0]766", %"#35[0]765"
store i1 %"#36[0]'svalue", i1* %"#36[0]"
%"#39[0]767" = load i1, i1* %"#39[0]"
%"#36[0]768" = load i1, i1* %"#36[0]"
%".fA_4_carry[0]'svalue" = or i1 %"#36[0]768", %"#39[0]767"
store i1 %".fA_4_carry[0]'svalue", i1* %".fA_4_carry[0]"
%"#34[0]769" = load i1, i1* %"#34[0]"
%".fA_4_axb[0]770" = load i1, i1* %".fA_4_axb[0]"
%".fA_4_sum[0]'svalue" = xor i1 %".fA_4_axb[0]770", %"#34[0]769"
store i1 %".fA_4_sum[0]'svalue", i1* %".fA_4_sum[0]"
%"#42[0]771" = load i1, i1* %"#42[0]"
%".fA_5_axb[0]772" = load i1, i1* %".fA_5_axb[0]"
%".fA_5_sum[0]'svalue" = xor i1 %".fA_5_axb[0]772", %"#42[0]771"
store i1 %".fA_5_sum[0]'svalue", i1* %".fA_5_sum[0]"
%"#51[0]773" = load i1, i1* %"#51[0]"
%".fA_6_axb[0]774" = load i1, i1* %".fA_6_axb[0]"
%"#52[0]'svalue" = and i1 %".fA_6_axb[0]774", %"#51[0]773"
store i1 %"#52[0]'svalue", i1* %"#52[0]"
%"#55[0]775" = load i1, i1* %"#55[0]"
%"#52[0]776" = load i1, i1* %"#52[0]"
%".fA_6_carry[0]'svalue" = or i1 %"#52[0]776", %"#55[0]775"
store i1 %".fA_6_carry[0]'svalue", i1* %".fA_6_carry[0]"
%"#50[0]777" = load i1, i1* %"#50[0]"
%".fA_6_axb[0]778" = load i1, i1* %".fA_6_axb[0]"
%".fA_6_sum[0]'svalue" = xor i1 %".fA_6_axb[0]778", %"#50[0]777"
store i1 %".fA_6_sum[0]'svalue", i1* %".fA_6_sum[0]"
```

```
%"#58[0]779" = load i1, i1* %"#58[0]"
%".fA_7_axb[0]780" = load i1, i1* %".fA_7_axb[0]"
%".fA_7_sum[0]'svalue" = xor i1 %".fA_7_axb[0]780", %"#58[0]779"
store i1 %".fA_7_sum[0]'svalue", i1* %".fA_7_sum[0]"
%"#67[0]781" = load i1, i1* %"#67[0]"
%".fA_8_axb[0]782" = load i1, i1* %".fA_8_axb[0]"
%"#68[0]'svalue" = and i1 %".fA_8_axb[0]782", %"#67[0]781"
store i1 %"#68[0]'svalue", i1* %"#68[0]"
%"#71[0]783" = load i1, i1* %"#71[0]"
%"#68[0]784" = load i1, i1* %"#68[0]"
%".fA_8_carry[0]'svalue" = or i1 %"#68[0]784", %"#71[0]783"
store i1 %".fA_8_carry[0]'svalue", i1* %".fA_8_carry[0]"
%"#66[0]785" = load i1, i1* %"#66[0]"
%".fA_8_axb[0]786" = load i1, i1* %".fA_8_axb[0]"
%".fA_8_sum[0]'svalue" = xor i1 %".fA_8_axb[0]786", %"#66[0]785"
store i1 %".fA_8_sum[0]'svalue", i1* %".fA_8_sum[0]"
%"#74[0]787" = load i1, i1* %"#74[0]"
%".fA_9_axb[0]788" = load i1, i1* %".fA_9_axb[0]"
%".fA_9_sum[0]'svalue" = xor i1 %".fA_9_axb[0]788", %"#74[0]787"
store i1 %".fA_9_sum[0]'svalue", i1* %".fA_9_sum[0]"
%"#83[0]789" = load i1, i1* %"#83[0]"
%".fA_10_axb[0]790" = load i1, i1* %".fA_10_axb[0]"
%"#84[0]'svalue" = and i1 %".fA_10_axb[0]790", %"#83[0]789"
store i1 %"#84[0]'svalue", i1* %"#84[0]"
%"#87[0]791" = load i1, i1* %"#87[0]"
%"#84[0]792" = load i1, i1* %"#84[0]"
%".fA_10_carry[0]'svalue" = or i1 %"#84[0]792", %"#87[0]791"
store i1 %".fA_10_carry[0]'svalue", i1* %".fA_10_carry[0]"
%"#82[0]793" = load i1, i1* %"#82[0]"
%".fA_10_axb[0]794" = load i1, i1* %".fA_10_axb[0]"
%".fA_10_sum[0]'svalue" = xor i1 %".fA_10_axb[0]794", %"#82[0]793"
store i1 %".fA_10_sum[0]'svalue", i1* %".fA_10_sum[0]"
%"#90[0]795" = load i1, i1* %"#90[0]"
%".fA_11_axb[0]796" = load i1, i1* %".fA_11_axb[0]"
%".fA_11_sum[0]'svalue" = xor i1 %".fA_11_axb[0]796", %"#90[0]795"
store i1 %".fA_11_sum[0]'svalue", i1* %".fA_11_sum[0]"
%"#99[0]797" = load i1, i1* %"#99[0]"
%".fA_12_axb[0]798" = load i1, i1* %".fA_12_axb[0]"
%"#100[0]'svalue" = and i1 %".fA_12_axb[0]798", %"#99[0]797"
store i1 %"#100[0]'svalue", i1* %"#100[0]"
%"#103[0]799" = load i1, i1* %"#103[0]"
%"#100[0]800" = load i1, i1* %"#100[0]"
%".fA_12_carry[0]'svalue" = or i1 %"#100[0]800", %"#103[0]799"
```

```
store i1 %".fA_12_carry[0]'svalue", i1* %".fA_12_carry[0]"
%"#98[0]801" = load i1, i1* %"#98[0]"
%".fA_12_axb[0]802" = load i1, i1* %".fA_12_axb[0]"
%".fA_12_sum[0]'svalue" = xor i1 %".fA_12_axb[0]802", %"#98[0]801"
store i1 %".fA_12_sum[0]'svalue", i1* %".fA_12_sum[0]"
%"#106[0]803" = load i1, i1* %"#106[0]"
%".fA_13_axb[0]804" = load i1, i1* %".fA_13_axb[0]"
%".fA_13_sum[0]'svalue" = xor i1 %".fA_13_axb[0]804", %"#106[0]803"
store i1 %".fA_13_sum[0]'svalue", i1* %".fA_13_sum[0]"
%"#115[0]805" = load i1, i1* %"#115[0]"
%".fA_14_axb[0]806" = load i1, i1* %".fA_14_axb[0]"
%"#116[0]'svalue" = and i1 %".fA_14_axb[0]806", %"#115[0]805"
store i1 %"#116[0]'svalue", i1* %"#116[0]"
%"#119[0]807" = load i1, i1* %"#119[0]"
%"#116[0]808" = load i1, i1* %"#116[0]"
%".fA_14_carry[0]'svalue" = or i1 %"#116[0]808", %"#119[0]807"
store i1 %".fA_14_carry[0]'svalue", i1* %".fA_14_carry[0]"
%"#114[0]809" = load i1, i1* %"#114[0]"
%".fA_14_axb[0]810" = load i1, i1* %".fA_14_axb[0]"
%".fA_14_sum[0]'svalue" = xor i1 %".fA_14_axb[0]810", %"#114[0]809"
store i1 %".fA_14_sum[0]'svalue", i1* %".fA_14_sum[0]"
%"#122[0]811" = load i1, i1* %"#122[0]"
%".fA_15_axb[0]812" = load i1, i1* %".fA_15_axb[0]"
%".fA_15_sum[0]'svalue" = xor i1 %".fA_15_axb[0]812", %"#122[0]811"
store i1 %".fA_15_sum[0]'svalue", i1* %".fA_15_sum[0]"
%".fA_2_sum[0]813" = load i1, i1* %".fA_2_sum[0]"
store i1 %".fA_2_sum[0]813", i1* %".add_0_sum[1]"
%".fA_4_sum[0]814" = load i1, i1* %".fA_4_sum[0]"
store i1 %".fA_4_sum[0]814", i1* %".add_0_sum[2]"
%".fA_6_sum[0]815" = load i1, i1* %".fA_6_sum[0]"
store i1 %".fA_6_sum[0]815", i1* %".add_0_sum[3]"
%".fA_8_sum[0]816" = load i1, i1* %".fA_8_sum[0]"
store i1 %".fA_8_sum[0]816", i1* %".add_0_sum[4]"
%".fA_10_sum[0]817" = load i1, i1* %".fA_10_sum[0]"
store i1 %".fA_10_sum[0]817", i1* %".add_0_sum[5]"
%".fA_12_sum[0]818" = load i1, i1* %".fA_12_sum[0]"
store i1 %".fA_12_sum[0]818", i1* %".add_0_sum[6]"
%".fA_14_sum[0]819" = load i1, i1* %".fA_14_sum[0]"
store i1 %".fA_14_sum[0]819", i1* %".add_0_sum[7]"
%".add_0_sum[1]820" = load i1, i1* %".add_0_sum[1]"
store i1 %".add_0_sum[1]820", i1* %".comp_0_cmp[1]"
%".add_0_sum[2]821" = load i1, i1* %".add_0_sum[2]"
store i1 %".add_0_sum[2]821", i1* %".comp_0_cmp[2]"
```

