C Annotations for Concurrent Information Flow Security

Alexander Blyth

June 2021

Contents

Abstract

The aim of this thesis is to to preserve annotations written in C source code through intermediate representations of compilation and compiler optimisations to assist verification of information flow security in concurrent programs through static analysis techniques. Information flow security in concurrency is difficult due to the increasing complexity introduced with multiple threads. Additionally, compiler optimisations can break security guarantees that have been verified in source code. Thus, any verification of security policies must be performed at assembly or lower representations. Three methods of preserving annotations were analysed for assessing the suitability for verifying information flow security in concurrent programs. Each technique was assessed to observe if all necessary annotations can be preserved and to observe how it performs through aggressive compiler optimisations. CompCert AIS annotations were robust and preserved annotations through aggressive compiler optimisations, however, it does not support volatile variables which are essential for concurrent programs. CompCert builtin annotations do allow annotations of volatile variables, however, it is not robust and compiler optimisations were not always performed due to the annotations. Finally, inline assembly was excellent at preserving annotations, and a tool was developed to assist annotating assembly using this method.

Chapter 1

Topic Definition

There is a high degree of complexity in verifying security guarantees in concurrent programs [19][27][29]. Additionally, aggressive compiler optimisations can modify the binary output in unexpected ways [7]. To preserve the security of a program, the flow of sensitive information must be protected to avoid flowing in to untrusted sources [2]. This is where static analysis tools can be used to verify the integrity of security guarantees and the flow of sensitive information. In this thesis, we look to explore a solution to information flow security in concurrent programs through analysing the output after aggressive compiler optimisations.

We propose a tool to analyse C programs to detect security violations in information flow control. This tool will preserve annotations provided by the programmer in source code through lowering passes and aggressive compiler optimisations. The tool will work alongside the *Weakest Precondition for Information Flow* (wpif) transformer described by Winter et al. [31] to allow the programmer to assess the security of information flow in their concurrent programs.

Similar tools for propagating annotations and properties through compiler optimisations have been explored [30] [25] [18], however, these tools focus on either generic solutions for propagating properties or to assist the static analysis of the *Worst Case Execution Time*.

Chapter 2

Background

Vulnerabilities in software can lead to catastrophic consequences when manipulated by attackers. In an open-source cryptographic software library (OpenSSL) used by an estimated two-thirds of web servers [16] a security flaw called Heartbleed was discovered. Secure secrets such as financial data, encryption keys, or anything else stored in the server's memory could be leaked. Normally, one would send a Heartbeat request with a text string payload and the length of the payload. For example, a message of "hello" could be sent with the length of the message, 5. However, due to a improper input validation (buffer over-read), one could send a length longer that the string they actually sent. This would cause the server to respond with the original message and anything that was in the allocated memory at the time, including any potentially sensitive information. An example of this is shown in figure 2.1 [13].

Heartbleed was one of the most dangerous security bugs ever, and calls for major reflection by everyone in industry and research [2].

2.1 Information Security

Computer security is defined as a preservation of **integrity**, **availability** and **confidentiality** of information, and extends to include not only software but hardware, firmware, information, data and telecommunications [15]. Confidentiality requires that data is not

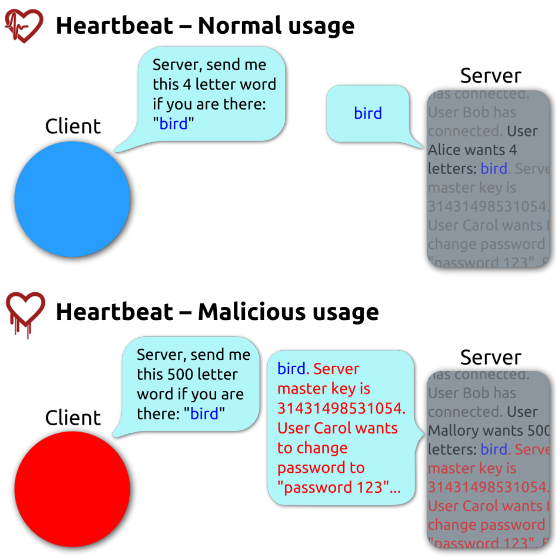

Figure 2.1: The Heartbleed bug. [13]

available to unauthorised users, and that individuals can control what information can be collected and disclosed to others. Data integrity requires that only authorised sources can modify data, and that the system can perform tasks without interference from outside sources. Finally, availability of a system requires that service is not denied to authorised users. Together, these principles create the CIA triad [28]. To enforce a secure system, all three principles must be upheld.

Modern programs are becoming increasingly complex with potential for networking, multi-threading and storage permissions and more. As such, security mechanisms must be put in place to verify and enforce the information security requirements. The adequacy of a security mechanism depends on the adversary model. The adversary model is a formal definition of the attacker and their abilities in a system, and defines who we are protecting against [10]. Ideally we would like to design a system to protect against the strongest adversary or attacker, however, this is often not required or even possible. Instead, we must consider the security policy, security mechanism and strongest adversary model to make a system secure [2].

Standard security processes that handle access control such as a firewall or antivirus software can fail as they do not constrain where information is allowed to flow, meaning that once access is granted it can propagate to insecure processes where it can be accessed

```
secret := 0xCODE mod 2
public := 1
if secret = 1
    public := 0

```

Figure 2.2: Implicit flow of data to a public variable

by attackers. Where a large system is being used, it is often the case that not all components of the codebase can be trusted, often containing potentially malicious code [24]. Take for example your modern-day web project. Where a package manager such as Node Package Manager (npm) could be used to utilise open-source packages to speed up development progress, it could also inadvertently introduce security vulnerabilities. Rewriting all packages used to ensure security would be time-consuming and expensive and is not a viable option. Instead, controlling where information can flow and preventing secure data from flowing into untrusted sources or packages can maintain confidentiality of a system.

One may suggest runtime monitoring the flow of data to prevent leakage of secure data. Aside from the obvious computational and memory overhead, this method can have its own issues. Although it can detect an *explicit* flow of data from a secure variable to a public variable, it is unable to detect *implicit* data flow, where the state of secure data can be inferred from the state of public data or a public variable [9]. Take for example figure 2.2. In this example, a public, readable variable is initially set to the value of 1. There is also a secret variable which may contain a key, password or some other secret that must be kept secure from any attackers. Depending on the value of the secret variable an attacker can infer information about this variable depending on whether the value of the public variable is updated to a value of 0. Assuming that the inner workings of the system is known by the attacker, information about the secret variable can be leaked *implicitly* and inferred by the state of public variables.

Security concerns do not only exist at the application level. In a huge codebase such as an OS, different low-level bugs can be exploited to gain access to data, such as by using buffer overflows to inject viruses or trojans [1].

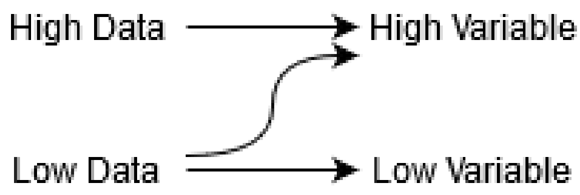

Figure 2.3: Permitted flow of data

2.2 Information Flow Control

As seen by the issues that can be introduced via implicit and explicit flow of data, there is room to improve on the existing techniques imposed by current security measures. To protect confidentiality, secure or sensitive information must be prevented from flowing into public on insecure variables. Additionally, to protect integrity, untrusted data from public sources must be prevented from flowing into secure or trusted destinations [2]. An information flow security policy can be introduced to classify or label data, or more formally, a set of *security levels* to which each object is bound by across a multi-level security lattice [8]. In this thesis, we will focus primarily on preserving confidentiality.

Many security levels can be identified to classify different classes of objects, however, for now we will consider two security levels: high and low. Data labelled as high signifies that the data is secret, and low data is classified as non-sensitive data, such that it does not need to be protected from an attacker or adversary. Variables that can hold data in a program can additionally be classified as high or low as a *security classification*. A variable's security classification shows the highest classification of data it can safely contain [31]. A high variable can hold both high and low data, whereas a low variable which is visible to an attacker can only safely hold low data. As mentioned previously, confidentiality must be upheld by preventing high or secret data from flowing to low or public variables where an attacker can observe it. The permitted flow of data can be observed in 2.3. Note that high data is not allowed to flow into low variables.

2.3 Information Flow Security in Concurrency

Controlling the flow of information is a difficult problem, however, this is only exacerbated in concurrent programs, which are a well known source of security issues [19][27][29]. Research has been conducted into concurrent programs to explore ways the security of concurrent programs can be verified. Mantel et al. [20] introduced the concept of assumption and guarantee conditions, where assumptions are made about how concurrent threads access shared memory and guarantees are made about how an individual thread access shared memory that other threads may rely upon. Each thread can be observed individually using assumptions over the environment behaviour of other threads that can be then used to prove a guarantee about that individual thread. As two concurrent threads can interleave their steps and behaviour, there is a lot of complexity and possibilities for the overall behaviour. This concept of assumptions (or rely) and guarantee conditions can reduce the complexity of understanding interleaving behaviour in threads and assist in verifying the correctness of information flow security in concurrency. However, this approach is limited in the types of assumptions and guarantees it supports. Building on this, Murray et al. [12] [21] provide information flow logic on how to handle dynamic, value-dependent security levels in concurrent programs. In this case, the security level of a particular variable may depend on one or more other variables in the program. As such, the variable's security level can change as the state of the program changes. This logic is essential where the security level of data depends on its source. However, this approach is not sufficient when analysing non-blocking programs. The approach relies heavily on locks which block particular threads from executing. This in turn leads to slower processing due to blocked threads [23].

To overcome information flow security in non-blocking concurrent threads, Winter et al. [31] explores verifying security properties such as non-interference through the use of general rely/guarantee conditions using backwards, weakest precondition reasoning. Such an analysis would additionally handle implicit flows as shown in figure 2.3. Ideally a tool could be created to verify security policies required for sensitive processes. Users of this system could provide rely/guarantee conditions for each thread as well as security levels for data and variables i.e. high or low data and variables. Working backwards through the execution of the program, violations of the security policy will be detected. Detected

```
crypt() {
    key := 0xCODE // Read key
    ... // Work with the key
    key := 0x0 // Clear memory
}

```

Figure 2.4: Implicit flow of data to a public variable [7]

violations could be due to an incorrect assumption of the rely and guarantee conditions or a failure to uphold the security policy. This thesis will focus on the compilation stage of this tool.

2.4 Compilers and Security

Compilers are well known to be a weak link between source code and the hardware executing it. Source code that has been verified to provide a security guarantee, potentially using formal techniques, may not hold those security guarantees when being executed. This is caused by compiler optimisations that may be technically correct, however, a compiler has no notion of timing behaviour or on the expected state of memory after executing a statement [7]. This problem is known as the *correctness security gap*. One example of the correctness security gap is caused by an optimisation called dead store elimination. Figure 2.4 was derived from CWE-14 [6] and CWE-733 [5] and used by D'Silva et al. [7]. Here a secret key was retrieved and stored in a local variable to perform some work. After completing the work, and to prevent sensitive data from flowing into untrusted sources, the key is wiped from memory by assigning it the value 0x0.

From the perspective of the source code, a programmer would expect the sensitive data from key to be scrubbed after exiting the function. However, key is a variable local to the function. As key is not read after exiting the function, the statement that assigns key to a value of 0x0 will be removed as part of dead store elimination. This results in lingering memory that could be exploited by an attacker. In GCC, with compiler optimisations on, dead store elimination is performed by default [22]. Additionally, dead store elimination has been proven to be functionally correct [3][17].

This leads to the question, *what security guarantees in source code are being violated by compiler optimisations?* Although one could analyse each individual compiler optimisation to check for potential security violations in source code, defensively programming against the compiler can be counter-initiative. Additionally, compilers are getting better at optimising away tricks programmers write to work against the compiler, and thus is not a future-proof solution [26]. One might also suggest turning compiler optimisations off, however, this leads to slower code. In a concurrent system where execution time is critical, turning compiler optimisations off is not a viable option. Instead an alternative solution is to perform a static analysis on binary or assembly for security violations. As compilation has already been executed, such analysis would reveal security guarantee violations that result due to compiler optimisations.

2.5 Annotations

This project can take two routes; the proposed tool will be required to run an analysis on either binary or assembly. For either route, annotations used to guide a static security analysis will need to be provided by the user in the C programs they write. The tool will then be required to propagate these annotations down to compiled forms, i.e. binary or assembly. From here, a static analysis can be conducted as described by Winter et al. [31]. Ideally these annotations can be propagated through with little to no modification of the C Compiler being used as to reduce complexity and increase modularity and reusability of such a a tool. However, it is unclear as to whether passing annotations down with no modification to the compiler is currently possible. In this thesis, this issue will be explored.

Running a static analysis on a binary can be difficult due to the low level nature of a binary file. As such, to sufficiently perform such an analysis, the binary would be required to be decompiled to a higher-level form, such as an assembly file. From here a static analysis could be conducted. The alternative approach would be to perform the analysis directly on the compiled assembly output files rather than reducing these to binary. Currently, it is unclear as to what compiler optimisations are made when reducing an assembly file to binary, and will be explored further throughout the lifetime of this thesis. The

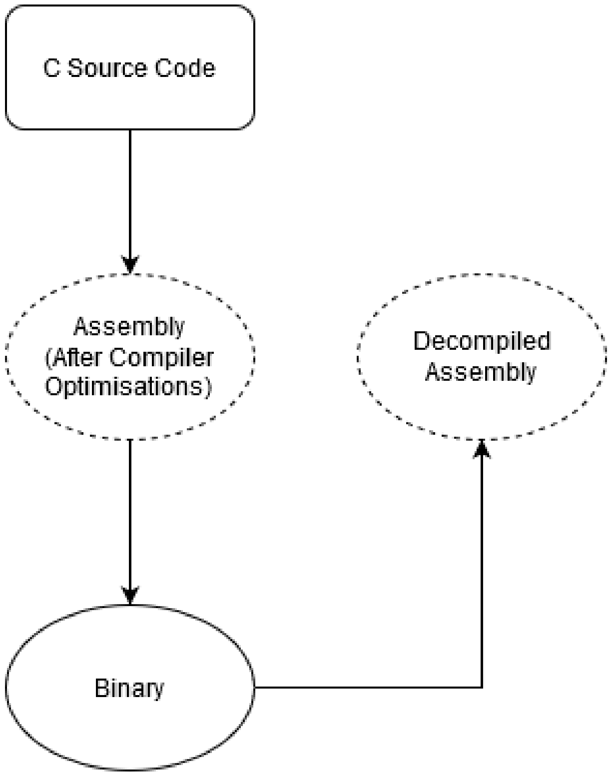

Figure 2.5: The static analysis options after compilation.

flow of information can be viewed in Figure 2.5, where formats a static analysis can be performed are outlined in a dashed line. In GCC, "temporary" intermediate files can be stored using the flag *save-temps* [14]. These stored files can then be used for analysis.