```
%".add_0_sum[3]822" = load i1, i1* %".add_0_sum[3]"
store i1 %".add_0_sum[3]822", i1* %".comp_0_cmp[3]"
%".add_0_sum[4]823" = load i1, i1* %".add_0_sum[4]"
store i1 %".add_0_sum[4]823", i1* %".comp_0_cmp[4]"
%".add_0_sum[5]824" = load i1, i1* %".add_0_sum[5]"
store i1 %".add_0_sum[5]824", i1* %".comp_0_cmp[5]"
%".add_0_sum[6]825" = load i1, i1* %".add_0_sum[6]"
store i1 %".add_0_sum[6]825", i1* %".comp_0_cmp[6]"
%".add_0_sum[7]826" = load i1, i1* %".add_0_sum[7]"
store i1 %".add_0_sum[7]826", i1* %".comp_0_cmp[7]"
%".comp_0_cmp[1]827" = load i1, i1* %".comp_0_cmp[1]"
store i1 %".comp_0_cmp[1]827", i1* %".math_0_bc[1]"
%".math_0_bc[1]828" = load i1, i1* %".math_0_bc[1]"
store i1 %".math_0_bc[1]828", i1* %"#138[0]"
%"#139[0]829" = load i1, i1* %"#139[0]"
%"#138[0]830" = load i1, i1* %"#138[0]"
%".mux_0_bs[1]'svalue" = and i1 %"#138[0]830", %"#139[0]829"
store i1 %".mux_0_bs[1]'svalue", i1* %".mux_0_bs[1]"
%".mux_0_bs[1]831" = load i1, i1* %".mux_0_bs[1]"
store i1 %".mux_0_bs[1]831", i1* %"#141[0]"
%".comp_0_cmp[2]832" = load i1, i1* %".comp_0_cmp[2]"
store i1 %".comp_0_cmp[2]832", i1* %".math_0_bc[2]"
%".math_0_bc[2]833" = load i1, i1* %".math_0_bc[2]"
store i1 %".math_0_bc[2]833", i1* %"#145[0]"
%"#146[0]834" = load i1, i1* %"#146[0]"
%"#145[0]835" = load i1, i1* %"#145[0]"
%".mux_0_bs[2]'svalue" = and i1 %"#145[0]835", %"#146[0]834"
store i1 %".mux_0_bs[2]'svalue", i1* %".mux_0_bs[2]"
%".mux_0_bs[2]836" = load i1, i1* %".mux_0_bs[2]"
store i1 %".mux_0_bs[2]836", i1* %"#148[0]"
%".comp_0_cmp[3]837" = load i1, i1* %".comp_0_cmp[3]"
store i1 %".comp_0_cmp[3]837", i1* %".math_0_bc[3]"
%".math_0_bc[3]838" = load i1, i1* %".math_0_bc[3]"
store i1 %".math_0_bc[3]838", i1* %"#152[0]"
%"#153[0]839" = load i1, i1* %"#153[0]"
%"#152[0]840" = load i1, i1* %"#152[0]"
%".mux_0_bs[3]'svalue" = and i1 %"#152[0]840", %"#153[0]839"
store i1 %".mux_0_bs[3]'svalue", i1* %".mux_0_bs[3]"
%".mux_0_bs[3]841" = load i1, i1* %".mux_0_bs[3]"
store i1 %".mux_0_bs[3]841", i1* %"#155[0]"
%".comp_0_cmp[4]842" = load i1, i1* %".comp_0_cmp[4]"
store i1 %".comp_0_cmp[4]842", i1* %".math_0_bc[4]"
%".math_0_bc[4]843" = load i1, i1* %".math_0_bc[4]"
```

```
store i1 %".math_0_bc[4]843", i1* %"#159[0]"
%"#160[0]844" = load i1, i1* %"#160[0]"
%"#159[0]845" = load i1, i1* %"#159[0]"
%".mux_0_bs[4]'svalue" = and i1 %"#159[0]845", %"#160[0]844"
store i1 %".mux_0_bs[4]'svalue", i1* %".mux_0_bs[4]"
%".mux_0_bs[4]846" = load i1, i1* %".mux_0_bs[4]"
store i1 %".mux_0_bs[4]846", i1* %"#162[0]"
%".comp_0_cmp[5]847" = load i1, i1* %".comp_0_cmp[5]"
store i1 %".comp_0_cmp[5]847", i1* %".math_0_bc[5]"
%".math_0_bc[5]848" = load i1, i1* %".math_0_bc[5]"
store i1 %".math_0_bc[5]848", i1* %"#166[0]"
%"#167[0]849" = load i1, i1* %"#167[0]"
%"#166[0]850" = load i1, i1* %"#166[0]"
%".mux_0_bs[5]'svalue" = and i1 %"#166[0]850", %"#167[0]849"
store i1 %".mux_0_bs[5]'svalue", i1* %".mux_0_bs[5]"
%".mux_0_bs[5]851" = load i1, i1* %".mux_0_bs[5]"
store i1 %".mux_0_bs[5]851", i1* %"#169[0]"
%".comp_0_cmp[6]852" = load i1, i1* %".comp_0_cmp[6]"
store i1 %".comp_0_cmp[6]852", i1* %".math_0_bc[6]"
%".math_0_bc[6]853" = load i1, i1* %".math_0_bc[6]"
store i1 %".math_0_bc[6]853", i1* %"#173[0]"
%"#174[0]854" = load i1, i1* %"#174[0]"
%"#173[0]855" = load i1, i1* %"#173[0]"
%".mux_0_bs[6]'svalue" = and i1 %"#173[0]855", %"#174[0]854"
store i1 %".mux_0_bs[6]'svalue", i1* %".mux_0_bs[6]"
%".mux_0_bs[6]856" = load i1, i1* %".mux_0_bs[6]"
store i1 %".mux_0_bs[6]856", i1* %"#176[0]"
%".comp_0_cmp[7]857" = load i1, i1* %".comp_0_cmp[7]"
store i1 %".comp_0_cmp[7]857", i1* %".math_0_bc[7]"
%".math_0_bc[7]858" = load i1, i1* %".math_0_bc[7]"
store i1 %".math_0_bc[7]858", i1* %"#180[0]"
%"#181[0]859" = load i1, i1* %"#181[0]"
%"#180[0]860" = load i1, i1* %"#180[0]"
%".mux_0_bs[7]'svalue" = and i1 %"#180[0]860", %"#181[0]859"
store i1 %".mux_0_bs[7]'svalue", i1* %".mux_0_bs[7]"
%".mux_0_bs[7]861" = load i1, i1* %".mux_0_bs[7]"
store i1 %".mux_0_bs[7]861", i1* %"#183[0]"
%"#141[0]862" = load i1, i1* %"#141[0]"
%"#140[0]863" = load i1, i1* %"#140[0]"
%".mux_0_o[1]'svalue" = or i1 %"#140[0]863", %"#141[0]862"
store i1 %".mux_0_o[1]'svalue", i1* %".mux_0_o[1]"
%"#148[0]864" = load i1, i1* %"#148[0]"
%"#147[0]865" = load i1, i1* %"#147[0]"
```

```
%".mux_0_o[2]'svalue" = or i1 %"#147[0]865", %"#148[0]864"
store i1 %".mux_0_o[2]'svalue", i1* %".mux_0_o[2]"
%"#155[0]866" = load i1, i1* %"#155[0]"
%"#154[0]867" = load i1, i1* %"#154[0]"
%".mux_0_o[3]'svalue" = or i1 %"#154[0]867", %"#155[0]866"
store i1 %".mux_0_o[3]'svalue", i1* %".mux_0_o[3]"
%"#162[0]868" = load i1, i1* %"#162[0]"
%"#161[0]869" = load i1, i1* %"#161[0]"
%".mux_0_o[4]'svalue" = or i1 %"#161[0]869", %"#162[0]868"
store i1 %".mux_0_o[4]'svalue", i1* %".mux_0_o[4]"
%"#169[0]870" = load i1, i1* %"#169[0]"
%"#168[0]871" = load i1, i1* %"#168[0]"
%".mux_0_o[5]'svalue" = or i1 %"#168[0]871", %"#169[0]870"
store i1 %".mux_0_o[5]'svalue", i1* %".mux_0_o[5]"
%"#176[0]872" = load i1, i1* %"#176[0]"
%"#175[0]873" = load i1, i1* %"#175[0]"
%".mux_0_o[6]'svalue" = or i1 %"#175[0]873", %"#176[0]872"
store i1 %".mux_0_o[6]'svalue", i1* %".mux_0_o[6]"
%"#183[0]874" = load i1, i1* %"#183[0]"
%"#182[0]875" = load i1, i1* %"#182[0]"
%".mux_0_o[7]'svalue" = or i1 %"#182[0]875", %"#183[0]874"
store i1 %".mux_0_o[7]'svalue", i1* %".mux_0_o[7]"
%"#185[0]876" = load i1, i1* %"#185[0]"
%"#184[0]877" = load i1, i1* %"#184[0]"
%".fA_16_axb[0]'svalue" = xor i1 %"#184[0]877", %"#185[0]876"
store i1 %".fA_16_axb[0]'svalue", i1* %".fA_16_axb[0]"
%"#190[0]878" = load i1, i1* %"#190[0]"
%"#189[0]879" = load i1, i1* %"#189[0]"
%"#191[0]'svalue" = and i1 %"#189[0]879", %"#190[0]878"
store i1 %"#191[0]'svalue", i1* %"#191[0]"
%"#187[0]880" = load i1, i1* %"#187[0]"
%".fA_16_axb[0]881" = load i1, i1* %".fA_16_axb[0]"
%"#188[0]'svalue" = and i1 %".fA_16_axb[0]881", %"#187[0]880"
store i1 %"#188[0]'svalue", i1* %"#188[0]"
%"#191[0]882" = load i1, i1* %"#191[0]"
%"#188[0]883" = load i1, i1* %"#188[0]"
%".fA_16_carry[0]'svalue" = or i1 %"#188[0]883", %"#191[0]882"
store i1 %".fA_16_carry[0]'svalue", i1* %".fA_16_carry[0]"
%"#186[0]884" = load i1, i1* %"#186[0]"
%".fA_16_axb[0]885" = load i1, i1* %".fA_16_axb[0]"
%".fA_16_sum[0]'svalue" = xor i1 %".fA_16_axb[0]885", %"#186[0]884"
store i1 %".fA_16_sum[0]'svalue", i1* %".fA_16_sum[0]"
%"#193[0]886" = load i1, i1* %"#193[0]"
```

```
%"#192[0]887" = load i1, i1* %"#192[0]"
%".fA_17_axb[0]'svalue" = xor i1 %"#192[0]887", %"#193[0]886"
store i1 %".fA_17_axb[0]'svalue", i1* %".fA_17_axb[0]"
%"#195[0]888" = load i1, i1* %"#195[0]"
%".fA_17_axb[0]889" = load i1, i1* %".fA_17_axb[0]"
%"#196[0]'svalue" = and i1 %".fA_17_axb[0]889", %"#195[0]888"
store i1 %"#196[0]'svalue", i1* %"#196[0]"
%"#199[0]890" = load i1, i1* %"#199[0]"
%"#196[0]891" = load i1, i1* %"#196[0]"
%".fA_17_carry[0]'svalue" = or i1 %"#196[0]891", %"#199[0]890"
store i1 %".fA_17_carry[0]'svalue", i1* %".fA_17_carry[0]"
%".fA_17_carry[0]892" = load i1, i1* %".fA_17_carry[0]"
store i1 %".fA_17_carry[0]892", i1* %".add_1_c[1]"
%".add_1_c[1]893" = load i1, i1* %".add_1_c[1]"
store i1 %".add_1_c[1]893", i1* %"#211[0]"
%".mux_0_o[1]894" = load i1, i1* %".mux_0_o[1]"
store i1 %".mux_0_o[1]894", i1* %".math_0_bnew[1]"
%".math_0_bnew[1]895" = load i1, i1* %".math_0_bnew[1]"
store i1 %".math_0_bnew[1]895", i1* %"#214[0]"
%"#214[0]896" = load i1, i1* %"#214[0]"
%"#213[0]897" = load i1, i1* %"#213[0]"
%"#215[0]'svalue" = and i1 %"#213[0]897", %"#214[0]896"
store i1 %"#215[0]'svalue", i1* %"#215[0]"
%".mux_0_o[2]898" = load i1, i1* %".mux_0_o[2]"
store i1 %".mux_0_o[2]898", i1* %".math_0_bnew[2]"
%".math_0_bnew[2]899" = load i1, i1* %".math_0_bnew[2]"
store i1 %".math_0_bnew[2]899", i1* %"#230[0]"
%"#230[0]900" = load i1, i1* %"#230[0]"
%"#229[0]901" = load i1, i1* %"#229[0]"
%"#231[0]'svalue" = and i1 %"#229[0]901", %"#230[0]900"
store i1 %"#231[0]'svalue", i1* %"#231[0]"
%".mux_0_o[3]902" = load i1, i1* %".mux_0_o[3]"
store i1 %".mux_0_o[3]902", i1* %".math_0_bnew[3]"
%".math_0_bnew[3]903" = load i1, i1* %".math_0_bnew[3]"
store i1 %".math_0_bnew[3]903", i1* %"#246[0]"
%"#246[0]904" = load i1, i1* %"#246[0]"
%"#245[0]905" = load i1, i1* %"#245[0]"
%"#247[0]'svalue" = and i1 %"#245[0]905", %"#246[0]904"
store i1 %"#247[0]'svalue", i1* %"#247[0]"
%".mux_0_o[4]906" = load i1, i1* %".mux_0_o[4]"
store i1 %".mux_0_o[4]906", i1* %".math_0_bnew[4]"
%".math_0_bnew[4]907" = load i1, i1* %".math_0_bnew[4]"
store i1 %".math_0_bnew[4]907", i1* %"#262[0]"
```

```
%"#262[0]908" = load i1, i1* %"#262[0]"
%"#261[0]909" = load i1, i1* %"#261[0]"
%"#263[0]'svalue" = and i1 %"#261[0]909", %"#262[0]908"
store i1 %"#263[0]'svalue", i1* %"#263[0]"
%".mux_0_o[5]910" = load i1, i1* %".mux_0_o[5]"
store i1 %".mux_0_o[5]910", i1* %".math_0_bnew[5]"
%".math_0_bnew[5]911" = load i1, i1* %".math_0_bnew[5]"
store i1 %".math_0_bnew[5]911", i1* %"#278[0]"
%"#278[0]912" = load i1, i1* %"#278[0]"
%"#277[0]913" = load i1, i1* %"#277[0]"
%"#279[0]'svalue" = and i1 %"#277[0]913", %"#278[0]912"
store i1 %"#279[0]'svalue", i1* %"#279[0]"
%".mux_0_o[6]914" = load i1, i1* %".mux_0_o[6]"
store i1 %".mux_0_o[6]914", i1* %".math_0_bnew[6]"
%".math_0_bnew[6]915" = load i1, i1* %".math_0_bnew[6]"
store i1 %".math_0_bnew[6]915", i1* %"#294[0]"
%"#294[0]916" = load i1, i1* %"#294[0]"
%"#293[0]917" = load i1, i1* %"#293[0]"
%"#295[0]'svalue" = and i1 %"#293[0]917", %"#294[0]916"
store i1 %"#295[0]'svalue", i1* %"#295[0]"
%".mux_0_o[7]918" = load i1, i1* %".mux_0_o[7]"
store i1 %".mux_0_o[7]918", i1* %".math_0_bnew[7]"
%".math_0_bnew[7]919" = load i1, i1* %".math_0_bnew[7]"
store i1 %".math_0_bnew[7]919", i1* %"#310[0]"
%"#310[0]920" = load i1, i1* %"#310[0]"
%"#309[0]921" = load i1, i1* %"#309[0]"
%"#311[0]'svalue" = and i1 %"#309[0]921", %"#310[0]920"
store i1 %"#311[0]'svalue", i1* %"#311[0]"
%".fA_16_sum[0]922" = load i1, i1* %".fA_16_sum[0]"
store i1 %".fA_16_sum[0]922", i1* %".add_1_sum[0]"
%".add_1_sum[0]923" = load i1, i1* %".add_1_sum[0]"
store i1 %".add_1_sum[0]923", i1* %".math_0_sum[0]"
%".math_0_sum[0]924" = load i1, i1* %".math_0_sum[0]"
store i1 %".math_0_sum[0]924", i1* %".alu_0_m[0]"
%".alu_0_m[0]925" = load i1, i1* %".alu_0_m[0]"
store i1 %".alu_0_m[0]925", i1* %"#528[0]"
%"#530[0]926" = load i1, i1* %"#530[0]"
%"#528[0]927" = load i1, i1* %"#528[0]"
%".mux_4_as[0]'svalue" = and i1 %"#528[0]927", %"#530[0]926"
store i1 %".mux_4_as[0]'svalue", i1* %".mux_4_as[0]"
%".mux_4_as[0]928" = load i1, i1* %".mux_4_as[0]"
store i1 %".mux_4_as[0]928", i1* %"#533[0]"
%"#534[0]929" = load i1, i1* %"#534[0]"
```