2.6 Related Work

In safety-critical real-time software such as flight control systems, it is required to analyse the *Worst Case Execution Time* (WCET). This kind of analysis can be conducted using static analysis tools to estimate safe upper bounds. In the case of AbsInt's aiT tool this analysis is conducted alongside compiler annotations to assist where loop bounds cannot be computed statically. In these cases, the user can provide annotations to guide the analysis tools [25]. This tool builds on an existing annotation mechanism that exist in CompCert, a C compiler that has been formally verified for use in life-critical and mission-critical software [4][18]. CompCert annotations are not limited to WCET analysis. A general mechanism for attaching free-form annotations that propagate through to assembly files can be achieved with CompCert. This approach is able to reliably transmit compiler annotations through to binary through method calls which are carried through compilation and the linked executable without using external annotation files. CompCert prints annotation strings as a comment in the generated assembly code, and an additional tool is used to parse these comments and generate annotations. However, due to its treatment as an external function, annotations cannot be placed at the top level of a compilation unit, unlike a variable declaration. Compiler optimisations can additionally cause further issues when trying to preserve annotations through compilation. If dead code is eliminated, annotations associated with that code can be lost as well. Extra care needs to be taken to avoid these optimisations destroying links between properties and the code they refer to during such transformations.

A similar approach to CompCert is used by The ENTRA (Whole-Systems ENergy TRAnsparency). As part of providing a common assertion language, pragmas are used to propagate information through to comments in the assembler files. Information is retained in LLVM IR and ISA representations. However, these annotations are not stored in the final binary and thus comments must be extracted from assembler files [11].

Vu et al. [30] explore capturing and propagating properties from the source code level though though lowering passes and intermediate representations. Their goal was to maintain these properties to binary through aggressive compiler optimisations. As compilers only care about functional correctness, they have no notion of the link between properties and the code it refers to. Thus, there is no way to constrain transformations to preserve this link or to update these properties after the transformation. As such, they approached the problem to create a generic solution, modifying a LLVM compiler with virtually no optimisation changes. This was done by creating a library in LLVM. The properties were stored in strings, and these strings were parsed to build a list of observed variables and memory location. A LLVM pass was inserted to store all these properties in metadata. After each optimisation pass, a verification pass was inserted to check the presence of metadata representing the properties, variables amd memory locations. If an optimisation pass had cased the verification to fail the programmer would then be notified, to which they could annotate differently or disable the optimisation.

Chapter 3

Approach

The approach was set out by first analysing existing methods of preserving annotations through intermediate representations. These include the:

- compCert Verified C Compiler,
- GNU Extension for Extended Inline Assembly, and
- Modifying the LLVM compiler to preserve annotations throughout intermediate representations.

Each of these approaches will be analysed individually for viability across each of the test cases outlined in section 3.1. For approaches that pass all necessary test cases, a further analysis will be conducted into its suitability and development of any necessary tools to assist in the preservation technique, as outlined in sections 3.2 and 3.4. Finally, an analysis on the runtime efficiency of the program will be conducted to assess the success of the annotations with various levels of optimisation. The approach for this analysis is outlined in 3.3.

3.1 Test Cases

A suite of test C programs (See Appendix A) were created to assist in guiding the process of evaluating each approach as a possible means of preserving annotations. Each program has inline comments documenting the annotation that should be preserved and its location within the program. Additionally, each program aims to test a separate element required to perform a static wpif analysis. Namely, these are to preserve the following through to the assembly output:

1. comments,
2. simple and complex variables (e.g. struct elements and volatile global variables),
3. security policies,
4. predicates on the initial state, and
5. loop invariants.

Each test was conducted to assess the viability of each approach of preserving annotations. If the approach cannot preserve all the required annotations described in the aforementioned list, then it is not viable for a wpif analysis and another technique must be explored.

The justification for each of the test files are as follows:

comment.c

This test case is primarily a stepping stone to testing more complex scenarios. Here we have a generic comment "critical comment" and we are looking to preserve it through to the assembly. As well as preserving the comment itself, the location of the comment within the source code is to be preserved.

variable.c

The test file *variable.c* builds off *comment.c*, however, we are additionally looking to preserve annotations about local variables within the program. Here multiple variable types are tested:

- int,
- char,
- unsigned int,
- short,
- long,
- float, and
- double.

With each of these variables their type data is included as an annotation. This test is particularly interesting as with higher levels of optimisation we can observe how the annotations behave when a variable is optimised out.

volatile.c

This test program looks at how the technique handles volatile variables. A variable declared as volatile tells the program its value could change unexpectedly. This is especially important when dealing with concurrent programs. If the technique cannot handle volatile variables it is unable to be used for a wpif analysis.

loop.c

This test program tests how the annotator handles loops and loop invariants. It contains security policies, predicates on the initial state and loop invariants.

rooster.c

The test program, *rooster.c* delves into a more complex program, combining several features of the previous tests. It contains annotations within functions and global variables.

password.c

This program tests how annotations are preserved within structs, a user-defined data type. Additionally, *password.c* is a more complex program with multiple functions.

deadStoreElimination.c

Testing dead store elimination is a bit more complex, as it requires comparing the compiled output before and after compiler optimisations are turned on. Here, the test program simulates the program described in section 2.4.

pread.c

The program *pread.c* is a culmination of all the previous test cases, and is similar to *loop.c*, however, the global variables within it are volatile. It requires all the necessary components for a wpif analysis.

3.2 Quality Analysis

Although a method of preserving c annotations may be able to successfully pass all the test cases, it is important to avoid modifying the assembly instructions. The reason for performing a static analysis on the compiled output is due to the optimisations performed by the compiler. As such, it is important to ensure that preserved annotations do not remove or undo any optimisations that may have been performed by the compiler.

The methodology for testing quality in this manner is to compare the compiled assembly output for a program with annotations on to the compiled assembly output for the same program without annotations. If unnecessary assembly instructions have been added, it is indicative that the annotations has modified the program in unintended ways.

3.3 Efficiency and Optimisation

In the case that the annotations have introduced additional statements into the compiled assembly output, understanding the extent of these changes is important. Here, a efficiency analysis can be conducted on the assembly. Using big O notation, an upper bound can be placed on the program. Doing so allows for a comparison of the efficiency of the annotated and non-annotated assembly.

Let $A(n)$ be the function describing the annotated assembly, and $B(n)$ be the function describing the non-annotated assembly. Then,

$$A(n) \in \Theta(g(n))$$

$$B(n) \in \Theta(h(n))$$

If the non-annotated assembly has a lower bound than the annotated assembly, such that

$$h(n) \in O(g(n)), and$$

$$g(n) \notin O(h(n))$$

then the annotations have modified the program in a way that reduces runtime efficiency. It is important to detect when this has happened, as it indicates the annotations have reversed the intended compiler optimisations.

In the case where the annotation process has resulted in additional assembly instructions inserted into the compiled output, however, they do not reduce runtime efficiency in terms of big O notation, a empirical analysis of the runtime duration of a program can be conducted to assess the disadvantage of the annotated program.

3.4 Tool Development

In cases where it is appropriate, a tool may be developed to assist in the annotating process. This tool may either:

- assist in the annotating process,
- verify the correctness of the annotations, or
- perform additional analysis on the compiled output.

If the approach of modifying the LLVM compiler is pursued, developing such a tool to assist the annotating process will be necessary.

Chapter 4

Execution

Experimentation began with the CompCert compiler and the provided assembly annotation tools, outlined in section 4.1. It was found that the CompCert compiler could not handle all cases necessary for the wpif analysis, specifically volatile variables. As a result, the testing moved on to other techniques. Following this, the GNU C extension for inline assembly was explored as a possibility to preserve annotations in C in section 4.3. This technique prevailed and was found to be excellent in handling assembly annotations by injecting comments in to the compiled assembly output. This technique was enhanced by developing a python program to inject inline assembly into the source C files to allow for enhanced analysis and furthermore avoids restricting the program to GNU extension supporting compilers. As a result of the success, modifying the compiler was not explored due to success documented in other research such as the work conducted by Vu et al. [30]. This allowed for further development and improvement of the inline assembly method.

4.1 CompCert AIS Annotations

CompCert is unfortunately not a free tool, however, for research purposes it can be used freely. The specifications of the CompCert install can be seen in Table 4.1.

Testing was initially conducted using the *comment.c* test file. The goal is to propagate the comment down to assembly where it can be used and interpreted. To do so, the

OS Name	Ubuntu 20.04.2 LTS
OS Type	64-bit
Processor	Intel® Core™ i7-6700K CPU @ 4.00GHz × 8
Instruction Set	x86-64
CompCert Version	The CompCert C verified compiler, version 3.7

Table 4.1: CompCert install specifications

comment in the source code needs to be replaced with a call to generate an annotation in the compiled assembly. Fortunately, with the CompCert compiler, this functionality is builtin. This assembly annotation is created through the use of the `__builtin_ais_annot` function described in 2.6. The following builtin annotation was placed in line 2, within the main function in *comment.c*.

```
2      __builtin_ais_annot("%here Critical Comment");
```

Listing 4.1: comment.c

Within this annotation, `%here` is used to represent the location within the program. If the location is not important, `%here` can be omitted. The comment, `"Critical Comment"`, has been included to represent some kind of critical comment that is required to conduct a static analysis on the output. To compile the source to assembler only the following command was used:

```
$ ccomp comment.c -O0 -S
```

Here -O0 is used to specify to perform no optimisations during compilation. The full compiled output can be seen in Appendix D. Below is a snippet of the compiled assembly.

```
.cfi_endproc
.type main, @function
.size main, . - main
.section  "__compcert_ais_annotations","",@note
.ascii "# file:comment.c line:2 function:main\n"
.byte 7,8
.quad .L100
.ascii " Critical Comment\n"
```

Listing 4.2: comment-O0.s

The annotation is stored within assembler directives. Assembler directives are not a part of the processor instruction set, however, are a part of the assembler syntax. Assembler directives all start a period (.). On line 19 a new section has been created, named `"__compcert_ais_annotations"`. Following the declaration of the section is an ascii string, locating the source of the annotation within the source program *comment.c*. Line 23 provides the comment we aimed to preserve with our annotation. Thus, CompCert has shown an initial success in preserving annotations in the form of comments.

Additionally, one major benefit of compCert annotations is that they do not modify the source program, as they are inserted at the end of the program as an assembler directive metadata.

When experimenting with annotated variables, the first issues began to arise. The test file *variable.c* contains several variables with their types to preserve to assembly. The annotations behaved as expected for the types:

- int,
- char,
- short,
- long, and
- any signed or unsigned variations of the above mentioned types.

However, the CompCert annotations does not support floating point types. Upon compiling *variable.c* the following errors were generated.

```
variable.c:13: error: floating point types for parameter '%e1' are not
   supported in ais annotations
variable.c:15: error: floating point types for parameter '%e1' are not
   supported in ais annotations
2 errors detected.
```

This result shows that it is impossible to use the CompCert embedded program annotations for floating point types, vastly restricting its potential use as a technique for a wpif analysis.

It was discovered soon after that the CompCert annotations are unable to handle volatile variables, generating the follow error upon compiling *volatile.c*.

```
volatile.c:4: error: access to volatile variable 'x' for parameter
    '%e1' is not supported in ais annotations
1 error detected.
```

Unfortunately, this result shows that the CompCert AIS annotations approach is not suitable for wpif analysis. The wpif analysis requires use of volatile variables. This is because the primary purpose of the wpif technique is to verify security policy across concurrent programs. Shared variables within concurrent programs can change at any time, and as such it is imperative that shared variables are marked as volatile. As the CompCert AIS annotations cannot handle volatile variables, annotations required for wpif analysis cannot be generated.

Aside from the aforementioned issues, the CompCert AIS annotations performed excellently in generating annotations. The location of global variables in memory are easily identified, as shown in *rooster.c*. The CompCert AIS annotations must be placed within a method and called as if it was its own function. This creates some confusion when dealing with global variables. However, placing annotations on global variables at the start of main is a perfectly valid method of preserving these annotations. As the location of the annotation within the program is no longer important, the `%here` format specifier can be omitted.

```
  .cfi_endproc
  .type main, @function
  .size main, . - main
  .section  "__compcert_ais_annotations","",@note
  .ascii "# file:rooster.c line:6 function:fun\n"
  .byte 7,8
  .quad .L100
  .ascii " CRITICAL COMMENT\n"
  .ascii "# file:rooster.c line:26 function:main\n"
  .byte 7,8
  .quad .L107
  .ascii " L(mem("
  .byte 7,8
```

```
97  .quad goose
98  .ascii ", 4)) = medium\n"
99  .ascii "# file:rooster.c line:27 function:main\n"
100 .byte 7,8
101 .quad .L108
102 .ascii " EXCEPTIONAL\n"
```

Listing 4.3: rooster-O0.s

From *rooster.c*, the comment "CRITICAL COMMENT" has been annotated from lines 88 to 91, and the comment "EXCEPTIONAL" has been annotated from lines 99 to 102. Most notably, the global variable `goose` has been annotated from lines 92 to 98. Reconstructed, the string `"L(mem(goose, 4)) = medium"` has been preserved. Thus, the CompCert annotations can successfully preserve annotations on global variables.