```
  %"#533[0]930" = load i1, i1* %"#533[0]"
  %".mux_4_o[0]'svalue" = or i1 %"#533[0]930", %"#534[0]929"
  store i1 %".mux_4_o[0]'svalue", i1* %".mux_4_o[0]"
  %".mux_4_o[0]931" = load i1, i1* %".mux_4_o[0]"
  store i1 %".mux_4_o[0]931", i1* %".alu_0_o[0]"
  %".alu_0_o[0]932" = load i1, i1* %".alu_0_o[0]"
  store i1 %".alu_0_o[0]932", i1* %x_0
  %x_0933 = load i1, i1* %x_0
  %printf = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt, i32 0, i32
0), i1 %x_0933)
  %".math_0_bnew[7]934" = load i1, i1* %".math_0_bnew[7]"
  store i1 %".math_0_bnew[7]934", i1* %"#305[0]"
  %".math_0_bnew[7]935" = load i1, i1* %".math_0_bnew[7]"
  store i1 %".math_0_bnew[7]935", i1* %"#302[0]"
  %".math_0_bnew[7]936" = load i1, i1* %".math_0_bnew[7]"
  store i1 %".math_0_bnew[7]936", i1* %"#297[0]"
  %".math_0_bnew[6]937" = load i1, i1* %".math_0_bnew[6]"
  store i1 %".math_0_bnew[6]937", i1* %"#289[0]"
  %".math_0_bnew[6]938" = load i1, i1* %".math_0_bnew[6]"
  store i1 %".math_0_bnew[6]938", i1* %"#286[0]"
  %".math_0_bnew[6]939" = load i1, i1* %".math_0_bnew[6]"
  store i1 %".math_0_bnew[6]939", i1* %"#281[0]"
  %".math_0_bnew[5]940" = load i1, i1* %".math_0_bnew[5]"
  store i1 %".math_0_bnew[5]940", i1* %"#273[0]"
  %".math_0_bnew[5]941" = load i1, i1* %".math_0_bnew[5]"
  store i1 %".math_0_bnew[5]941", i1* %"#270[0]"
  %".math_0_bnew[5]942" = load i1, i1* %".math_0_bnew[5]"
  store i1 %".math_0_bnew[5]942", i1* %"#265[0]"
  %".math_0_bnew[4]943" = load i1, i1* %".math_0_bnew[4]"
  store i1 %".math_0_bnew[4]943", i1* %"#257[0]"
  %".math_0_bnew[4]944" = load i1, i1* %".math_0_bnew[4]"
  store i1 %".math_0_bnew[4]944", i1* %"#254[0]"
  %".math_0_bnew[4]945" = load i1, i1* %".math_0_bnew[4]"
  store i1 %".math_0_bnew[4]945", i1* %"#249[0]"
  %".math_0_bnew[3]946" = load i1, i1* %".math_0_bnew[3]"
  store i1 %".math_0_bnew[3]946", i1* %"#241[0]"
  %".math_0_bnew[3]947" = load i1, i1* %".math_0_bnew[3]"
  store i1 %".math_0_bnew[3]947", i1* %"#238[0]"
  %".math_0_bnew[3]948" = load i1, i1* %".math_0_bnew[3]"
  store i1 %".math_0_bnew[3]948", i1* %"#233[0]"
  %".math_0_bnew[2]949" = load i1, i1* %".math_0_bnew[2]"
  store i1 %".math_0_bnew[2]949", i1* %"#225[0]"
  %".math_0_bnew[2]950" = load i1, i1* %".math_0_bnew[2]"
```

```
store i1 %".math_0_bnew[2]950", i1* %"#222[0]"
%".math_0_bnew[2]951" = load i1, i1* %".math_0_bnew[2]"
store i1 %".math_0_bnew[2]951", i1* %"#217[0]"
%".add_1_c[1]952" = load i1, i1* %".add_1_c[1]"
store i1 %".add_1_c[1]952", i1* %"#210[0]"
%".math_0_bnew[1]953" = load i1, i1* %".math_0_bnew[1]"
store i1 %".math_0_bnew[1]953", i1* %"#209[0]"
%".math_0_bnew[1]954" = load i1, i1* %".math_0_bnew[1]"
store i1 %".math_0_bnew[1]954", i1* %"#206[0]"
%".add_1_c[1]955" = load i1, i1* %".add_1_c[1]"
store i1 %".add_1_c[1]955", i1* %"#203[0]"
%".add_1_c[1]956" = load i1, i1* %".add_1_c[1]"
store i1 %".add_1_c[1]956", i1* %"#202[0]"
%".math_0_bnew[1]957" = load i1, i1* %".math_0_bnew[1]"
store i1 %".math_0_bnew[1]957", i1* %"#201[0]"
%"#194[0]958" = load i1, i1* %"#194[0]"
%".fA_17_axb[0]959" = load i1, i1* %".fA_17_axb[0]"
%".fA_17_sum[0]'svalue" = xor i1 %".fA_17_axb[0]959", %"#194[0]958"
store i1 %".fA_17_sum[0]'svalue", i1* %".fA_17_sum[0]"
%"#201[0]960" = load i1, i1* %"#201[0]"
%"#200[0]961" = load i1, i1* %"#200[0]"
%".fA_18_axb[0]'svalue" = xor i1 %"#200[0]961", %"#201[0]960"
store i1 %".fA_18_axb[0]'svalue", i1* %".fA_18_axb[0]"
%"#206[0]962" = load i1, i1* %"#206[0]"
%"#205[0]963" = load i1, i1* %"#205[0]"
%"#207[0]'svalue" = and i1 %"#205[0]963", %"#206[0]962"
store i1 %"#207[0]'svalue", i1* %"#207[0]"
%"#203[0]964" = load i1, i1* %"#203[0]"
%".fA_18_axb[0]965" = load i1, i1* %".fA_18_axb[0]"
%"#204[0]'svalue" = and i1 %".fA_18_axb[0]965", %"#203[0]964"
store i1 %"#204[0]'svalue", i1* %"#204[0]"
%"#207[0]966" = load i1, i1* %"#207[0]"
%"#204[0]967" = load i1, i1* %"#204[0]"
%".fA_18_carry[0]'svalue" = or i1 %"#204[0]967", %"#207[0]966"
store i1 %".fA_18_carry[0]'svalue", i1* %".fA_18_carry[0]"
%"#202[0]968" = load i1, i1* %"#202[0]"
%".fA_18_axb[0]969" = load i1, i1* %".fA_18_axb[0]"
%".fA_18_sum[0]'svalue" = xor i1 %".fA_18_axb[0]969", %"#202[0]968"
store i1 %".fA_18_sum[0]'svalue", i1* %".fA_18_sum[0]"
%"#209[0]970" = load i1, i1* %"#209[0]"
%"#208[0]971" = load i1, i1* %"#208[0]"
%".fA_19_axb[0]'svalue" = xor i1 %"#208[0]971", %"#209[0]970"
store i1 %".fA_19_axb[0]'svalue", i1* %".fA_19_axb[0]"
```

```
%"#211[0]972" = load i1, i1* %"#211[0]"
%".fA_19_axb[0]973" = load i1, i1* %".fA_19_axb[0]"
%"#212[0]'svalue" = and i1 %".fA_19_axb[0]973", %"#211[0]972"
store i1 %"#212[0]'svalue", i1* %"#212[0]"
%"#215[0]974" = load i1, i1* %"#215[0]"
%"#212[0]975" = load i1, i1* %"#212[0]"
%".fA_19_carry[0]'svalue" = or i1 %"#212[0]975", %"#215[0]974"
store i1 %".fA_19_carry[0]'svalue", i1* %".fA_19_carry[0]"
%"#217[0]976" = load i1, i1* %"#217[0]"
%"#216[0]977" = load i1, i1* %"#216[0]"
%".fA_20_axb[0]'svalue" = xor i1 %"#216[0]977", %"#217[0]976"
store i1 %".fA_20_axb[0]'svalue", i1* %".fA_20_axb[0]"
%"#222[0]978" = load i1, i1* %"#222[0]"
%"#221[0]979" = load i1, i1* %"#221[0]"
%"#223[0]'svalue" = and i1 %"#221[0]979", %"#222[0]978"
store i1 %"#223[0]'svalue", i1* %"#223[0]"
%".fA_19_carry[0]980" = load i1, i1* %".fA_19_carry[0]"
store i1 %".fA_19_carry[0]980", i1* %".add_1_c[2]"
%".add_1_c[2]981" = load i1, i1* %".add_1_c[2]"
store i1 %".add_1_c[2]981", i1* %"#227[0]"
%"#225[0]982" = load i1, i1* %"#225[0]"
%"#224[0]983" = load i1, i1* %"#224[0]"
%".fA_21_axb[0]'svalue" = xor i1 %"#224[0]983", %"#225[0]982"
store i1 %".fA_21_axb[0]'svalue", i1* %".fA_21_axb[0]"
%"#227[0]984" = load i1, i1* %"#227[0]"
%".fA_21_axb[0]985" = load i1, i1* %".fA_21_axb[0]"
%"#228[0]'svalue" = and i1 %".fA_21_axb[0]985", %"#227[0]984"
store i1 %"#228[0]'svalue", i1* %"#228[0]"
%"#231[0]986" = load i1, i1* %"#231[0]"
%"#228[0]987" = load i1, i1* %"#228[0]"
%".fA_21_carry[0]'svalue" = or i1 %"#228[0]987", %"#231[0]986"
store i1 %".fA_21_carry[0]'svalue", i1* %".fA_21_carry[0]"
%"#233[0]988" = load i1, i1* %"#233[0]"
%"#232[0]989" = load i1, i1* %"#232[0]"
%".fA_22_axb[0]'svalue" = xor i1 %"#232[0]989", %"#233[0]988"
store i1 %".fA_22_axb[0]'svalue", i1* %".fA_22_axb[0]"
%"#238[0]990" = load i1, i1* %"#238[0]"
%"#237[0]991" = load i1, i1* %"#237[0]"
%"#239[0]'svalue" = and i1 %"#237[0]991", %"#238[0]990"
store i1 %"#239[0]'svalue", i1* %"#239[0]"
%".fA_21_carry[0]992" = load i1, i1* %".fA_21_carry[0]"
store i1 %".fA_21_carry[0]992", i1* %".add_1_c[3]"
%".add_1_c[3]993" = load i1, i1* %".add_1_c[3]"
```