Another interesting problem faced when working with CompCert AIS annotations is found when working with structs. If the programmer wants to annotate a member of a struct for all structs of that type, each instance of that type of struct must be annotated when using CompCert AIS annotations. This is because CompCert treats `__builtin_ais_annot()` as a call to an external function. As such, an annotation cannot be created from outside a method, similar to when dealing with global variables. An example of this process can be seen in *password.c*. Within the program, each instantiation of the struct `user_t` requires another annotation.

```
    user_t* user_admin = malloc(sizeof(user_t));
    strcpy(user_admin->name, "admin");
    strcpy(user_admin->password, "4dm1n__4eva");
    __builtin_ais_annot("%here L(%e1) = high", user_admin->password);
    user_admin->balance = 1000000;

    user_t* user_alice = malloc(sizeof(user_t));
    strcpy(user_alice->name, "alice");
    strcpy(user_alice->password, "!alice12!_veuje@@hak");
    __builtin_ais_annot("%here L(%e1) = high", user_alice->password);
    user_alice->balance = 783;

    user_t* user_abdul = malloc(sizeof(user_t));
    strcpy(user_abdul->name, "abdul");
```

```
    strcpy(user_abdul->password, "passw0rd123");
    __builtin_ais_annot("%here L(%e1) = high", user_abdul->password);
    user_abdul->balance = 2;
```

Listing 4.4: password.c

The compiled output is as expected, with an annotation within the assembly for each of the annotations created within the source file.

```
320  .section  "__compcert_ais_annotations","",@note
321  .ascii "# file:password.c line:20 function:setup_users\n"
322  .byte 7,8
323  .quad .L100
324  .ascii " L((reg(\"rbp\") + 264)) = high\n"
325  .ascii "# file:password.c line:26 function:setup_users\n"
326  .byte 7,8
327  .quad .L101
328  .ascii " L((reg(\"r12\") + 264)) = high\n"
329  .ascii "# file:password.c line:32 function:setup_users\n"
330  .byte 7,8
331  .quad .L102
332  .ascii " L((reg(\"rbx\") + 264)) = high\n"
```

Listing 4.5: password-O0.s

As seen in the assembly annotations, the location of the struct members have been preserved. Line 324 contains the annotation `L((reg("rbp") + 264)) = high`. This annotation notifies that the variable stored in register rbp with an offset of 264 has a security classification of high. Thus, another success for CompCert AIS annotations.

4.1.1 Quality Analysis

To complete a quality analysis, a comparison of the assembly will need to be conducted with and without annotations. The CompCert assembly output can be seen in Appendix C. As we are primarily concerned with the annotated assembly after aggressive optimisations have been performed, the assembly with optimisation level 3 will be compared. To begin with, the assembly for *comment.c* will be compared. Performing a diff on the annotated and non annotated assembly produces the following diff:

```
1 2c2
2 < # Command line: comment.c -S -O3 -o compCert/out/comment-O3.s
3 ---
4 > # Command line: comment.c -O3 -S
5 11a12
6 > .L100:
7 17a19,23
8 >     .section  "__compcert_ais_annotations","",@note
9 >     .ascii "# file:comment.c line:2 function:main\n"
10 >     .byte 7,8
11 >     .quad .L100
12 >     .ascii " Critical Comment\n"
```

Listing 4.6: comment-O3.s diff

The diff explains some interesting differences in the assembly. To begin with, an additional label `.L100:` has been inserted. The only other notable difference is in the compcert annotations. The reason behind the additional label is to allow the location of the annotation to be identified, as can be seen in line 11 of the diff. Thus, this shows a success. There is no difference in the compiled output, even with aggressive compiler optimisations turned on.

Next, *variable-O3.s* will be compared.

```
2c2
< # Command line: variable.c -S -O3 -o compCert/out/variable-O3.s
---
> # Command line: variable.c -S -O3
11a12,16
> .L100:
> .L101:
> .L102:
> .L103:
> .L104:
17a23,43
>     .section  "__compcert_ais_annotations","",@note
>     .ascii "# file:variable.c line:3 function:main\n"
>     .byte 7,8
```

```
15 >    .quad .L100
16 >    .ascii " -10 = int\n"
17 >    .ascii "# file:variable.c line:5 function:main\n"
18 >    .byte 7,8
19 >    .quad .L101
20 >    .ascii " 98 = char\n"
21 >    .ascii "# file:variable.c line:7 function:main\n"
22 >    .byte 7,8
23 >    .quad .L102
24 >    .ascii " -98 = unsigned int\n"
25 >    .ascii "# file:variable.c line:9 function:main\n"
26 >    .byte 7,8
27 >    .quad .L103
28 >    .ascii " 1 = short\n"
29 >    .ascii "# file:variable.c line:11 function:main\n"
30 >    .byte 7,8
31 >    .quad .L104
32 >    .ascii " 4294967296 = long\n"
```

Listing 4.7: variable-O3.s diff

Similar to before, additional labels `.L100:` to `.L104:` have been inserted to identify the annotations from within the source code. However, with aggressive optimisations turned on, an interesting change has occurred to the annotations. As the variables have been optimised out of the compiled assembly, the annotations no longer make sense. Line 16 of the diff shows the annotation for variable `a` from *comment.c*. However, as the variable has been completely optimised out, rather than the location of the register being preserved in the annotation, only the value stored within `a` has been preserved. In this case, that value was `-10`.

Although the annotations do not interfere with the compiler optimisations, the compiler optimisations have rendered the annotations useless. Unfortunately, in cases such as these, there is not much to be done.

Another interesting case arises when a loop is optimised out by the compiler. Take for example *count.c*

```
int main() {
```

```
    int count = 0;
    // here count is always zero
    for(int i = 0; i < count; i++) {
        __builtin_ais_annot("try  loop %here  bound: %e1;", count);
    }

    return 0;
}
```

Listing 4.8: count.c

As count will always be zero, the loop will be optimised out when optimisations are turned on. As CompCert treats `__builtin_ais_annot()` as a call to an external function, it too will be optimised out with aggressive compiler optimisations. Let's compare the assembly with and without compiler optimisations.

```
# File generated by CompCert 3.7
# Command line: count.c -S -O0 -o annotated/count-O0.s
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
  xorl  %ecx, %ecx
  xorl  %eax, %eax
.L100:
  cmpl  %ecx, %eax
  jge .L101
.L102:
  leal  1(%eax), %eax
  jmp .L100
.L101:
  xorl  %eax, %eax
  addq  $8, %rsp
  ret
```

```
24  .cfi_endproc
25  .type main, @function
26  .size main, . - main
27  .section  "__compcert_ais_annotations","",@note
28  .ascii "# file:count.c line:5 function:main\ntry  loop "
29  .byte 7,8
30  .quad .L102
31  .ascii "  bound: reg(\"rcx\");\n"
```

Listing 4.9: count-O0.s

With optimisations turned off, the annotation is preserved, as seen from lines 27-31. Following the annotation, the location of the annotation can be located with label `.L102`. However, with compiler optimisations, it would be expected that `.L100` and `.L102` will be optimised out. This can be seen in *count-O3.s*.

```
# File generated by CompCert 3.7
# Command line: count.c -S -O3 -o annotated/count-O3.s
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
  xorl  %eax, %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
```

Listing 4.10: count-O3.s

As expected, the optimised assembly has completely removed the annotation. In the case of a wpif analysis, this is not a large concern. Although preserving loop invariants is necessary, if the loop is optimised out by the compiler it is no longer of concern and the annotations are no longer necessary.

4.2 CompCert Builtin Annotations

Upon revisiting CompCert AIS annotations, it became apparent that the form of annotations being experimented on previously, namely AIS annotations, were not the only form of C annotations available to CompCert. The CompCert AIS annotations are built on top of the CompCert builtin annotations. The AIS annotations are built primarily for worst case execution time analysis, and thus it was assumed that these annotations would be the most suitable for wpif purposes. However, Similar to the AIS annotations, the CompCert builtin annotations can be called upon using a method call, `__builtin_annot()`. To begin with, annotating `"CRITICIAL COMMENT"` will be tested. The call to `__builtin_annot()` can be seen below.

```
    __builtin_annot("Critical Comment");
```

Listing 4.11: comment.c

Once compiled to assembly, rather than outputting to the bottom of the assembly within assembler directives. Instead, it is placed in the middle of the assembly as a comment. The annotated assembly from the builtin annotation can be seen below.

```
# File generated by CompCert 3.7
# Command line: comment.c -S -O0 -o annotated/comment-O0.s
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
# annotation: "Critical Comment"
  xorl  %eax, %eax
  addq  $8, %rsp
  ret
```

```
16    .cfi_endproc
17    .type main, @function
18    .size main, . - main
```

Listing 4.12: comment-O0.s

On line 12, the annotation has been inserted, with "Critical Comment" preserved. Rather than using the `%here` directive to locate the annotation, the annotation itself has been placed directly within source code in the relevant location. A variable annotation is created in a similar fashion to the CompCert AIS annotations.

```
    __builtin_annot("%1 = int", a);
```

Listing 4.13: variable.c

Within Listing 4.13, the variable a has been annotated with its type. Rather than using `%e1` and so on to reference each element, positional parameters `%1`, `%2`, etc, can be used to locate variables. The annotated assembly is as expected.

```
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
  movl  $-10, %esi
# annotation: "%rsi = int"
  movl  $98, %ecx
# annotation: "%rcx = char"
  negl  %ecx
# annotation: "%rcx = unsigned int"
  movl  $1, %r9d
# annotation: "%r9 = short"
  movabsq $4294967296, %rax
# annotation: "%rax = long"
  movsd .L100(%rip), %xmm1 # 3.14159265358979312
  cvtsd2ss %xmm1, %xmm2
# annotation: "%xmm2 = float"
  cvtss2sd %xmm2, %xmm0
  movsd .L101(%rip), %xmm3 # 2.37840000000000007
```

```
  divsd %xmm3, %xmm0
# annotation: "%xmm0 = double"
```

Listing 4.14: variable-O0.s

Significantly, there is no issues caused by annotating floating point numbers, a surprising success. This suggests that CompCert builtin annotations may able to perform annotations that CompCert AIS annotations cannot. Also worth noting is how this method performs under compiler optimisations. Similar to AIS annotations, when variables are optimised away only their value remains within the annotation.

```
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
# annotation: "-10 = int"
# annotation: "98 = char"
# annotation: "-98 = unsigned int"
# annotation: "1 = short"
# annotation: "4294967296 = long"
# annotation: "3.14159274101257324 = float"
# annotation: "1.32088493988083289 = double"
  leal  134217625(%edi), %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
```

Listing 4.15: variable-O3.s

The primary reason AIS annotations were not appropriate as a method for wpif analysis was because the technique could not handle volatile variables. However, the builtin annotations can. Below is the builtin annotation for a volatile global variable `x`.

```
volatile int x;

int main() {
```

```
    __builtin_annot("L(%1)= false", x);
    return x + 1;
}
```

Listing 4.16: volatile.c

Successfully, the annotation is compiled into the assembly. The annotated assembly can be seen in Listing 4.17.

```
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
  movl  x(%rip), %eax
# annotation: "L(%rax)= false"
  movl  x(%rip), %eax
  leal  1(%eax), %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
```

Listing 4.17: volatile-O3.s

Further testing shows the CompCert builtin annotation method is viable for all purposes for wpif analysis.

4.2.1 Quality Analysis

As the AIS annotations are built on top of the functionality of the CompCert builtin annotations, the results of the quality analysis are very similar. For the majority of annotations observed, the only difference between the annotated and non-annotated assembly was the annotations. Take for example *variable-O0.s*.

```
2c2
< # Command line: variable.c -S -O0 -o compCert/out/variable-O0.s
```

```
---
> # Command line: variable.c -S -O0 -o annotated/variable-O0.s
12a13
> # annotation: "%rsi = int"
13a15
> # annotation: "%rcx = char"
15c17,19
<    movl  $1, %r8d
---
> # annotation: "%rcx = unsigned int"
>    movl  $1, %r9d
> # annotation: "%r9 = short"
16a21
> # annotation: "%rax = long"
18,21c23,28
<    cvtsd2ss %xmm1, %xmm3
<    cvtss2sd %xmm3, %xmm0
<    movsd .L101(%rip), %xmm2 # 2.37840000000000007
<    divsd %xmm2, %xmm0
---
>    cvtsd2ss %xmm1, %xmm2
> # annotation: "%xmm2 = float"
>    cvtss2sd %xmm2, %xmm0
>    movsd .L101(%rip), %xmm3 # 2.37840000000000007
>    divsd %xmm3, %xmm0
> # annotation: "%xmm0 = double"
```

Listing 4.18: variable-O0.s diff

Although it appears as if many changes have been made, the only modifications are to the locations within memory and registers. The last half of the diff has not been included here for spacial purposes.

However, interesting changes occur when working with more complex programs. For example, *volatile-O3.s*.

```
2c2
< # Command line: volatile.c -S -O3 -o compCert/out/volatile-O3.s
---
```

OS Name	Ubuntu 20.04.2 LTS
OS Type	64-bit
Processor	Intel® Core™ i7-6700K CPU @ 4.00GHz × 8
Target	x86_64-pc-linux-gnu
Compiler	clang version 10.0.0-4ubuntu1

Table 4.2: Clang install specifications

```
4 > # Command line: volatile.c -S -O3 -o annotated/volatile-O3.s
5 12a13,14
6 >   movl  x(%rip), %eax
7 > # annotation: "L(%rax)= false"
```

Listing 4.19: volatile-O3.s diff

Line 6 of the diff shows and interesting occurrence. An additional move statement has been added. This was behaviour not observed from the CompCert AIS annotations. Testing of programs such as *pread.c* have shown the same results, with extraneous statements inserted. The cause of these additional statements are unknown. However, it is likely the reason AIS annotations did not support volatile variables was because handling compiler optimisations with volatile variables is quite difficult. It is likely the case that because volatile variables have been annotated, the annotations could not be created without reverting some compiler optimisations.

4.3 Inline Assembly

Extended `asm` is a GNU Extension supported by many compilers. As such, it presents itself as an excellent method for preserving annotations to assembly. As it allows programmers to write assembly code within the C program, it provides an opportunity to hook in to this functionality and utilise it for annotation purposes. To begin with, a very simple program will be experimented on to find the limits of inline assembly and to assess if it is fit for wpif analysis purposes. The compiler used was clang, with the install specifications available in Table 4.2. To begin testing, *comment.c* will be used.

Within the assembly, an inline comment can be created by inserting a `#` at the beginning of the line or comment. The goal with this test is to try and preserve the annotation "CRITICAL COMMENT" to assembly by inserting it as a comment within the assembly. The following line was inserted within the main method of *comment.c*. All the source files used for the inline assembly method can be seen in Appendix E.

```
    asm("# CRITICAL COMMENT");
```

Listing 4.20: comment.c

Here, the call to insert inline assembly is treated as a call to a method, similar to CompCert. The first argument takes the `AssemblerTemplate`. The `AssemblerTemplate` is the template used for formatting the input operands, output operands and the goto parameters. In this case only a comment is inserted in assembly, and as such the input operands and output operands parameters are omitted. The full compiled output can be seen in Appendix F.

```
6  main:                                   # @main
7    .cfi_startproc
8  # %bb.0:
9    pushq %rbp
10   .cfi_def_cfa_offset 16
11   .cfi_offset %rbp, -16
12   movq  %rsp, %rbp
13   .cfi_def_cfa_register %rbp
14   movl  $0, -4(%rbp)
15   #APP
16   # CRITICAL COMMENT
17   #NO_APP
18   xorl  %eax, %eax
19   popq  %rbp
20   .cfi_def_cfa %rsp, 8
21   retq
```

Listing 4.21: comment-O0.s

Lines 15 to 17 show the annotation preserved within the assembly. As can be seen, the compiler does not understand the extent of the assembly provided to it. As such, it treats it as a 'black box', and inserts it where relevant within the assembly instructions. In comparison to CompCert its placement within the assembly is inconvenient. As it is not neatly packaged at the bottom of the assembly it must be parsed out. To distinguish assembly comments from important annotations, the string `"annotation:  "` will be inserted before any annotations to allow for easier parsing. Thus, the annotation described from *comment.c* would be inserted as:

```
    asm ("# annotation: CRITICAL COMMENT");
```

Following, the compiled output would be:

```
#APP
# annotation: CRITICAL COMMENT
#NO_APP
```

One may suggest using the `APP` and `NO_APP` comments above and below our annotation to identify it, however, these comments are system and compiler dependent. As such, a more robust solution has been used here.

Following this success, annotating the location of a variable is the next challenge. With extended `asm` input and output operands are available to allow programmers to work with variables. The goal is to use these mechanisms to identify the location of variables alongside their annotated data. To begin with, the location of a simple integer will be attempted to be preserved. The format of the extended asm is as follows:

```
asm asm-qualifiers ( AssemblerTemplate
    : OutputOperands
    : InputOperands
    : Clobbers
    : GotoLabels)
```

Where `asm-qualifiers`, `OutputOperands`, `InputOperands`, `Clobbers` and `GotoLabels` are optional parameters. In this experimentation, asm-qualifiers will not be used as any inline assembly generated will not modify the value of any variables nor jump to any labels.

Previously the `AssemblerTemplate` was used as a string. However, the string can be templated to locate the value of a variable. Take the following example:

```
asm("# annotation: %0 = int" : "=m"(a))
```

Here `%0` is used to refer to the first output operand, a. Output constraints must begin with either a `'='` or a `'+'`. A constraint beginning with a `'='` is for variable overwriting an existing value, whereas a `'+'` is used when reading and writing. Constraints are used to specify what operands are permitted. In this case, `'m'` is used, signifying a memory operand is allowed, with any kind of address the machine supports. Compiling the inline assembly with no optimisations creates the following annotation.

```
#APP
# annotation: -20(%rbp) = int
#NO_APP
```

Listing 4.22: variable-O0.s

This annotation was created without modifying any additional assembly instructions, seemingly a success. However, when optimisations are turned on an interesting result occurs.

```
main:                                   # @main
  .cfi_startproc
# %bb.0:
                                        # kill: def $edi killed $edi def $rdi
  movl  $-10, -4(%rsp)
  #APP
  # annotation: -4(%rsp) = int
  #NO_APP
  movl  -4(%rsp), %eax
  addl  %edi, %eax
  addl  $134217635, %eax        # imm = 0x7FFFFA3
  retq
```

Listing 4.23: variable-O3.s

Although the annotation is preserved to assembly, a number of optimisations performed by the compiler have been reverted to allow the location of the variable `a` to be preserved. This is because the compiler does not understand what the inline assembly does outside of what information has been provided to it. In this case, the compiler was provided inline assembly that used some kind of memory operand with the intention to overwrite it. Thus, the compiler was required to remove optimisations to allow the the rewritten value of `a` to be propagated through the program successfully.

This appears like a failure of the inline assembly method, however, constructing the statement differently should allow for less modification of the assembly by the compiler. Rather than instructing the compiler that the assembly overwrites the existing value, `'+'` can be used to instruct it that our assembly reads and writes. The following statement is inserted in the C source.

```
asm("# annotation: %0 = int" : "+m"(a))
```

However, this results in the same outcome as Listing 4.23. The issue occurring here is caused due to the output operand. As the inline `asm` is notifying the compiler that the value is modified, it removes optimisations to allow this. Instead, using an input operand may allow the optimisation to run, as instead the compiler has been notified that only the value is being read. The following `asm` is inserted in *variable.c*.

```
asm("# annotation: %0 = int" : : "m"(a))
```

The `'+'` constraint has been removed from the operand as it no longer applies to an input operand. Listing 4.24 shows the assembly with no optimisations.

```
23  movl   $0, -4(%rbp)
24  movl   %edi, -8(%rbp)
25  movq   %rsi, -16(%rbp)
26  movl   $-10, -20(%rbp)
27  #APP
28  # annotation: -20(%rbp) = int
29  #NO_APP
30  movsd .LCPI0_0(%rip), %xmm0    # xmm0 = mem[0],zero
31  movss .LCPI0_1(%rip), %xmm1    # xmm1 = mem[0],zero,zero,zero
32  xorl   %eax, %eax
```

Listing 4.24: variable-O0.s

Here, the annotation can be seen in line 28, with the location of the variable successfully identified. However, interestingly the instructions from lines 30-32 were previously above the instructions from lines 23-26. Although this makes no difference to the runtime of the program, it is an interesting change caused by the inline `asm`.

Additionally, the optimised assembly using the input operands has the same issue as before from Listing 4.23.

```
    main:                                   # @main
        .cfi_startproc
    # %bb.0:
                                            # kill: def $edi killed
   $edi def $rdi
        movl   $-10, -4(%rsp)
        #APP
        # annotation: -4(%rsp) = int
        #NO_APP
        movl   -4(%rsp), %eax
        addl   %edi, %eax
        addl   $134217635, %eax        # imm = 0x7FFFFA3
        retq
```

Listing 4.25: variable-O3.s

Further investigation revealed that the issue arises as the inline `asm` specifies the location of the operand must be in memory with the constraint `"m"(a)`. Rather than limiting the location to memory, allowing any operand available to be used allows for more optimisations to be performed by the compiler. As such, the following line of `asm` was tested.

```
asm("# annotation: %0 = int" : : "X"(a))
```

Here the constraint `"X"(a)` specifies that any operand whatsoever is allowed. The resulting optimised assembly is as follows.

```
  .text
  .file "variable.c"
  .section  .rodata.cst4,"aM",@progbits,4
  .p2align  2                 # -- Begin function main
.LCPI0_0:
  .long 1078530011             # float 3.14159274
  .section  .rodata.cst8,"aM",@progbits,8
  .p2align  3
.LCPI0_1:
  .quad 4608627556095693531    # double 1.3208849398808329
```

```
	.text
	.globl	main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
                                        # kill: def $edi killed $edi def $rdi
	#APP
	# annotation: $-10 = int
	#NO_APP
	#APP
	# annotation: $98 = char
	#NO_APP
	#APP
	# annotation: $-98 = unsigned int
	#NO_APP
	#APP
	# annotation: $1 = short
	#NO_APP
	#APP
	# annotation: $4294967296 = long
	#NO_APP
	movss	.LCPI0_0(%rip), %xmm0   # xmm0 = mem[0],zero,zero,zero
	#APP
	# annotation: %xmm0 = float
	#NO_APP
	movsd	.LCPI0_1(%rip), %xmm0   # xmm0 = mem[0],zero
	#APP
	# annotation: %xmm0 = double
	#NO_APP
	leal	134217625(%rdi), %eax
	retq
.Lfunc_end0:
	.size	main, .Lfunc_end0-main
	.cfi_endproc
                                        # -- End function
	.ident	"clang version 10.0.0-4ubuntu1 "
```

```
49 	.section	".note.GNU-stack","",@progbits
50 	.addrsig
```

Listing 4.26: variable-O3.s

Each of the different types of annotations were additionally annotated using the same method. Some interesting results occur from this annotation technique. To begin with, because all variables have been optimised away, the location of the simple variables have instead been replaced with their value. For example, line 20 shows that the value of -10 was stored in an integer. This behaviour is identical to that observed by the CompCert annotations. Additionally, an interesting scenario occurs when working with floating point numbers. Although they too have been fully optimised out, because the inline `asm` has required reading their values, their value has been placed within a register designated for floating point arithmetic.

The next goal was to attempt preserving annotations from more complex variables. To begin with, a volatile, global variable will be annotated. The test file *volatile.c* was experimented on.

```c
1 volatile int x;
2 
3 int main() {
4     asm("# annotation: %0 = High" : : "X"(x));
5     return x + 1;
6 }
```

Listing 4.27: volatile.c

On line 4, an `asm` statement has been inserted, containing the same format and information as with *variable.c*. However, the variable x is instead a volatile variable. The annotated assembly with full optimisation is listed below.

```
	.text
	.file	"volatile.c"
	.globl	main                    # -- Begin function main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
```

```
8  # %bb.0:
9    movl  x(%rip), %eax
10   #APP
11   # annotation: %eax = High
12   #NO_APP
13   movl  x(%rip), %eax
14   addl  $1, %eax
15   retq
16 .Lfunc_end0:
17   .size main, .Lfunc_end0-main
18   .cfi_endproc
19                                         # -- End function
20   .type x,@object               # @x
21   .comm x,4,4
22   .ident  "clang version 10.0.0-4ubuntu1 "
23   .section  ".note.GNU-stack","",@progbits
24   .addrsig
25   .addrsig_sym x
```

Listing 4.28: volatile-O3.s

Line 11 successfully shows the annotation referencing the global variable `x`. However, an additional move statement has been inserted on line 9. This additional move statement was quite puzzling, however, it is inserted due to the nature of the inline `asm`. As the compiler does not know or understand the assembly created by the programmer, additional move statements may need to be inserted by the compiler. By working backwards, the location of `x` can be located within memory at the location `x(%rip)`. Because the constraint `'X'` has been provided, any operand whatsoever is permitted. As a result, the compiler chose a register as the solution to fill the annotation's requirements. As such, an unnecessary move statement has been inserted. In such a case, the programmer should observe the result and update the `asm` restraint to only allow memory locations. Doing so results in the preferred behaviour with only the annotation inserted into the assembly.

This form of inline `asm` was used to test annotations across the remaining test files. The annotated assembly can be viewed in Appendix F.

4.3.1 Quality Analysis

Although the annotated assembly does contain additional move statements, the program still behaves as expected. However, it does result in heightened difficulty to parse the true location of the variable. For a large program with many annotations, systematically identifying where each variable is stored and modifying the constraints until move statements are eliminated is time consuming and impractical. Therefore, an analysis of the program with the additional move statements preserved will be performed. One proposed method of handling this issue is to build a tool to assist in this parsing and analysis. This tool is covered in section 4.3.2.

To analyse the affect of these additional move statements inserted, big O notation will be used. From the comprehensive testing done, no compiler optimisations were removed when annotations were addeded. Thus, for this analysis, it will be assumed that no optimisations will be reverted when annotations are added. Let $A(n)$ be the function describing the annotated assembly, and $B(n)$ be the function describing the non-annotated assembly. Then,

$$B(n) \in \Theta(h(n))$$

As move statements are constant, any move statement is an element of $\Theta(1)$. Thus, for m annotations added to the program,

$$A(n, m) \in \Theta(m \cdot h(n))$$

As can be seen, in the worst case, the program runtime increases linearly with each annotation added. For time critical or concurrent programs, this is unacceptable. Ideally, a static analysis should be possible without slowing the program for each annotation added. The added move statements are only used to perform the analysis and to identify the location of variables within the program. Therefore, one should be able to compile the program with and without annotations. Assuming all optimisations are performed with annotations turned on, an analysis conducted on a program will still be valid when annotations are turned off. As such, a tool can be created to assist in this annotation and analysis process.

4.3.2 Tool Assisted Annotations

To allow for the program to compile without annotations, the annotations need to be stored where they cannot affect the program output. Littering the program with extended `asm` statements results in a difficult and tedious process of removing them once the program is ready to be compiled. Instead, annotations can be written by the programmer using inline comments. As these comments will be ignored by the compiler, the program can be compiled on any compiler supporting their C version. To reference a variable within the annotation, the variable name can be wrapped in a `var` keyword. An example of an annotated file can be seen below in *pread.c*.

```
volatile int z;
volatile int x;

int main() {
    // security policies:
    // annotation: L(var(z)) = true
    // annotation: L(var(x)) = var(z) % 2 == 0
    // annotation: var(x) < var(x)'
    int r1 = 0;
    int r2 = 0;
    // annotation: _P_0: var(r1) % 2 == 0
    // annotation: _Gamma_0: var(r1) -> LOW, var(r2) -> LOW
    // annotation: L(var(r2)) = false

    while(1) {
        do {
            // annotation: _invariant: var(r1) % 2 == 0 /\ var(r1) <= z
            // annotation: _Gamma: var(r1) -> LOW, var(r2) -> (var(r1)
  == var(z)), var(z) -> LOW
            do {
                // annotation: _invariant: var(r1) <= var(z)
```

```
21                  // annotation: _Gamma: var(r1) -> LOW
22                  r1 = z;
23              } while (r1 %2 != 0);
24              r2 = x;
25          } while (z != r1);
26      }
27      return r2;
28  }
```

Listing 4.29: pread.c

Annotations have been listed with an inline comment beginning with `// annotation:`. Following the declaration of a variable, the annotation can be listed in whatever format the programmer prefers. Line 6 shows the security policy for the variable `z`, denoting that its security policy is always high. Whereas, on line 7, the security policy of the variable `x` is dependant on the value of variable `z`. Also shown in this example are all the necessary annotations required for a wpif analysis.

The goal of the tool is to preserve these annotations stored in these comments. The approach for this technique is to parse the annotation comments from the source file. Once parsed, these comments can be converted to extended `asm` calls in the source file and recompiled. The annotated and non-annotated sources can then be compared to reconstruct the location of variables.