```
store i1 %".add_1_c[3]993", i1* %"#243[0]"
%"#241[0]994" = load i1, i1* %"#241[0]"
%"#240[0]995" = load i1, i1* %"#240[0]"
%".fA_23_axb[0]'svalue" = xor i1 %"#240[0]995", %"#241[0]994"
store i1 %".fA_23_axb[0]'svalue", i1* %".fA_23_axb[0]"
%"#243[0]996" = load i1, i1* %"#243[0]"
%".fA_23_axb[0]997" = load i1, i1* %".fA_23_axb[0]"
%"#244[0]'svalue" = and i1 %".fA_23_axb[0]997", %"#243[0]996"
store i1 %"#244[0]'svalue", i1* %"#244[0]"
%"#247[0]998" = load i1, i1* %"#247[0]"
%"#244[0]999" = load i1, i1* %"#244[0]"
%".fA_23_carry[0]'svalue" = or i1 %"#244[0]999", %"#247[0]998"
store i1 %".fA_23_carry[0]'svalue", i1* %".fA_23_carry[0]"
%"#249[0]1000" = load i1, i1* %"#249[0]"
%"#248[0]1001" = load i1, i1* %"#248[0]"
%".fA_24_axb[0]'svalue" = xor i1 %"#248[0]1001", %"#249[0]1000"
store i1 %".fA_24_axb[0]'svalue", i1* %".fA_24_axb[0]"
%"#254[0]1002" = load i1, i1* %"#254[0]"
%"#253[0]1003" = load i1, i1* %"#253[0]"
%"#255[0]'svalue" = and i1 %"#253[0]1003", %"#254[0]1002"
store i1 %"#255[0]'svalue", i1* %"#255[0]"
%".fA_23_carry[0]1004" = load i1, i1* %".fA_23_carry[0]"
store i1 %".fA_23_carry[0]1004", i1* %".add_1_c[4]"
%".add_1_c[4]1005" = load i1, i1* %".add_1_c[4]"
store i1 %".add_1_c[4]1005", i1* %"#259[0]"
%"#257[0]1006" = load i1, i1* %"#257[0]"
%"#256[0]1007" = load i1, i1* %"#256[0]"
%".fA_25_axb[0]'svalue" = xor i1 %"#256[0]1007", %"#257[0]1006"
store i1 %".fA_25_axb[0]'svalue", i1* %".fA_25_axb[0]"
%"#259[0]1008" = load i1, i1* %"#259[0]"
%".fA_25_axb[0]1009" = load i1, i1* %".fA_25_axb[0]"
%"#260[0]'svalue" = and i1 %".fA_25_axb[0]1009", %"#259[0]1008"
store i1 %"#260[0]'svalue", i1* %"#260[0]"
%"#263[0]1010" = load i1, i1* %"#263[0]"
%"#260[0]1011" = load i1, i1* %"#260[0]"
%".fA_25_carry[0]'svalue" = or i1 %"#260[0]1011", %"#263[0]1010"
store i1 %".fA_25_carry[0]'svalue", i1* %".fA_25_carry[0]"
%"#265[0]1012" = load i1, i1* %"#265[0]"
%"#264[0]1013" = load i1, i1* %"#264[0]"
%".fA_26_axb[0]'svalue" = xor i1 %"#264[0]1013", %"#265[0]1012"
store i1 %".fA_26_axb[0]'svalue", i1* %".fA_26_axb[0]"
%"#270[0]1014" = load i1, i1* %"#270[0]"
%"#269[0]1015" = load i1, i1* %"#269[0]"
```

```
%"#271[0]'svalue" = and i1 %"#269[0]1015", %"#270[0]1014"
store i1 %"#271[0]'svalue", i1* %"#271[0]"
%".fA_25_carry[0]1016" = load i1, i1* %".fA_25_carry[0]"
store i1 %".fA_25_carry[0]1016", i1* %".add_1_c[5]"
%".add_1_c[5]1017" = load i1, i1* %".add_1_c[5]"
store i1 %".add_1_c[5]1017", i1* %"#275[0]"
%"#273[0]1018" = load i1, i1* %"#273[0]"
%"#272[0]1019" = load i1, i1* %"#272[0]"
%".fA_27_axb[0]'svalue" = xor i1 %"#272[0]1019", %"#273[0]1018"
store i1 %".fA_27_axb[0]'svalue", i1* %".fA_27_axb[0]"
%"#275[0]1020" = load i1, i1* %"#275[0]"
%".fA_27_axb[0]1021" = load i1, i1* %".fA_27_axb[0]"
%"#276[0]'svalue" = and i1 %".fA_27_axb[0]1021", %"#275[0]1020"
store i1 %"#276[0]'svalue", i1* %"#276[0]"
%"#279[0]1022" = load i1, i1* %"#279[0]"
%"#276[0]1023" = load i1, i1* %"#276[0]"
%".fA_27_carry[0]'svalue" = or i1 %"#276[0]1023", %"#279[0]1022"
store i1 %".fA_27_carry[0]'svalue", i1* %".fA_27_carry[0]"
%"#281[0]1024" = load i1, i1* %"#281[0]"
%"#280[0]1025" = load i1, i1* %"#280[0]"
%".fA_28_axb[0]'svalue" = xor i1 %"#280[0]1025", %"#281[0]1024"
store i1 %".fA_28_axb[0]'svalue", i1* %".fA_28_axb[0]"
%"#286[0]1026" = load i1, i1* %"#286[0]"
%"#285[0]1027" = load i1, i1* %"#285[0]"
%"#287[0]'svalue" = and i1 %"#285[0]1027", %"#286[0]1026"
store i1 %"#287[0]'svalue", i1* %"#287[0]"
%".fA_27_carry[0]1028" = load i1, i1* %".fA_27_carry[0]"
store i1 %".fA_27_carry[0]1028", i1* %".add_1_c[6]"
%".add_1_c[6]1029" = load i1, i1* %".add_1_c[6]"
store i1 %".add_1_c[6]1029", i1* %"#291[0]"
%"#289[0]1030" = load i1, i1* %"#289[0]"
%"#288[0]1031" = load i1, i1* %"#288[0]"
%".fA_29_axb[0]'svalue" = xor i1 %"#288[0]1031", %"#289[0]1030"
store i1 %".fA_29_axb[0]'svalue", i1* %".fA_29_axb[0]"
%"#291[0]1032" = load i1, i1* %"#291[0]"
%".fA_29_axb[0]1033" = load i1, i1* %".fA_29_axb[0]"
%"#292[0]'svalue" = and i1 %".fA_29_axb[0]1033", %"#291[0]1032"
store i1 %"#292[0]'svalue", i1* %"#292[0]"
%"#295[0]1034" = load i1, i1* %"#295[0]"
%"#292[0]1035" = load i1, i1* %"#292[0]"
%".fA_29_carry[0]'svalue" = or i1 %"#292[0]1035", %"#295[0]1034"
store i1 %".fA_29_carry[0]'svalue", i1* %".fA_29_carry[0]"
%"#297[0]1036" = load i1, i1* %"#297[0]"
```

```
%"#296[0]1037" = load i1, i1* %"#296[0]"
%".fA_30_axb[0]'svalue" = xor i1 %"#296[0]1037", %"#297[0]1036"
store i1 %".fA_30_axb[0]'svalue", i1* %".fA_30_axb[0]"
%"#302[0]1038" = load i1, i1* %"#302[0]"
%"#301[0]1039" = load i1, i1* %"#301[0]"
%"#303[0]'svalue" = and i1 %"#301[0]1039", %"#302[0]1038"
store i1 %"#303[0]'svalue", i1* %"#303[0]"
%".fA_29_carry[0]1040" = load i1, i1* %".fA_29_carry[0]"
store i1 %".fA_29_carry[0]1040", i1* %".add_1_c[7]"
%".add_1_c[7]1041" = load i1, i1* %".add_1_c[7]"
store i1 %".add_1_c[7]1041", i1* %"#307[0]"
%"#305[0]1042" = load i1, i1* %"#305[0]"
%"#304[0]1043" = load i1, i1* %"#304[0]"
%".fA_31_axb[0]'svalue" = xor i1 %"#304[0]1043", %"#305[0]1042"
store i1 %".fA_31_axb[0]'svalue", i1* %".fA_31_axb[0]"
%"#307[0]1044" = load i1, i1* %"#307[0]"
%".fA_31_axb[0]1045" = load i1, i1* %".fA_31_axb[0]"
%"#308[0]'svalue" = and i1 %".fA_31_axb[0]1045", %"#307[0]1044"
store i1 %"#308[0]'svalue", i1* %"#308[0]"
%"#311[0]1046" = load i1, i1* %"#311[0]"
%"#308[0]1047" = load i1, i1* %"#308[0]"
%".fA_31_carry[0]'svalue" = or i1 %"#308[0]1047", %"#311[0]1046"
store i1 %".fA_31_carry[0]'svalue", i1* %".fA_31_carry[0]"
%".fA_31_carry[0]1048" = load i1, i1* %".fA_31_carry[0]"
store i1 %".fA_31_carry[0]1048", i1* %".add_1_c[8]"
%".fA_18_sum[0]1049" = load i1, i1* %".fA_18_sum[0]"
store i1 %".fA_18_sum[0]1049", i1* %".add_1_sum[1]"
%".add_1_sum[1]1050" = load i1, i1* %".add_1_sum[1]"
store i1 %".add_1_sum[1]1050", i1* %".math_0_sum[1]"
%".math_0_sum[1]1051" = load i1, i1* %".math_0_sum[1]"
store i1 %".math_0_sum[1]1051", i1* %".alu_0_m[1]"
%".alu_0_m[1]1052" = load i1, i1* %".alu_0_m[1]"
store i1 %".alu_0_m[1]1052", i1* %"#535[0]"
%"#537[0]1053" = load i1, i1* %"#537[0]"
%"#535[0]1054" = load i1, i1* %"#535[0]"
%".mux_4_as[1]'svalue" = and i1 %"#535[0]1054", %"#537[0]1053"
store i1 %".mux_4_as[1]'svalue", i1* %".mux_4_as[1]"
%".mux_4_as[1]1055" = load i1, i1* %".mux_4_as[1]"
store i1 %".mux_4_as[1]1055", i1* %"#540[0]"
%"#541[0]1056" = load i1, i1* %"#541[0]"
%"#540[0]1057" = load i1, i1* %"#540[0]"
%".mux_4_o[1]'svalue" = or i1 %"#540[0]1057", %"#541[0]1056"
store i1 %".mux_4_o[1]'svalue", i1* %".mux_4_o[1]"
```