To develop this tool, a python program *annotator.py* was created. The program takes in three arguments;

- the file to compile,
- the location for the annotated output, and
- the optimisation level to compile at.

The program then creates a clone of the source file to modify. The clone is compiled to create an assembly output without annotations. This file is stored in a temporary file until it is ready to be used. The cloned source file is then transpiled. All annotations are located and transformed into extended `asm`. Any special characters that would break extended asm rules are appropriately escaped to allow for their preservation to assembly. This code is then injected into the cloned source file, ready for recompilation. An example of a transpiled source file can be seen in Listing 4.30.

```
volatile int z;
volatile int x;

int main() {
    // security policies:
asm("# annotation: L(%0) = true" : : "X"(z));
asm("# annotation: L(%0) = %1 %% 2 == 0" : : "X"(x), "X"(z));
asm("# annotation: %0 < %1'" : : "X"(x), "X"(x));
    int r1 = 0;
    int r2 = 0;
asm("# annotation: _P_0: %0 %% 2 == 0" : : "X"(r1));
asm("# annotation: _Gamma_0: %0 -> LOW, %1 -> LOW" : : "X"(r1), "X"(r2));
asm("# annotation: L(%0) = false" : : "X"(r2));

    while(1) {
        do {
asm("# annotation: _invariant: %0 %% 2 == 0 /\\ %1 <= z" : : "X"(r1), "X"(r1));
asm("# annotation: _Gamma: %0 -> LOW, %1 -> (%2 == %3), %4 -> LOW" : : "X"(r1), "X"(r2), "X"(r1), "X"(z), "X"(z));
            do {
asm("# annotation: _invariant: %0 <= %1" : : "X"(r1), "X"(z));
asm("# annotation: _Gamma: %0 -> LOW" : : "X"(r1));
                r1 = z;
            } while (r1 %2 != 0);
            r2 = x;
        } while (z != r1);
    }
    return r2;
}
```

Listing 4.30: pread-transpiled.c

As can be seen, each of these annotations follow the annotation style developed earlier in this section. Each reference of a `var` has been appropriately replaced with a variable constraint and corresponding operand. Once again, the transpiled source file is compiled to assembly, and the results of the assembly are compared against the non-annotated counterpart. If the only difference in each of the assembly outputs is the annotations, the process is complete and the program exits. However, if there is a difference, the programmer is notified and the difference of the two compiled sources are listed.

The annotator program uses the `clang` compiler, however, because of the decoupled nature of the program, any compiler supporting extended `asm` could be used by the program.

4.4 LLVM Compiler Modification

The final technique of modifying the LLVM compiler was not experimented on. This was primarily due to two reasons. To begin with, the primary objective of this thesis is to explore techniques that do not modify the compiler, and instead work alongside the functionality of the compiler to preserve annotations. It is well known and documented that modifying the compiler to preserve annotations is possible and successful, as in the case of Vu et al. [30] Additionally, earlier success through the technique of using inline assembly allowed for more time to be allocated to exploring and improving this technique, as seen in 4.3. Therefore, evaluating compiler modification for static analysis purposes was not performed in this research.

Chapter 5

Conclusions and Future Work

In conclusion, the analysed methods of preserving annotations in C have yielded some interesting results. To begin with, CompCert AIS annotations were experimented with to analyse their effectiveness for preserving annotations. The AIS annotations were excellent as they were quite robust and did not interfere with any compiler optimisations. However, this did come with a downside. The method did not support volatile variables. Unfortunately, this excludes it from possibly being used as an annotation technique for concurrent programs. As concurrent programs rely on volatile variables, being unable to annotate the location of these variables excludes it as a possibility. Additionally, however, not so importantly, it does not support floating point numbers either. For programmers wishing to annotate their code that do not require annotations for volatile or floating point numbers, CompCert AIS annotations is an excellent choice. However, one must also consider the downside associated with being locked-in the the CompCert compiler.

For all three methods analysed of preserving annotations, they were treated as a call to an external function. Thus, any annotations placed within dead code was optimised out and removed with compiler optimisations turned on, and no annotations could be placed at the top-level of a program alongside a variable. As such, this has been left for future work. Additionally, it was a consistent theme across all methods for variable annotations to lose meaning when variables have been optimised away. In all three cases, when the variable has been optimised out, references to the variable are replaced with its value instead. This is an interesting situation, as although we want all compiler optimisations to run, losing a variable does increase the difficultly of performing a security analysis.

The CompCert builtin annotations were also explored as a method of preserving annotations. It performs similarly to that of the CompCert AIS annotations, however, is not robust like the AIS annotations were. Although it does support volatile variables and floating point numbers, it also adds additional assembly instructions to the compiled output. These additional instructions are likely added as compiler optimisations are sometimes unable to be performed due to the annotations. Unfortunately, these additional assembly instructions do slow the program and thus are an unfortunate side-effect of the builtin CompCert annotations.

Finally, the inline assembly method of annotating assembly showed a lot of promise. The annotations worked for all necessary components of a static analysis of a concurrent program, however, they did add additional move statements. These move statements could be backtraced to find the true location of the variable, however, they slowed the program linearly with the number of annotations added. Thus, to circumvent this issue, a program was successfully developed to assist in the annotation process without preventing compiler optimisations. The program transpiles the source code and extracts annotations compiled into the assembly. However, it does not properly locate variables from the additional move statements. This is because doing so would require parsing the assembly, which is machine dependent. Thus this has been left for future work.

5.1 Performance Review

From the initial plan set out in this thesis's proposal, the stated schedule was not followed closely. The initial plan had 5 separate stages of investigation, evaluation, development, testing and formalisation, with 30 hours, 10 hours, 100 hours, 30 hours and 30 hours allocated respectively. Notably, in this initial plan much of the allocated time was given to developing some kind of tool and testing it, however, far less time was spent doing so than originally expected. This was primarily because the final technique of modifying the compiler was not explored. Earlier successes with inline assembly annotations allowed for more time to be dedicated to the annotation technique.

One major aspect of the disproportion of time estimated was due to the underestimation of the time it would take to evaluate and understand the results and findings of the assembly. As I have not worked with assembly in the part, a large portion of my time was spent learning to understand and follow the results found.

Additionally, the time required to formalise the results was vastly underestimated. 30 hours of time was allocated, however, much more time was spent writing this paper. This is likely because I have not formalised the results of such a large amount of work in the past. Most of the writing was left until the final two weeks. As a result, a large amount of time was spent returning to work completed in the previous semester and re-evaluating the results to include in this report.

In regards to the goals of the thesis, they were largely met. Several methods were discovered for preserving annotations in assembly for a static analysis of concurrent programs. However, modifying the compiler was not explored as much as I had originally hoped. However, this is due to the large successes formulated from analysing the inline assembly method. There is still work to be completed to extract the location of variables from annotations using the inline assembly method, however, the information is successfully preserved to assembly.

Developing a methodology and approach for assessing each compilation technique drastically helped achieve the goals of this thesis. Before developing a testing and analysis plan, the work conducted was unsatisfactory and had no clear objective or path forwards. However, once a plan and approach was in place the thesis work conducted had a clear goal of what was to be preserved in annotations and how to analyse and assess the results.

Bibliography

[1] Pieter Agten et al. “Recent developments in low-level software security”. In: *IFIP International Workshop on Information Security Theory and Practice*. Springer. 2012, pp. 1–16.

[2] Musard Balliu. “Logics for information flow security: from specification to verification”. PhD thesis. KTH Royal Institute of Technology, 2014.

[3] Nick Benton. “Simple relational correctness proofs for static analyses and program transformations”. In: *ACM SIGPLAN Notices* 39.1 (2004), pp. 14–25.

[4] *CompCert - The CompCert C compiler*. Accessed: 2020-09-01. 2020. URL: `http://compcert.inria.fr/compcert-C.html`.

[5] *Compiler Optimization Removal or Modification of Security-critical Code*. Accessed: 2020-09-01. 2008. URL: `https://cwe.mitre.org/data/definitions/733.html`.

[6] *Compiler Removal of Code to Clear Buffers*. Accessed: 2020-09-01. 2006. URL: `https://cwe.mitre.org/data/definitions/14.html`.

[7] Vijay D’Silva, Mathias Payer, and Dawn Song. “The correctness-security gap in compiler optimization”. In: *2015 IEEE Security and Privacy Workshops*. IEEE. 2015, pp. 73–87.

[8] Dorothy E Denning. “A lattice model of secure information flow”. In: *Communications of the ACM* 19.5 (1976), pp. 236–243.

[9] Dorothy E Denning and Peter J Denning. “Certification of programs for secure information flow”. In: *Communications of the ACM* 20.7 (1977), pp. 504–513.

[10] Quang Do, Ben Martini, and Kim-Kwang Raymond Choo. “The role of the adversary model in applied security research”. In: *Computers & Security* 81 (2019), pp. 156–181.

[11] K Eder, K Georgiou, and N Grech. *Common Assertion Language. ENTRA Project: Whole-Systems Energy Transparency (FET project 318337). Deliverable 2.1.* 2013.

[12] Gidon Ernst and Toby Murray. "SecCSL: Security concurrent separation logic". In: *International Conference on Computer Aided Verification*. Springer. 2019, pp. 208–230.

[13] *File:Simplified Heartbleed explanation.svg - Wikimedia Commons*. Accessed: 2020-09-02. 2014. URL: `https://commons.wikimedia.org/wiki/File:Simplified_Heartbleed_explanation.svg#mediaviewer/File:Simplified_Heartbleed_explanation.svg`.

[14] *GCC Developer Options*. Accessed: 2020-09-02. URL: `https://gcc.gnu.org/onlinedocs/gcc/Developer-Options.html`.

[15] Barbara Guttman and Edward A Roback. *An introduction to computer security: the NIST handbook*. Diane Publishing, 1995.

[16] *Heartbleed Bug*. Accessed: 2020-09-02. 2020. URL: `https://heartbleed.com/`.

[17] Xavier Leroy. "Formal certification of a compiler back-end or: programming a compiler with a proof assistant". In: *Conference record of the 33rd ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. 2006, pp. 42–54.

[18] Xavier Leroy et al. "CompCert-a formally verified optimizing compiler". In: 2016.

[19] Heiko Mantel, Matthias Perner, and Jens Sauer. "Noninterference under weak memory models". In: *2014 IEEE 27th Computer Security Foundations Symposium*. IEEE. 2014, pp. 80–94.

[20] Heiko Mantel, David Sands, and Henning Sudbrock. "Assumptions and guarantees for compositional noninterference". In: *2011 IEEE 24th Computer Security Foundations Symposium*. IEEE. 2011, pp. 218–232.

[21] Toby Murray, Robert Sison, and Kai Engelhardt. "COVERN: A logic for compositional verification of information flow control". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 16–30.

[22] *Options That Control Optimization*. Accessed: 2020-09-01. URL: `https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html`.

[23] Sundeep Prakash, Yann-Hang Lee, and Theodore Johnson. "Non-blocking algorithms for concurrent data structures". MA thesis. Citeseer, 1991.

[24] Andrei Sabelfeld and Andrew C Myers. "Language-based information-flow security". In: *IEEE Journal on selected areas in communications* 21.1 (2003), pp. 5–19.

[25] Bernhard Schommer et al. "Embedded program annotations for WCET analysis". In: *18th International Workshop on Worst-Case Execution Time Analysis (WCET 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik. 2018.

[26] Laurent Simon, David Chisnall, and Ross Anderson. "What you get is what you C: Controlling side effects in mainstream C compilers". In: *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2018, pp. 1–15.

[27] Graeme Smith, Nicholas Coughlin, and Toby Murray. "Value-Dependent Information-Flow Security on Weak Memory Models". In: *International Symposium on Formal Methods*. Springer. 2019, pp. 539–555.

[28] William Stallings et al. *Computer security: principles and practice*. Pearson Education Upper Saddle River, NJ, USA, 2012.

[29] Jeffrey A Vaughan and Todd Millstein. "Secure information flow for concurrent programs under total store order". In: *2012 IEEE 25th Computer Security Foundations Symposium*. IEEE. 2012, pp. 19–29.

[30] Son Tuan Vu et al. "Secure delivery of program properties through optimizing compilation". In: *Proceedings of the 29th International Conference on Compiler Construction*. 2020, pp. 14–26.

[31] Kirsten Winter, Graeme Smith, and Nicholas Coughlin. "Information flow security in the presence of fine-grained concurrency". Unpublished. Aug. 2020.