```
  %".mux_4_o[1]1058" = load i1, i1* %".mux_4_o[1]"
  store i1 %".mux_4_o[1]1058", i1* %".alu_0_o[1]"
  %".alu_0_o[1]1059" = load i1, i1* %".alu_0_o[1]"
  store i1 %".alu_0_o[1]1059", i1* %x_1
  %x_11060 = load i1, i1* %x_1
  %printf1061 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt.2, i32
0, i32 0), i1 %x_11060)
  %".add_1_c[7]1062" = load i1, i1* %".add_1_c[7]"
  store i1 %".add_1_c[7]1062", i1* %"#306[0]"
  %".add_1_c[7]1063" = load i1, i1* %".add_1_c[7]"
  store i1 %".add_1_c[7]1063", i1* %"#299[0]"
  %".add_1_c[7]1064" = load i1, i1* %".add_1_c[7]"
  store i1 %".add_1_c[7]1064", i1* %"#298[0]"
  %".add_1_c[6]1065" = load i1, i1* %".add_1_c[6]"
  store i1 %".add_1_c[6]1065", i1* %"#290[0]"
  %".add_1_c[6]1066" = load i1, i1* %".add_1_c[6]"
  store i1 %".add_1_c[6]1066", i1* %"#283[0]"
  %".add_1_c[6]1067" = load i1, i1* %".add_1_c[6]"
  store i1 %".add_1_c[6]1067", i1* %"#282[0]"
  %".add_1_c[5]1068" = load i1, i1* %".add_1_c[5]"
  store i1 %".add_1_c[5]1068", i1* %"#274[0]"
  %".add_1_c[5]1069" = load i1, i1* %".add_1_c[5]"
  store i1 %".add_1_c[5]1069", i1* %"#267[0]"
  %".add_1_c[5]1070" = load i1, i1* %".add_1_c[5]"
  store i1 %".add_1_c[5]1070", i1* %"#266[0]"
  %".add_1_c[4]1071" = load i1, i1* %".add_1_c[4]"
  store i1 %".add_1_c[4]1071", i1* %"#258[0]"
  %".add_1_c[4]1072" = load i1, i1* %".add_1_c[4]"
  store i1 %".add_1_c[4]1072", i1* %"#251[0]"
  %".add_1_c[4]1073" = load i1, i1* %".add_1_c[4]"
  store i1 %".add_1_c[4]1073", i1* %"#250[0]"
  %".add_1_c[3]1074" = load i1, i1* %".add_1_c[3]"
  store i1 %".add_1_c[3]1074", i1* %"#242[0]"
  %".add_1_c[3]1075" = load i1, i1* %".add_1_c[3]"
  store i1 %".add_1_c[3]1075", i1* %"#235[0]"
  %".add_1_c[3]1076" = load i1, i1* %".add_1_c[3]"
  store i1 %".add_1_c[3]1076", i1* %"#234[0]"
  %".add_1_c[2]1077" = load i1, i1* %".add_1_c[2]"
  store i1 %".add_1_c[2]1077", i1* %"#226[0]"
  %".add_1_c[2]1078" = load i1, i1* %".add_1_c[2]"
  store i1 %".add_1_c[2]1078", i1* %"#219[0]"
  %".add_1_c[2]1079" = load i1, i1* %".add_1_c[2]"
  store i1 %".add_1_c[2]1079", i1* %"#218[0]"
```

```
%"#210[0]1080" = load i1, i1* %"#210[0]"
%".fA_19_axb[0]1081" = load i1, i1* %".fA_19_axb[0]"
%".fA_19_sum[0]'svalue" = xor i1 %".fA_19_axb[0]1081", %"#210[0]1080"
store i1 %".fA_19_sum[0]'svalue", i1* %".fA_19_sum[0]"
%"#219[0]1082" = load i1, i1* %"#219[0]"
%".fA_20_axb[0]1083" = load i1, i1* %".fA_20_axb[0]"
%"#220[0]'svalue" = and i1 %".fA_20_axb[0]1083", %"#219[0]1082"
store i1 %"#220[0]'svalue", i1* %"#220[0]"
%"#223[0]1084" = load i1, i1* %"#223[0]"
%"#220[0]1085" = load i1, i1* %"#220[0]"
%".fA_20_carry[0]'svalue" = or i1 %"#220[0]1085", %"#223[0]1084"
store i1 %".fA_20_carry[0]'svalue", i1* %".fA_20_carry[0]"
%"#218[0]1086" = load i1, i1* %"#218[0]"
%".fA_20_axb[0]1087" = load i1, i1* %".fA_20_axb[0]"
%".fA_20_sum[0]'svalue" = xor i1 %".fA_20_axb[0]1087", %"#218[0]1086"
store i1 %".fA_20_sum[0]'svalue", i1* %".fA_20_sum[0]"
%"#226[0]1088" = load i1, i1* %"#226[0]"
%".fA_21_axb[0]1089" = load i1, i1* %".fA_21_axb[0]"
%".fA_21_sum[0]'svalue" = xor i1 %".fA_21_axb[0]1089", %"#226[0]1088"
store i1 %".fA_21_sum[0]'svalue", i1* %".fA_21_sum[0]"
%"#235[0]1090" = load i1, i1* %"#235[0]"
%".fA_22_axb[0]1091" = load i1, i1* %".fA_22_axb[0]"
%"#236[0]'svalue" = and i1 %".fA_22_axb[0]1091", %"#235[0]1090"
store i1 %"#236[0]'svalue", i1* %"#236[0]"
%"#239[0]1092" = load i1, i1* %"#239[0]"
%"#236[0]1093" = load i1, i1* %"#236[0]"
%".fA_22_carry[0]'svalue" = or i1 %"#236[0]1093", %"#239[0]1092"
store i1 %".fA_22_carry[0]'svalue", i1* %".fA_22_carry[0]"
%"#234[0]1094" = load i1, i1* %"#234[0]"
%".fA_22_axb[0]1095" = load i1, i1* %".fA_22_axb[0]"
%".fA_22_sum[0]'svalue" = xor i1 %".fA_22_axb[0]1095", %"#234[0]1094"
store i1 %".fA_22_sum[0]'svalue", i1* %".fA_22_sum[0]"
%"#242[0]1096" = load i1, i1* %"#242[0]"
%".fA_23_axb[0]1097" = load i1, i1* %".fA_23_axb[0]"
%".fA_23_sum[0]'svalue" = xor i1 %".fA_23_axb[0]1097", %"#242[0]1096"
store i1 %".fA_23_sum[0]'svalue", i1* %".fA_23_sum[0]"
%"#251[0]1098" = load i1, i1* %"#251[0]"
%".fA_24_axb[0]1099" = load i1, i1* %".fA_24_axb[0]"
%"#252[0]'svalue" = and i1 %".fA_24_axb[0]1099", %"#251[0]1098"
store i1 %"#252[0]'svalue", i1* %"#252[0]"
%"#255[0]1100" = load i1, i1* %"#255[0]"
%"#252[0]1101" = load i1, i1* %"#252[0]"
%".fA_24_carry[0]'svalue" = or i1 %"#252[0]1101", %"#255[0]1100"
```

```
store i1 %".fA_24_carry[0]'svalue", i1* %".fA_24_carry[0]"
%"#250[0]1102" = load i1, i1* %"#250[0]"
%".fA_24_axb[0]1103" = load i1, i1* %".fA_24_axb[0]"
%".fA_24_sum[0]'svalue" = xor i1 %".fA_24_axb[0]1103", %"#250[0]1102"
store i1 %".fA_24_sum[0]'svalue", i1* %".fA_24_sum[0]"
%"#258[0]1104" = load i1, i1* %"#258[0]"
%".fA_25_axb[0]1105" = load i1, i1* %".fA_25_axb[0]"
%".fA_25_sum[0]'svalue" = xor i1 %".fA_25_axb[0]1105", %"#258[0]1104"
store i1 %".fA_25_sum[0]'svalue", i1* %".fA_25_sum[0]"
%"#267[0]1106" = load i1, i1* %"#267[0]"
%".fA_26_axb[0]1107" = load i1, i1* %".fA_26_axb[0]"
%"#268[0]'svalue" = and i1 %".fA_26_axb[0]1107", %"#267[0]1106"
store i1 %"#268[0]'svalue", i1* %"#268[0]"
%"#271[0]1108" = load i1, i1* %"#271[0]"
%"#268[0]1109" = load i1, i1* %"#268[0]"
%".fA_26_carry[0]'svalue" = or i1 %"#268[0]1109", %"#271[0]1108"
store i1 %".fA_26_carry[0]'svalue", i1* %".fA_26_carry[0]"
%"#266[0]1110" = load i1, i1* %"#266[0]"
%".fA_26_axb[0]1111" = load i1, i1* %".fA_26_axb[0]"
%".fA_26_sum[0]'svalue" = xor i1 %".fA_26_axb[0]1111", %"#266[0]1110"
store i1 %".fA_26_sum[0]'svalue", i1* %".fA_26_sum[0]"
%"#274[0]1112" = load i1, i1* %"#274[0]"
%".fA_27_axb[0]1113" = load i1, i1* %".fA_27_axb[0]"
%".fA_27_sum[0]'svalue" = xor i1 %".fA_27_axb[0]1113", %"#274[0]1112"
store i1 %".fA_27_sum[0]'svalue", i1* %".fA_27_sum[0]"
%"#283[0]1114" = load i1, i1* %"#283[0]"
%".fA_28_axb[0]1115" = load i1, i1* %".fA_28_axb[0]"
%"#284[0]'svalue" = and i1 %".fA_28_axb[0]1115", %"#283[0]1114"
store i1 %"#284[0]'svalue", i1* %"#284[0]"
%"#287[0]1116" = load i1, i1* %"#287[0]"
%"#284[0]1117" = load i1, i1* %"#284[0]"
%".fA_28_carry[0]'svalue" = or i1 %"#284[0]1117", %"#287[0]1116"
store i1 %".fA_28_carry[0]'svalue", i1* %".fA_28_carry[0]"
%"#282[0]1118" = load i1, i1* %"#282[0]"
%".fA_28_axb[0]1119" = load i1, i1* %".fA_28_axb[0]"
%".fA_28_sum[0]'svalue" = xor i1 %".fA_28_axb[0]1119", %"#282[0]1118"
store i1 %".fA_28_sum[0]'svalue", i1* %".fA_28_sum[0]"
%"#290[0]1120" = load i1, i1* %"#290[0]"
%".fA_29_axb[0]1121" = load i1, i1* %".fA_29_axb[0]"
%".fA_29_sum[0]'svalue" = xor i1 %".fA_29_axb[0]1121", %"#290[0]1120"
store i1 %".fA_29_sum[0]'svalue", i1* %".fA_29_sum[0]"
%"#299[0]1122" = load i1, i1* %"#299[0]"
%".fA_30_axb[0]1123" = load i1, i1* %".fA_30_axb[0]"
```

```
%"#300[0]'svalue" = and i1 %".fA_30_axb[0]1123", %"#299[0]1122"
store i1 %"#300[0]'svalue", i1* %"#300[0]"
%"#303[0]1124" = load i1, i1* %"#303[0]"
%"#300[0]1125" = load i1, i1* %"#300[0]"
%".fA_30_carry[0]'svalue" = or i1 %"#300[0]1125", %"#303[0]1124"
store i1 %".fA_30_carry[0]'svalue", i1* %".fA_30_carry[0]"
%"#298[0]1126" = load i1, i1* %"#298[0]"
%".fA_30_axb[0]1127" = load i1, i1* %".fA_30_axb[0]"
%".fA_30_sum[0]'svalue" = xor i1 %".fA_30_axb[0]1127", %"#298[0]1126"
store i1 %".fA_30_sum[0]'svalue", i1* %".fA_30_sum[0]"
%"#306[0]1128" = load i1, i1* %"#306[0]"
%".fA_31_axb[0]1129" = load i1, i1* %".fA_31_axb[0]"
%".fA_31_sum[0]'svalue" = xor i1 %".fA_31_axb[0]1129", %"#306[0]1128"
store i1 %".fA_31_sum[0]'svalue", i1* %".fA_31_sum[0]"
%".fA_20_sum[0]1130" = load i1, i1* %".fA_20_sum[0]"
store i1 %".fA_20_sum[0]1130", i1* %".add_1_sum[2]"
%".fA_22_sum[0]1131" = load i1, i1* %".fA_22_sum[0]"
store i1 %".fA_22_sum[0]1131", i1* %".add_1_sum[3]"
%".fA_24_sum[0]1132" = load i1, i1* %".fA_24_sum[0]"
store i1 %".fA_24_sum[0]1132", i1* %".add_1_sum[4]"
%".fA_26_sum[0]1133" = load i1, i1* %".fA_26_sum[0]"
store i1 %".fA_26_sum[0]1133", i1* %".add_1_sum[5]"
%".fA_28_sum[0]1134" = load i1, i1* %".fA_28_sum[0]"
store i1 %".fA_28_sum[0]1134", i1* %".add_1_sum[6]"
%".fA_30_sum[0]1135" = load i1, i1* %".fA_30_sum[0]"
store i1 %".fA_30_sum[0]1135", i1* %".add_1_sum[7]"
%".add_1_sum[2]1136" = load i1, i1* %".add_1_sum[2]"
store i1 %".add_1_sum[2]1136", i1* %".math_0_sum[2]"
%".add_1_sum[3]1137" = load i1, i1* %".add_1_sum[3]"
store i1 %".add_1_sum[3]1137", i1* %".math_0_sum[3]"
%".add_1_sum[4]1138" = load i1, i1* %".add_1_sum[4]"
store i1 %".add_1_sum[4]1138", i1* %".math_0_sum[4]"
%".add_1_sum[5]1139" = load i1, i1* %".add_1_sum[5]"
store i1 %".add_1_sum[5]1139", i1* %".math_0_sum[5]"
%".add_1_sum[6]1140" = load i1, i1* %".add_1_sum[6]"
store i1 %".add_1_sum[6]1140", i1* %".math_0_sum[6]"
%".add_1_sum[7]1141" = load i1, i1* %".add_1_sum[7]"
store i1 %".add_1_sum[7]1141", i1* %".math_0_sum[7]"
%".math_0_sum[2]1142" = load i1, i1* %".math_0_sum[2]"
store i1 %".math_0_sum[2]1142", i1* %".alu_0_m[2]"
%".alu_0_m[2]1143" = load i1, i1* %".alu_0_m[2]"
store i1 %".alu_0_m[2]1143", i1* %"#542[0]"
%"#544[0]1144" = load i1, i1* %"#544[0]"
```

```
%"#542[0]1145" = load i1, i1* %"#542[0]"
%".mux_4_as[2]'svalue" = and i1 %"#542[0]1145", %"#544[0]1144"
store i1 %".mux_4_as[2]'svalue", i1* %".mux_4_as[2]"
%".mux_4_as[2]1146" = load i1, i1* %".mux_4_as[2]"
store i1 %".mux_4_as[2]1146", i1* %"#547[0]"
%".math_0_sum[3]1147" = load i1, i1* %".math_0_sum[3]"
store i1 %".math_0_sum[3]1147", i1* %".alu_0_m[3]"
%".alu_0_m[3]1148" = load i1, i1* %".alu_0_m[3]"
store i1 %".alu_0_m[3]1148", i1* %"#549[0]"
%"#551[0]1149" = load i1, i1* %"#551[0]"
%"#549[0]1150" = load i1, i1* %"#549[0]"
%".mux_4_as[3]'svalue" = and i1 %"#549[0]1150", %"#551[0]1149"
store i1 %".mux_4_as[3]'svalue", i1* %".mux_4_as[3]"
%".mux_4_as[3]1151" = load i1, i1* %".mux_4_as[3]"
store i1 %".mux_4_as[3]1151", i1* %"#554[0]"
%".math_0_sum[4]1152" = load i1, i1* %".math_0_sum[4]"
store i1 %".math_0_sum[4]1152", i1* %".alu_0_m[4]"
%".alu_0_m[4]1153" = load i1, i1* %".alu_0_m[4]"
store i1 %".alu_0_m[4]1153", i1* %"#556[0]"
%"#558[0]1154" = load i1, i1* %"#558[0]"
%"#556[0]1155" = load i1, i1* %"#556[0]"
%".mux_4_as[4]'svalue" = and i1 %"#556[0]1155", %"#558[0]1154"
store i1 %".mux_4_as[4]'svalue", i1* %".mux_4_as[4]"
%".mux_4_as[4]1156" = load i1, i1* %".mux_4_as[4]"
store i1 %".mux_4_as[4]1156", i1* %"#561[0]"
%".math_0_sum[5]1157" = load i1, i1* %".math_0_sum[5]"
store i1 %".math_0_sum[5]1157", i1* %".alu_0_m[5]"
%".alu_0_m[5]1158" = load i1, i1* %".alu_0_m[5]"
store i1 %".alu_0_m[5]1158", i1* %"#563[0]"
%"#565[0]1159" = load i1, i1* %"#565[0]"
%"#563[0]1160" = load i1, i1* %"#563[0]"
%".mux_4_as[5]'svalue" = and i1 %"#563[0]1160", %"#565[0]1159"
store i1 %".mux_4_as[5]'svalue", i1* %".mux_4_as[5]"
%".mux_4_as[5]1161" = load i1, i1* %".mux_4_as[5]"
store i1 %".mux_4_as[5]1161", i1* %"#568[0]"
%".math_0_sum[6]1162" = load i1, i1* %".math_0_sum[6]"
store i1 %".math_0_sum[6]1162", i1* %".alu_0_m[6]"
%".alu_0_m[6]1163" = load i1, i1* %".alu_0_m[6]"
store i1 %".alu_0_m[6]1163", i1* %"#570[0]"
%"#572[0]1164" = load i1, i1* %"#572[0]"
%"#570[0]1165" = load i1, i1* %"#570[0]"
%".mux_4_as[6]'svalue" = and i1 %"#570[0]1165", %"#572[0]1164"
store i1 %".mux_4_as[6]'svalue", i1* %".mux_4_as[6]"
```