Appendices

Appendix A

Test C Programs

A.1 comment.c

```
int main() {
    // Critical Comment
    return 0;
}
```

A.2 variable.c

```
int main(int argc, char* argv[]) {
    // a = int
    // b = char
    // c = unsigned int
    // d = short
    // e = long
    // x = float
    // y = double
    int a = -10;
    char b = 'b';
    unsigned int c = -b;
    short d = 0x1;
    long e = 4294967296;
    float x = 3.141592653589793;
```

```
    double y = x / 2.3784;
    return (int)(e / 32) + (int)a + (int)c + (int)d + (int) x + (int)
  y + argc;
}
```

A.3 volatile.c

```
volatile int x;

int main() {
    // L(x) = High
    return x + 1;
}
```

A.4 loop.c

```
int z;
int x;

// security policies
// {L(z)=true}
// {L(x)=z % 2 == 0}

// predicates on initial state
// {_P_0: r1 % 2 == 0}
// {_Gamma_0: r1 -> LOW, r2 -> LOW}

int main() {
    int r1 = 0;
    // {L(r2)=False}
    int r2 = 0;

    while(1) {
    do {
        // {_invariant: r1 % 2 == 0 /\ r1 <= z}
        // {_Gamma: r1 -> LOW, r2 -> (r1 == z), z -> LOW}
        do {
            // {_invariant: r1 <= z}
```

```
            // {_Gamma: r1 -> LOW}
            r1 = z;
        } while (r1 %2 != 0);
            r2 = x;
        } while (z != r1);
    }
    return r2;
}
```

A.5 rooster.c

```
int rooster;
int drake;
// MEDIUM
int goose;

int fun(int a, int b, int c) {
    // CRITICAL COMMENT
    static int count = 0;
    int sum = a + b + c;
    if (sum < 0) {
        return sum;
    }
    if (a < b && b < c) {
        while (a != b) {
            a++;
            count++;
            while (b != c) {
                c--;
                count++;
            }
        }
    }
    return count;
}

int main(void) {
    // EXCEPTIONAL
```

```c
    rooster = 1;
    drake = 5;
    goose = 10;
    int result;
    result = fun(rooster,drake,goose);
    return 0;
}
```

A.6 password.c

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

#define BUFF_LEN 256

typedef struct user_t user_t;

struct user_t {
    user_t* next;
    char name[BUFF_LEN];
    // L(password) = High
    char password[BUFF_LEN];
    size_t balance;
};

user_t* setup_users() {
    user_t* user_admin = malloc(sizeof(user_t));
    strcpy(user_admin->name, "admin");
    strcpy(user_admin->password, "4dm1n__4eva");
    user_admin->balance = 1000000;

    user_t* user_alice = malloc(sizeof(user_t));
    strcpy(user_alice->name, "alice");
    strcpy(user_alice->password, "!alice12!_veuje@@hak");
    user_alice->balance = 783;

    user_t* user_abdul = malloc(sizeof(user_t));
```

```c
    strcpy(user_abdul->name, "abdul");
    strcpy(user_abdul->password, "passw0rd123");
    user_abdul->balance = 2;

    user_admin->next = user_alice;
    user_alice->next = user_abdul;
    user_abdul->next = NULL;

    return user_admin;
}

void print_users(user_t* users) {
    printf("--- USERS ---\n");
    size_t count = 0;
    while (users != NULL) {
        printf(" %021d. %s\n", ++count, users->name);
        users = users->next;
    }
    printf("\n");
}

user_t* getUser(user_t* user_list, char* name) {
    while (user_list != NULL) {
        if (strcmp(user_list->name, name) == 0) {
            return user_list;
        }
        user_list = user_list->next;
    }
    return NULL;
}

int main() {
    user_t* users = setup_users();

    printf("Welcome to BigBank Australia!\n");

    char username[BUFF_LEN];
    printf("Username: ");
    scanf("%255s", username);
```

```c

    user_t* user = getUser(users, username);
    if (user == NULL) {
        printf("User < %s > does not exist.\n", username);
        return 0;
    }

    char password[BUFF_LEN];
    printf("Password: ");
    scanf("%255s", password);
    if (strcmp(user->password, password) != 0) {
        printf("ERROR: incorrect password\n");
        return 0;
    }

    printf("Logged in as < %s >!\n", user->name);
    printf("\n");
    printf("Welcome, %s!\n", user->name);
    printf("Your balance: $%ld\n", user->balance);
}
```

A.7 deadStoreElimination.c

```c
int deadStore(int i, int n) {
    int key = 0xabcd;
    // L(key) = high

    // do some work
    int result = 0;
    while (i > n) {
        result += key;
        i--;
    }

    // clear out our secret key
    key = 0;
    return i + n;
}
```

```

int main(int argc, char *argv[]) {
    deadStore(argc, 2);
}
```

A.8 pread.c

```
volatile int z;
volatile int x;

// security policies
// {L(z)=true}
// {L(x)=z % 2 == 0}

// predicates on initial state
// {_P_0: r1 % 2 == 0}
// {_Gamma_0: r1 -> LOW, r2 -> LOW}

int main() {
    int r1 = 0;
    // {L(r2)=False}
    int r2 = 0;

    while(1) {
    do {
        // {_invariant: r1 % 2 == 0 /\ r1 <= z}
        // {_Gamma: r1 -> LOW, r2 -> (r1 == z), z -> LOW}
        do {
            // {_invariant: r1 <= z}
            // {_Gamma: r1 -> LOW}
            r1 = z;
        } while (r1 %2 != 0);
            r2 = x;
        } while (z != r1);
    }
    return r2;
}
```

Appendix B

CompCert Annotated C Programs

B.1 comment.c

```c
int main() {
    __builtin_ais_annot("%here Critical Comment");
    return 0;
}
```

B.2 variable.c

```c
int main(int argc, char* argv[]) {
    int a = -10;
    __builtin_ais_annot("%here %e1 = int", a);
    char b = 'b';
    __builtin_ais_annot("%here %e1 = char", b);
    unsigned int c = -b;
    __builtin_ais_annot("%here %e1 = unsigned int", c);
    short d = 0x1;
    __builtin_ais_annot("%here %e1 = short", d);
    long e = 4294967296;
    __builtin_ais_annot("%here %e1 = long", e);
    float x = 3.141592653589793;
    __builtin_ais_annot("%here %e1 = float", x);
    double y = x / 2.3784;
```

```
    __builtin_ais_annot("%here %e1 = double", y);
    return (int)(e / 32) + (int)a + (int)c + (int)d + (int) x + (int)
   y + argc;
}
```

B.3 volatile.c

```
volatile int x;

int main() {
    __builtin_ais_annot("%here L(%e1)= false", x);
    return x + 1;
}
```

B.4 loop.c

```

int z;
int x;

int main() {
    // Security Policies
    __builtin_ais_annot("%here L(%e1) = true", z);
    __builtin_ais_annot("%here L(%e1)= %e2 %% 2 == 0", x, z);
    int r1 = 0;
    int r2 = 0;
    __builtin_ais_annot("%here L(%e1)= false", r2);

    // Predicates on initial state
    __builtin_ais_annot("%here _P_0: %e1 %% 2 == 0", r1);
    __builtin_ais_annot("%here _Gamma_0: %e1 -> LOW, %e2 -> LOW", r1,
   r2);

    while(1) {
    do {
        __builtin_ais_annot("%here _invariant: %e1 %% 2 == 0 & %e1 <=
   %e2", r1, z);
```

```
        __builtin_ais_annot("%here _Gamma: %e1 -> LOW, %e2 -> (%e1 ==
  %e3), %e3 -> LOW", r1, r2, z);
        do {
            __builtin_ais_annot("%here _invariant: %e1 <= %e2", r1, z);
            __builtin_ais_annot("%here _Gamma: %e1 -> LOW", r1);
            r1 = z;
        } while (r1 %2 != 0);
            r2 = x;
        } while (z != r1);
    }
    return r2;
}
```

B.5 rooster.c

```
int rooster;
int drake;
int goose;

int fun(int a, int b, int c) {
    __builtin_ais_annot("%here CRITICAL COMMENT");
    static int count = 0;
    int sum = a + b + c;
    if (sum < 0) {
        return sum;
    }
    if (a < b && b < c) {
        while (a != b) {
            a++;
            count++;
            while (b != c) {
                c--;
                count++;
            }
        }
    }
    return count;
}
```

```
int main(void) {
    __builtin_ais_annot("%here L(%e1) = medium", goose);
    __builtin_ais_annot("%here EXCEPTIONAL");
    rooster = 1;
    drake = 5;
    goose = 10;
    int result;
    result = fun(rooster,drake,goose);
    return 0;
}
```

B.6 password.c

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

#define BUFF_LEN 256

typedef struct user_t user_t;

struct user_t {
    user_t* next;
    char name[BUFF_LEN];
    char password[BUFF_LEN];          //          { L(password) = High }
    size_t balance;
};

user_t* setup_users() {
    user_t* user_admin = malloc(sizeof(user_t));
    strcpy(user_admin->name, "admin");
    strcpy(user_admin->password, "4dm1n__4eva");
    __builtin_ais_annot("%here L(%e1) = high", user_admin->password);
    user_admin->balance = 1000000;

    user_t* user_alice = malloc(sizeof(user_t));
```

```
    strcpy(user_alice->name, "alice");
    strcpy(user_alice->password, "!alice12!_veuje@@hak");
    __builtin_ais_annot("%here L(%e1) = high", user_alice->password);
    user_alice->balance = 783;

    user_t* user_abdul = malloc(sizeof(user_t));
    strcpy(user_abdul->name, "abdul");
    strcpy(user_abdul->password, "passw0rd123");
    __builtin_ais_annot("%here L(%e1) = high", user_abdul->password);
    user_abdul->balance = 2;

    user_admin->next = user_alice;
    user_alice->next = user_abdul;
    user_abdul->next = NULL;

    return user_admin;
}

void print_users(user_t* users) {
    printf("--- USERS ---\n");
    size_t count = 0;
    while (users != NULL) {
        printf(" %02ld. %s\n", ++count, users->name);
        users = users->next;
    }
    printf("\n");
}

user_t* getUser(user_t* user_list, char* name) {
    while (user_list != NULL) {
        if (strcmp(user_list->name, name) == 0) {
            return user_list;
        }
        user_list = user_list->next;
    }
    return NULL;
}

int main() {
```

```c
    user_t* users = setup_users();

    printf("Welcome to BigBank Australia!\n");

    char username[BUFF_LEN];
    printf("Username: ");
    scanf("%255s", username);

    user_t* user = getUser(users, username);
    if (user == NULL) {
        printf("User < %s > does not exist.\n", username);
        return 0;
    }

    char password[BUFF_LEN];
    printf("Password: ");
    scanf("%255s", password);
    if (strcmp(user->password, password) != 0) {
        printf("ERROR: incorrect password\n");
        return 0;
    }

    printf("Logged in as < %s >!\n", user->name);
    printf("\n");
    printf("Welcome, %s!\n", user->name);
    printf("Your balance: $%ld\n", user->balance);
}
```

B.7 deadStoreElimination.c

```c
int deadStore(int i, int n) {
    int key = 0xabcd;
    __builtin_ais_annot("%here L(%e1) = high", key);

    // do some work
    int result = 0;
    while (i > n) {
        result += key;
```

```
        i--;
    }

    // clear out our secret key
    key = 0;
    return i + n;
}

int main(int argc, char *argv[]) {
    deadStore(argc, 2);
}
```

B.8 pread.c

```
volatile int z;
volatile int x;

int main() {
    // Security Policies
    __builtin_ais_annot("%here L(%e1) = true", z);
    __builtin_ais_annot("%here L(%e1)= %e2 %% 2 == 0", x, z);

    int r1 = 0;
    int r2 = 0;          //            {L(r2)=False}
    __builtin_ais_annot("%here L(%e1)= false", r2);

    // Predicates on initial state
    __builtin_ais_annot("%here _P_0: %e1 %% 2 == 0", r1);
    __builtin_ais_annot("%here _Gamma_0: %e1 -> LOW, %e2 -> LOW", r1,
    r2);


    while(1) {
        do {
            __builtin_ais_annot("%here _invariant: %e1 %% 2 == 0 & %e1
    <= %e2", r1, z);
            __builtin_ais_annot("%here _Gamma: %e1 -> LOW, %e2 -> (%e1
    == %e3), %e3 -> LOW", r1, r2, z);
```

```
            do {
                __builtin_ais_annot("%here _invariant: %e1 <= %e2",
   r1, z);
                __builtin_ais_annot("%here _Gamma: %e1 -> LOW", r1);
                r1 = z;
            } while (r1 %2 != 0);
                r2 = x;
        } while (z != r1);
    }
    return r2;
}
```

Appendix C

CompCert Assembly Output

C.1 comment-O0.s

```
# File generated by CompCert 3.7
# Command line: comment.c -S -O0 -o compCert/out/comment-O0.s
	.text
	.align	16
	.globl main
main:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
	xorl	%eax, %eax
	addq	$8, %rsp
	ret
	.cfi_endproc
	.type	main, @function
	.size	main, . - main
```

C.2 comment-O3.s

```
# File generated by CompCert 3.7
```

```
# Command line: comment.c -S -O3 -o compCert/out/comment-O3.s
	.text
	.align 16
	.globl main
main:
	.cfi_startproc
	subq $8, %rsp
	.cfi_adjust_cfa_offset 8
	leaq 16(%rsp), %rax
	movq %rax, 0(%rsp)
	xorl %eax, %eax
	addq $8, %rsp
	ret
	.cfi_endproc
	.type main, @function
	.size main, . - main
```

C.3 variable-O0.s

```
# File generated by CompCert 3.7
# Command line: variable.c -S -O0 -o compCert/out/variable-O0.s
	.text
	.align 16
	.globl main
main:
	.cfi_startproc
	subq $8, %rsp
	.cfi_adjust_cfa_offset 8
	leaq 16(%rsp), %rax
	movq %rax, 0(%rsp)
	movl $-10, %esi
	movl $98, %ecx
	negl %ecx
	movl $1, %r8d
	movabsq $4294967296, %rax
	movsd .L100(%rip), %xmm1 # 3.14159265358979312
	cvtsd2ss %xmm1, %xmm3
	cvtss2sd %xmm3, %xmm0
```

```
	movsd .L101(%rip), %xmm2 # 2.3784000000000007
	divsd %xmm2, %xmm0
	cqto
	shrq	$59, %rdx
	leaq	0(%rax,%rdx,1), %rax
	sarq	$5, %rax
	leal	0(%eax,%esi,1), %r9d
	leal	0(%r9d,%ecx,1), %r10d
	leal	0(%r10d,%r8d,1), %r8d
	cvttss2si %xmm3, %edx
	leal	0(%r8d,%edx,1), %r8d
	cvttsd2si %xmm0, %eax
	leal	0(%r8d,%eax,1), %r11d
	leal	0(%r11d,%edi,1), %eax
	addq	$8, %rsp
	ret
	.cfi_endproc
	.type main, @function
	.size main, . - main
	.section	.rodata.cst8,"aM",@progbits,8
	.align	8
.L100:	.quad 0x400921fb54442d18
.L101:	.quad 0x400306f694467382
```

C.4 variable-O3.s

```
# File generated by CompCert 3.7
# Command line: variable.c -S -O3 -o compCert/out/variable-O3.s
	.text
	.align	16
	.globl main
main:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
	leal	134217625(%edi), %eax
```

```
	addq	$8, %rsp
	ret
	.cfi_endproc
	.type main, @function
	.size main, . - main
```

C.5 volatile-O0.s

```
# File generated by CompCert 3.7
# Command line: volatile.c -S -O0 -o compCert/out/volatile-O0.s
	.comm x, 4, 4
	.text
	.align	16
	.globl main
main:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
	movl	x(%rip), %eax
	leal	1(%eax), %eax
	addq	$8, %rsp
	ret
	.cfi_endproc
	.type main, @function
	.size main, . - main
```

C.6 volatile-O3.s

```
# File generated by CompCert 3.7
# Command line: volatile.c -S -O3 -o compCert/out/volatile-O3.s
	.comm x, 4, 4
	.text
	.align	16
	.globl main
main:
	.cfi_startproc
```

```
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
  movl  x(%rip), %eax
  leal  1(%eax), %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
```