```
%".mux_4_as[6]1166" = load i1, i1* %".mux_4_as[6]"
store i1 %".mux_4_as[6]1166", i1* %"#575[0]"
%".math_0_sum[7]1167" = load i1, i1* %".math_0_sum[7]"
store i1 %".math_0_sum[7]1167", i1* %".alu_0_m[7]"
%".alu_0_m[7]1168" = load i1, i1* %".alu_0_m[7]"
store i1 %".alu_0_m[7]1168", i1* %"#577[0]"
%"#579[0]1169" = load i1, i1* %"#579[0]"
%"#577[0]1170" = load i1, i1* %"#577[0]"
%".mux_4_as[7]'svalue" = and i1 %"#577[0]1170", %"#579[0]1169"
store i1 %".mux_4_as[7]'svalue", i1* %".mux_4_as[7]"
%".mux_4_as[7]1171" = load i1, i1* %".mux_4_as[7]"
store i1 %".mux_4_as[7]1171", i1* %"#582[0]"
%"#548[0]1172" = load i1, i1* %"#548[0]"
%"#547[0]1173" = load i1, i1* %"#547[0]"
%".mux_4_o[2]'svalue" = or i1 %"#547[0]1173", %"#548[0]1172"
store i1 %".mux_4_o[2]'svalue", i1* %".mux_4_o[2]"
%"#555[0]1174" = load i1, i1* %"#555[0]"
%"#554[0]1175" = load i1, i1* %"#554[0]"
%".mux_4_o[3]'svalue" = or i1 %"#554[0]1175", %"#555[0]1174"
store i1 %".mux_4_o[3]'svalue", i1* %".mux_4_o[3]"
%"#562[0]1176" = load i1, i1* %"#562[0]"
%"#561[0]1177" = load i1, i1* %"#561[0]"
%".mux_4_o[4]'svalue" = or i1 %"#561[0]1177", %"#562[0]1176"
store i1 %".mux_4_o[4]'svalue", i1* %".mux_4_o[4]"
%"#569[0]1178" = load i1, i1* %"#569[0]"
%"#568[0]1179" = load i1, i1* %"#568[0]"
%".mux_4_o[5]'svalue" = or i1 %"#568[0]1179", %"#569[0]1178"
store i1 %".mux_4_o[5]'svalue", i1* %".mux_4_o[5]"
%"#576[0]1180" = load i1, i1* %"#576[0]"
%"#575[0]1181" = load i1, i1* %"#575[0]"
%".mux_4_o[6]'svalue" = or i1 %"#575[0]1181", %"#576[0]1180"
store i1 %".mux_4_o[6]'svalue", i1* %".mux_4_o[6]"
%"#583[0]1182" = load i1, i1* %"#583[0]"
%"#582[0]1183" = load i1, i1* %"#582[0]"
%".mux_4_o[7]'svalue" = or i1 %"#582[0]1183", %"#583[0]1182"
store i1 %".mux_4_o[7]'svalue", i1* %".mux_4_o[7]"
%".mux_4_o[2]1184" = load i1, i1* %".mux_4_o[2]"
store i1 %".mux_4_o[2]1184", i1* %".alu_0_o[2]"
%".mux_4_o[3]1185" = load i1, i1* %".mux_4_o[3]"
store i1 %".mux_4_o[3]1185", i1* %".alu_0_o[3]"
%".mux_4_o[4]1186" = load i1, i1* %".mux_4_o[4]"
store i1 %".mux_4_o[4]1186", i1* %".alu_0_o[4]"
%".mux_4_o[5]1187" = load i1, i1* %".mux_4_o[5]"
```

```
  store i1 %".mux_4_o[5]1187", i1* %".alu_0_o[5]"
  %".mux_4_o[6]1188" = load i1, i1* %".mux_4_o[6]"
  store i1 %".mux_4_o[6]1188", i1* %".alu_0_o[6]"
  %".mux_4_o[7]1189" = load i1, i1* %".mux_4_o[7]"
  store i1 %".mux_4_o[7]1189", i1* %".alu_0_o[7]"
  %".alu_0_o[2]1190" = load i1, i1* %".alu_0_o[2]"
  store i1 %".alu_0_o[2]1190", i1* %x_2
  %x_21191 = load i1, i1* %x_2
  %printf1192 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt.3, i32
0, i32 0), i1 %x_21191)
  %".alu_0_o[3]1193" = load i1, i1* %".alu_0_o[3]"
  store i1 %".alu_0_o[3]1193", i1* %x_3
  %x_31194 = load i1, i1* %x_3
  %printf1195 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt.4, i32
0, i32 0), i1 %x_31194)
  %".alu_0_o[4]1196" = load i1, i1* %".alu_0_o[4]"
  store i1 %".alu_0_o[4]1196", i1* %x_4
  %x_41197 = load i1, i1* %x_4
  %printf1198 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt.5, i32
0, i32 0), i1 %x_41197)
  %".alu_0_o[5]1199" = load i1, i1* %".alu_0_o[5]"
  store i1 %".alu_0_o[5]1199", i1* %x_5
  %x_51200 = load i1, i1* %x_5
  %printf1201 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt.6, i32
0, i32 0), i1 %x_51200)
  %".alu_0_o[6]1202" = load i1, i1* %".alu_0_o[6]"
  store i1 %".alu_0_o[6]1202", i1* %x_6
  %x_61203 = load i1, i1* %x_6
  %printf1204 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt.7, i32
0, i32 0), i1 %x_61203)
  %".alu_0_o[7]1205" = load i1, i1* %".alu_0_o[7]"
  store i1 %".alu_0_o[7]1205", i1* %x_7
  %x_71206 = load i1, i1* %x_7
  %printf1207 = call i32 (i8*, ...) @printf(i8* getelementptr inbounds ([10 x i8], [10 x i8]* @prt.8, i32
0, i32 0), i1 %x_71206)
  ret i1 false
}
```

8. Lessons Learned

Maryam

I learned that time isn't our friend in this class. If you want to succeed in this class, then you have to start early and plan ahead. Meeting weekly and working on the project even when you don't have a deliverable is the key to success in this class. Not everyone will have the same skills on the team, but making sure you cater to their strengths is essential. Finally, make sure to use all the resources that are available to you. Overall, you will learn a lot from this project and you will succeed if you start early.

Lalo

I learned how painful functional programming can be, but I enjoyed going through that pain. There is definitely a lot of individual work in this class and time that you need to dedicate outside of class in order to understand what is happening and to be able to build your own compiler. I definitely learned a lot from this class and I enjoyed it. I got so excited by the work of the compiler that I look forward to implementing some lingering features that we didn't have time for.

Gael

I learned how essential it is have a soft deadline a couple of days before our deliverables. I also learned how efficient it is to use divide and conquer strategy. I definitely learned the importance of communication when you're working on a big team. I also learned how important it is to stay on top you're things because you can easily get lost and fall behind in this project.

Adiza

In terms of my major takeaways from the course, I truly think working on a software development team was key in my personal growth as an engineer. For me, it was something new to work with others on a consistent basis for such a long period of time. Additionally, in terms of advice for future students, I would suggest consulting the professor or the teaching assistants as much as necessary. They are there to help and the earlier you receive that help, the much better your project will be.

Dan

I learned an equal mix of both technical and "soft" skills in this course. I now have a stronger understanding of regular expressions and am generally comfortable writing and reading OCaml. OCaml was my first exposure to functional programming as well and I expect this to be useful in the future. Because we chose the create a hardware descriptive language, I was also able to better connect software engineering with my circuit design knowledge. I have a better understanding of overall compiler design and noticed I am now able to make more sense of compiler error messages in other languages because I better understand the pipeline my code goes through when it is run. As the language guru for my team, I learned a lot about what goes into an LRM, how specific it must be, and the challenges in designing a language before writing the compiler for it. There were a lot of ideas we threw around that were changed multiple times or completely scrapped. I found the scanner, parser and AST portions of the project to be the most fun. But I think the most valuable skill I am taking away from PLT is how to work efficiently and effectively with a team. Good team coordination and communication fall into the realm of intangible skills that go beyond academia and can set you apart from other coders. Everyone on team UNI-corn put an impressive amount of commitment and enthusiasm into this project.