C.7 loop-O0.s

```
# File generated by CompCert 3.7
# Command line: loop.c -S -O0 -o compCert/out/loop-O0.s
  .comm z, 4, 4
  .comm x, 4, 4
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
.L100:
  movl  z(%rip), %edx
  movq  %rdx, %rax
  testl %eax, %eax
  leal  1(%eax), %ecx
  cmovl %rcx, %rax
  sarl  $1, %eax
  leal  0(,%eax,2), %esi
  movq  %rdx, %rcx
  subl  %esi, %ecx
  testl %ecx, %ecx
```

```
	jne	.L100
	movl	z(%rip), %esi
	jmp	.L100
	.cfi_endproc
	.type	main, @function
	.size	main, . - main
```

C.8 loop-O3.s

```
# File generated by CompCert 3.7
# Command line: loop.c -S -O3 -o compCert/out/loop-O3.s
	.comm	z, 4, 4
	.comm	x, 4, 4
	.text
	.align	16
	.globl main
main:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
.L100:
	movl	z(%rip), %edx
	movq	%rdx, %rax
	testl	%eax, %eax
	leal	1(%eax), %ecx
	cmovl	%rcx, %rax
	sarl	$1, %eax
	leal	0(,%eax,2), %esi
	movq	%rdx, %rcx
	subl	%esi, %ecx
	testl	%ecx, %ecx
	jne	.L100
	movq	%rdx, %rsi
	jmp	.L100
	.cfi_endproc
	.type	main, @function
```

```
  .size main, . - main
```

C.9 rooster-O0.s

```
# File generated by CompCert 3.7
# Command line: rooster.c -S -O0 -o compCert/out/rooster-O0.s
  .comm rooster, 4, 4
  .comm drake, 4, 4
  .comm goose, 4, 4
  .data
  .align  4
count:
  .long 0
  .type count, @object
  .size count, . - count
  .text
  .align  16
  .globl fun
fun:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
  leal  0(%edi,%esi,1), %r8d
  leal  0(%r8d,%edx,1), %eax
  testl %eax, %eax
  jl   .L100
  cmpl  %esi, %edi
  jl   .L101
  xorl  %r8d, %r8d
  jmp .L102
.L101:
  cmpl  %edx, %esi
  setl  %r8b
  movzbl  %r8b, %r8d
.L102:
  cmpl  $0, %r8d
```

```
  je   .L103
.L104:
  cmpl  %esi, %edi
  je   .L103
  leal  1(%edi), %edi
  movl  count(%rip), %eax
  leal  1(%eax), %ecx
  movl  %ecx, count(%rip)
.L105:
  cmpl  %edx, %esi
  je   .L104
  leal  -1(%edx), %edx
  movl  count(%rip), %r9d
  leal  1(%r9d), %r8d
  movl  %r8d, count(%rip)
  jmp .L105
.L103:
  movl  count(%rip), %eax
.L100:
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type fun, @function
  .size fun, . - fun
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
  movl  $1, %eax
  movl  %eax, rooster(%rip)
  movl  $5, %eax
  movl  %eax, drake(%rip)
  movl  $10, %eax
  movl  %eax, goose(%rip)
```

```
  movl   rooster(%rip), %edi
  movl   drake(%rip), %esi
  movl   goose(%rip), %edx
  call   fun
  xorl   %eax, %eax
  addq   $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
```

C.10 rooster-O3.s

```
# File generated by CompCert 3.7
# Command line: rooster.c -S -O3 -o compCert/out/rooster-O3.s
  .comm rooster, 4, 4
  .comm drake, 4, 4
  .comm goose, 4, 4
  .data
  .align  4
count:
  .long 0
  .type count, @object
  .size count, . - count
  .text
  .align  16
  .globl fun
fun:
  .cfi_startproc
  subq   $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq   16(%rsp), %rax
  movq   %rax, 0(%rsp)
  leal   0(%edi,%esi,1), %r9d
  leal   0(%r9d,%edx,1), %eax
  testl %eax, %eax
  jl   .L100
  cmpl   %edx, %esi
```

```
  setl  %al
  movzbl  %al, %eax
  xorl  %r8d, %r8d
  cmpl  %esi, %edi
  cmovge  %r8, %rax
  cmpl  $0, %eax
  je  .L101
.L102:
  cmpl  %esi, %edi
  je  .L101
  leal  1(%edi), %edi
  movl  count(%rip), %ecx
  leal  1(%ecx), %r8d
  movl  %r8d, count(%rip)
.L103:
  cmpl  %edx, %esi
  je  .L102
  leal  -1(%edx), %edx
  movl  count(%rip), %r10d
  leal  1(%r10d), %r8d
  movl  %r8d, count(%rip)
  jmp .L103
.L101:
  movl  count(%rip), %eax
.L100:
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type fun, @function
  .size fun, . - fun
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
```

```
movl  $1, %eax
movl  %eax, rooster(%rip)
movl  $5, %eax
movl  %eax, drake(%rip)
movl  $10, %eax
movl  %eax, goose(%rip)
movl  $1, %edi
movl  $5, %esi
movl  $10, %edx
call  fun
xorl  %eax, %eax
addq  $8, %rsp
ret
.cfi_endproc
.type main, @function
.size main, . - main
```

C.11 password-O0.s

```
# File generated by CompCert 3.7
# Command line: password.c -S -O0 -o compCert/out/password-O0.s
  .section  .rodata
  .align  1
__stringlit_7:
  .ascii  "--- USERS ---\012\000"
  .type __stringlit_7, @object
  .size __stringlit_7, . - __stringlit_7
  .section  .rodata
  .align  1
__stringlit_6:
  .ascii  "passw0rd123\000"
  .type __stringlit_6, @object
  .size __stringlit_6, . - __stringlit_6
  .section  .rodata
  .align  1
__stringlit_4:
  .ascii  "!alice12!_veuje@@hak\000"
  .type __stringlit_4, @object
```

```
    .size __stringlit_4, . - __stringlit_4
    .section  .rodata
    .align  1
__stringlit_14:
    .ascii  "Password: \000"
    .type __stringlit_14, @object
    .size __stringlit_14, . - __stringlit_14
    .section  .rodata
    .align  1
__stringlit_18:
    .ascii  "Your balance: $%ld\012\000"
    .type __stringlit_18, @object
    .size __stringlit_18, . - __stringlit_18
    .section  .rodata
    .align  1
__stringlit_13:
    .ascii  "User < %s > does not exist.\012\000"
    .type __stringlit_13, @object
    .size __stringlit_13, . - __stringlit_13
    .section  .rodata
    .align  1
__stringlit_8:
    .ascii  " %02ld. %s\012\000"
    .type __stringlit_8, @object
    .size __stringlit_8, . - __stringlit_8
    .section  .rodata
    .align  1
__stringlit_1:
    .ascii  "admin\000"
    .type __stringlit_1, @object
    .size __stringlit_1, . - __stringlit_1
    .section  .rodata
    .align  1
__stringlit_2:
    .ascii  "4dm1n__4eva\000"
    .type __stringlit_2, @object
    .size __stringlit_2, . - __stringlit_2
    .section  .rodata
    .align  1
```

```
__stringlit_3:
  .ascii  "alice\000"
  .type __stringlit_3, @object
  .size __stringlit_3, . - __stringlit_3
  .section  .rodata
  .align  1
__stringlit_11:
  .ascii  "Username: \000"
  .type __stringlit_11, @object
  .size __stringlit_11, . - __stringlit_11
  .section  .rodata
  .align  1
__stringlit_5:
  .ascii  "abdul\000"
  .type __stringlit_5, @object
  .size __stringlit_5, . - __stringlit_5
  .section  .rodata
  .align  1
__stringlit_17:
  .ascii  "Welcome, %s!\012\000"
  .type __stringlit_17, @object
  .size __stringlit_17, . - __stringlit_17
  .section  .rodata
  .align  1
__stringlit_12:
  .ascii  "%255s\000"
  .type __stringlit_12, @object
  .size __stringlit_12, . - __stringlit_12
  .section  .rodata
  .align  1
__stringlit_9:
  .ascii  "\012\000"
  .type __stringlit_9, @object
  .size __stringlit_9, . - __stringlit_9
  .section  .rodata
  .align  1
__stringlit_15:
  .ascii  "ERROR: incorrect password\012\000"
  .type __stringlit_15, @object
```

```
	.size __stringlit_15, . - __stringlit_15
	.section	.rodata
	.align	1
__stringlit_10:
	.ascii	"Welcome to BigBank Australia!\012\000"
	.type __stringlit_10, @object
	.size __stringlit_10, . - __stringlit_10
	.section	.rodata
	.align	1
__stringlit_16:
	.ascii	"Logged in as < %s >!\012\000"
	.type __stringlit_16, @object
	.size __stringlit_16, . - __stringlit_16
	.text
	.align	16
	.globl setup_users
setup_users:
	.cfi_startproc
	subq	$40, %rsp
	.cfi_adjust_cfa_offset	40
	leaq	48(%rsp), %rax
	movq	%rax, 0(%rsp)
	movq	%rbx, 8(%rsp)
	movq	%rbp, 16(%rsp)
	movq	%r12, 24(%rsp)
	movq	$528, %rdi
	call	malloc
	movq	%rax, %rbp
	leaq	8(%rbp), %rdi
	leaq	__stringlit_1(%rip), %rsi
	call	strcpy
	leaq	264(%rbp), %rdi
	leaq	__stringlit_2(%rip), %rsi
	call	strcpy
	movq	$1000000, %rax
	movq	%rax, 520(%rbp)
	movq	$528, %rdi
	call	malloc
	movq	%rax, %r12
```

```
  leaq  8(%r12), %rdi
  leaq  __stringlit_3(%rip), %rsi
  call  strcpy
  leaq  264(%r12), %rdi
  leaq  __stringlit_4(%rip), %rsi
  call  strcpy
  movq  $783, %rsi
  movq  %rsi, 520(%r12)
  movq  $528, %rdi
  call  malloc
  movq  %rax, %rbx
  leaq  8(%rbx), %rdi
  leaq  __stringlit_5(%rip), %rsi
  call  strcpy
  leaq  264(%rbx), %rdi
  leaq  __stringlit_6(%rip), %rsi
  call  strcpy
  movq  $2, %r10
  movq  %r10, 520(%rbx)
  movq  %r12, 0(%rbp)
  movq  %rbx, 0(%r12)
  xorq  %r8, %r8
  movq  %r8, 0(%rbx)
  movq  %rbp, %rax
  movq  8(%rsp), %rbx
  movq  16(%rsp), %rbp
  movq  24(%rsp), %r12
  addq  $40, %rsp
  ret
  .cfi_endproc
  .type setup_users, @function
  .size setup_users, . - setup_users
  .text
  .align  16
  .globl print_users
print_users:
  .cfi_startproc
  subq  $24, %rsp
  .cfi_adjust_cfa_offset  24
```

```
  leaq  32(%rsp), %rax
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  movq  %rbp, 16(%rsp)
  movq  %rdi, %rbx
  leaq  __stringlit_7(%rip), %rdi
  movl  $0, %eax
  call  printf
  xorq  %rbp, %rbp
.L100:
  cmpq  $0, %rbx
  je  .L101
  leaq  1(%rbp), %rbp
  leaq  __stringlit_8(%rip), %rdi
  leaq  8(%rbx), %rdx
  movq  %rbp, %rsi
  movl  $0, %eax
  call  printf
  movq  0(%rbx), %rbx
  jmp .L100
.L101:
  leaq  __stringlit_9(%rip), %rdi
  movl  $0, %eax
  call  printf
  movq  8(%rsp), %rbx
  movq  16(%rsp), %rbp
  addq  $24, %rsp
  ret
  .cfi_endproc
  .type print_users, @function
  .size print_users, . - print_users
  .text
  .align  16
  .globl getUser
getUser:
  .cfi_startproc
  subq  $24, %rsp
  .cfi_adjust_cfa_offset  24
  leaq  32(%rsp), %rax
```

```
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  movq  %rbp, 16(%rsp)
  movq  %rsi, %rbp
  movq  %rdi, %rbx
.L102:
  cmpq  $0, %rbx
  je   .L103
  leaq  8(%rbx), %rdi
  movq  %rbp, %rsi
  call  strcmp
  testl %eax, %eax
  je   .L104
  movq  0(%rbx), %rbx
  jmp .L102
.L103:
  xorq  %rbx, %rbx
.L104:
  movq  %rbx, %rax
  movq  8(%rsp), %rbx
  movq  16(%rsp), %rbp
  addq  $24, %rsp
  ret
  .cfi_endproc
  .type getUser, @function
  .size getUser, . - getUser
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $536, %rsp
  .cfi_adjust_cfa_offset  536
  leaq  544(%rsp), %rax
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  call  setup_users
  movq  %rax, %rbx
  leaq  __stringlit_10(%rip), %rdi
```

```
    movl   $0, %eax
    call   printf
    leaq   __stringlit_11(%rip), %rdi
    movl   $0, %eax
    call   printf
    leaq   __stringlit_12(%rip), %rdi
    leaq   16(%rsp), %rsi
    movl   $0, %eax
    call   __isoc99_scanf
    leaq   16(%rsp), %rsi
    movq   %rbx, %rdi
    call   getUser
    movq   %rax, %rbx
    cmpq   $0, %rbx
    jne .L105
    leaq   __stringlit_13(%rip), %rdi
    leaq   16(%rsp), %rsi
    movl   $0, %eax
    call   printf
    xorl   %eax, %eax
    jmp .L106
.L105:
    leaq   __stringlit_14(%rip), %rdi
    movl   $0, %eax
    call   printf
    leaq   __stringlit_12(%rip), %rdi
    leaq   272(%rsp), %rsi
    movl   $0, %eax
    call   __isoc99_scanf
    leaq   264(%rbx), %rdi
    leaq   272(%rsp), %rsi
    call   strcmp
    testl  %eax, %eax
    je   .L107
    leaq   __stringlit_15(%rip), %rdi
    movl   $0, %eax
    call   printf
    xorl   %eax, %eax
    jmp .L106
```

```
.L107:
  leaq   __stringlit_16(%rip), %rdi
  leaq   8(%rbx), %rsi
  movl   $0, %eax
  call   printf
  leaq   __stringlit_9(%rip), %rdi
  movl   $0, %eax
  call   printf
  leaq   __stringlit_17(%rip), %rdi
  leaq   8(%rbx), %rsi
  movl   $0, %eax
  call   printf
  leaq   __stringlit_18(%rip), %rdi
  movq   520(%rbx), %rsi
  movl   $0, %eax
  call   printf
  xorl   %eax, %eax
.L106:
  movq   8(%rsp), %rbx
  addq   $536, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
```

C.12 password-O3.s

```
# File generated by CompCert 3.7
# Command line: password.c -S -O3 -o compCert/out/password-O3.s
  .section  .rodata
  .align  1
__stringlit_7:
  .ascii  "--- USERS ---\012\000"
  .type __stringlit_7, @object
  .size __stringlit_7, . - __stringlit_7
  .section  .rodata
  .align  1
__stringlit_6:
```

```
  .ascii  "passw0rd123\000"
  .type __stringlit_6, @object
  .size __stringlit_6, . - __stringlit_6
  .section  .rodata
  .align  1
__stringlit_4:
  .ascii  "!alice12!_veuje@@hak\000"
  .type __stringlit_4, @object
  .size __stringlit_4, . - __stringlit_4
  .section  .rodata
  .align  1
__stringlit_14:
  .ascii  "Password: \000"
  .type __stringlit_14, @object
  .size __stringlit_14, . - __stringlit_14
  .section  .rodata
  .align  1
__stringlit_18:
  .ascii  "Your balance: $%ld\012\000"
  .type __stringlit_18, @object
  .size __stringlit_18, . - __stringlit_18
  .section  .rodata
  .align  1
__stringlit_13:
  .ascii  "User < %s > does not exist.\012\000"
  .type __stringlit_13, @object
  .size __stringlit_13, . - __stringlit_13
  .section  .rodata
  .align  1
__stringlit_8:
  .ascii  " %02ld. %s\012\000"
  .type __stringlit_8, @object
  .size __stringlit_8, . - __stringlit_8
  .section  .rodata
  .align  1
__stringlit_1:
  .ascii  "admin\000"
  .type __stringlit_1, @object
  .size __stringlit_1, . - __stringlit_1
```

```
  .section  .rodata
  .align  1
__stringlit_2:
  .ascii  "4dm1n__4eva\000"
  .type __stringlit_2, @object
  .size __stringlit_2, . - __stringlit_2
  .section  .rodata
  .align  1
__stringlit_3:
  .ascii  "alice\000"
  .type __stringlit_3, @object
  .size __stringlit_3, . - __stringlit_3
  .section  .rodata
  .align  1
__stringlit_11:
  .ascii  "Username: \000"
  .type __stringlit_11, @object
  .size __stringlit_11, . - __stringlit_11
  .section  .rodata
  .align  1
__stringlit_5:
  .ascii  "abdul\000"
  .type __stringlit_5, @object
  .size __stringlit_5, . - __stringlit_5
  .section  .rodata
  .align  1
__stringlit_17:
  .ascii  "Welcome, %s!\012\000"
  .type __stringlit_17, @object
  .size __stringlit_17, . - __stringlit_17
  .section  .rodata
  .align  1
__stringlit_12:
  .ascii  "%255s\000"
  .type __stringlit_12, @object
  .size __stringlit_12, . - __stringlit_12
  .section  .rodata
  .align  1
__stringlit_9:
```

```
  .ascii  "\012\000"
  .type __stringlit_9, @object
  .size __stringlit_9, . - __stringlit_9
  .section  .rodata
  .align  1
__stringlit_15:
  .ascii  "ERROR: incorrect password\012\000"
  .type __stringlit_15, @object
  .size __stringlit_15, . - __stringlit_15
  .section  .rodata
  .align  1
__stringlit_10:
  .ascii  "Welcome to BigBank Australia!\012\000"
  .type __stringlit_10, @object
  .size __stringlit_10, . - __stringlit_10
  .section  .rodata
  .align  1
__stringlit_16:
  .ascii  "Logged in as < %s >!\012\000"
  .type __stringlit_16, @object
  .size __stringlit_16, . - __stringlit_16
  .text
  .align  16
  .globl setup_users
setup_users:
  .cfi_startproc
  subq  $40, %rsp
  .cfi_adjust_cfa_offset  40
  leaq  48(%rsp), %rax
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  movq  %rbp, 16(%rsp)
  movq  %r12, 24(%rsp)
  movq  $528, %rdi
  call  malloc
  movq  %rax, %rbp
  leaq  8(%rbp), %rdi
  leaq  __stringlit_1(%rip), %rsi
  call  strcpy
```

```
leaq   264(%rbp), %rdi
leaq   __stringlit_2(%rip), %rsi
call   strcpy
movq   $1000000, %rax
movq   %rax, 520(%rbp)
movq   $528, %rdi
call   malloc
movq   %rax, %r12
leaq   8(%r12), %rdi
leaq   __stringlit_3(%rip), %rsi
call   strcpy
leaq   264(%r12), %rdi
leaq   __stringlit_4(%rip), %rsi
call   strcpy
movq   $783, %rsi
movq   %rsi, 520(%r12)
movq   $528, %rdi
call   malloc
movq   %rax, %rbx
leaq   8(%rbx), %rdi
leaq   __stringlit_5(%rip), %rsi
call   strcpy
leaq   264(%rbx), %rdi
leaq   __stringlit_6(%rip), %rsi
call   strcpy
movq   $2, %r10
movq   %r10, 520(%rbx)
movq   %r12, 0(%rbp)
movq   %rbx, 0(%r12)
xorq   %r8, %r8
movq   %r8, 0(%rbx)
movq   %rbp, %rax
movq   8(%rsp), %rbx
movq   16(%rsp), %rbp
movq   24(%rsp), %r12
addq   $40, %rsp
ret
.cfi_endproc
.type setup_users, @function
```

```
    .size setup_users, . - setup_users
    .text
    .align  16
    .globl print_users
print_users:
    .cfi_startproc
    subq  $24, %rsp
    .cfi_adjust_cfa_offset  24
    leaq  32(%rsp), %rax
    movq  %rax, 0(%rsp)
    movq  %rbx, 8(%rsp)
    movq  %rbp, 16(%rsp)
    movq  %rdi, %rbp
    leaq  __stringlit_7(%rip), %rdi
    movl  $0, %eax
    call  printf
    xorq  %rbx, %rbx
.L100:
    cmpq  $0, %rbp
    je  .L101
    leaq  1(%rbx), %rbx
    leaq  __stringlit_8(%rip), %rdi
    leaq  8(%rbp), %rdx
    movq  %rbx, %rsi
    movl  $0, %eax
    call  printf
    movq  0(%rbp), %rbp
    jmp .L100
.L101:
    leaq  __stringlit_9(%rip), %rdi
    movl  $0, %eax
    call  printf
    movq  8(%rsp), %rbx
    movq  16(%rsp), %rbp
    addq  $24, %rsp
    ret
    .cfi_endproc
    .type print_users, @function
    .size print_users, . - print_users
```

```
	.text
	.align  16
	.globl getUser
getUser:
	.cfi_startproc
	subq  $24, %rsp
	.cfi_adjust_cfa_offset  24
	leaq  32(%rsp), %rax
	movq  %rax, 0(%rsp)
	movq  %rbx, 8(%rsp)
	movq  %rbp, 16(%rsp)
	movq  %rsi, %rbp
	movq  %rdi, %rbx
.L102:
	cmpq  $0, %rbx
	je   .L103
	leaq  8(%rbx), %rdi
	movq  %rbp, %rsi
	call  strcmp
	testl %eax, %eax
	je   .L104
	movq  0(%rbx), %rbx
	jmp .L102
.L103:
	xorq  %rbx, %rbx
.L104:
	movq  %rbx, %rax
	movq  8(%rsp), %rbx
	movq  16(%rsp), %rbp
	addq  $24, %rsp
	ret
	.cfi_endproc
	.type getUser, @function
	.size getUser, . - getUser
	.text
	.align  16
	.globl main
main:
	.cfi_startproc
```

```
  subq   $536, %rsp
  .cfi_adjust_cfa_offset  536
  leaq   544(%rsp), %rax
  movq   %rax, 0(%rsp)
  movq   %rbx, 8(%rsp)
  call   setup_users
  movq   %rax, %rbx
  leaq   __stringlit_10(%rip), %rdi
  movl   $0, %eax
  call   printf
  leaq   __stringlit_11(%rip), %rdi
  movl   $0, %eax
  call   printf
  leaq   __stringlit_12(%rip), %rdi
  leaq   16(%rsp), %rsi
  movl   $0, %eax
  call   __isoc99_scanf
  leaq   16(%rsp), %rsi
  movq   %rbx, %rdi
  call   getUser
  movq   %rax, %rbx
  cmpq   $0, %rbx
  jne .L105
  leaq   __stringlit_13(%rip), %rdi
  leaq   16(%rsp), %rsi
  movl   $0, %eax
  call   printf
  xorl   %eax, %eax
  jmp .L106
.L105:
  leaq   __stringlit_14(%rip), %rdi
  movl   $0, %eax
  call   printf
  leaq   __stringlit_12(%rip), %rdi
  leaq   272(%rsp), %rsi
  movl   $0, %eax
  call   __isoc99_scanf
  leaq   264(%rbx), %rdi
  leaq   272(%rsp), %rsi
```

```
  call   strcmp
  testl %eax, %eax
  je   .L107
  leaq   __stringlit_15(%rip), %rdi
  movl   $0, %eax
  call   printf
  xorl   %eax, %eax
  jmp .L106
.L107:
  leaq   __stringlit_16(%rip), %rdi
  leaq   8(%rbx), %rsi
  movl   $0, %eax
  call   printf
  leaq   __stringlit_9(%rip), %rdi
  movl   $0, %eax
  call   printf
  leaq   __stringlit_17(%rip), %rdi
  leaq   8(%rbx), %rsi
  movl   $0, %eax
  call   printf
  leaq   __stringlit_18(%rip), %rdi
  movq   520(%rbx), %rsi
  movl   $0, %eax
  call   printf
  xorl   %eax, %eax
.L106:
  movq   8(%rsp), %rbx
  addq   $536, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
```

C.13 deadStoreElimination-O0.s

```
# File generated by CompCert 3.7
# Command line: deadStoreElimination.c -S -O0 -o
   compCert/out/deadStoreElimination-O0.s
```

```
    .text
    .align  16
    .globl deadStore
deadStore:
    .cfi_startproc
    subq  $8, %rsp
    .cfi_adjust_cfa_offset  8
    leaq  16(%rsp), %rax
    movq  %rax, 0(%rsp)
.L100:
    cmpl  %esi, %edi
    jle .L101
    leal  -1(%edi), %edi
    jmp .L100
.L101:
    leal  0(%edi,%esi,1), %eax
    addq  $8, %rsp
    ret
    .cfi_endproc
    .type deadStore, @function
    .size deadStore, . - deadStore
    .text
    .align  16
    .globl main
main:
    .cfi_startproc
    subq  $8, %rsp
    .cfi_adjust_cfa_offset  8
    leaq  16(%rsp), %rax
    movq  %rax, 0(%rsp)
    movl  $2, %esi
    call  deadStore
    xorl  %eax, %eax
    addq  $8, %rsp
    ret
    .cfi_endproc
    .type main, @function
    .size main, . - main
```

C.14 deadStoreElimination-O3.s

```
# File generated by CompCert 3.7
# Command line: deadStoreElimination.c -S -O3 -o
    compCert/out/deadStoreElimination-O3.s
  .text
  .align  16
  .globl deadStore
deadStore:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
.L100:
  cmpl  %esi, %edi
  jle .L101
  leal  -1(%edi), %edi
  jmp .L100
.L101:
  leal  0(%edi,%esi,1), %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type deadStore, @function
  .size deadStore, . - deadStore
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
  movl  $2, %esi
  call  deadStore
  xorl  %eax, %eax
  addq  $8, %rsp
```

```
ret
.cfi_endproc
.type main, @function
.size main, . - main
```

C.15 pread-O0.s

```
# File generated by CompCert 3.7
# Command line: pread.c -S -O0 -o compCert/out/pread-O0.s
  .comm z, 4, 4
  .comm x, 4, 4
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
.L100:
  movl  z(%rip), %edx
  movq  %rdx, %rsi
  movq  %rsi, %rax
  testl %eax, %eax
  leal  1(%eax), %ecx
  cmovl %rcx, %rax
  sarl  $1, %eax
  leal  0(,%eax,2), %edi
  subl  %edi, %esi
  testl %esi, %esi
  jne .L100
  movl  x(%rip), %esi
  movl  z(%rip), %esi
  jmp .L100
  .cfi_endproc
  .type main, @function
  .size main, . - main
```

C.16 pread-O3.s

```
# File generated by CompCert 3.7
# Command line: pread.c -S -O3 -o compCert/out/pread-O3.s
  .comm z, 4, 4
  .comm x, 4, 4
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
.L100:
  movl  z(%rip), %edx
  movq  %rdx, %rsi
  movq  %rsi, %rax
  testl %eax, %eax
  leal  1(%eax), %ecx
  cmovl %rcx, %rax
  sarl  $1, %eax
  leal  0(,%eax,2), %edi
  subl  %edi, %esi
  testl %esi, %esi
  jne .L100
  movl  x(%rip), %esi
  movl  z(%rip), %esi
  jmp .L100
  .cfi_endproc
  .type main, @function
  .size main, . - main
```

Appendix D

CompCert Annotated Assembly Output

D.1 comment-O0.s

```
# File generated by CompCert 3.7
# Command line: comment.c -S -O0 -o annotated/comment-O0.s
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
.L100:
  xorl  %eax, %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
  .section  "__compcert_ais_annotations","",@note
```

```
    .ascii "# file:comment.c line:2 function:main\n"
    .byte 7,8
    .quad .L100
    .ascii " Critical Comment\n"
```

D.2 comment-O3.s

```
# File generated by CompCert 3.7
# Command line: comment.c -S -O3 -o annotated/comment-O3.s
    .text
    .align 16
    .globl main
main:
    .cfi_startproc
    subq  $8, %rsp
    .cfi_adjust_cfa_offset  8
    leaq  16(%rsp), %rax
    movq  %rax, 0(%rsp)
.L100:
    xorl  %eax, %eax
    addq  $8, %rsp
    ret
    .cfi_endproc
    .type main, @function
    .size main, . - main
    .section  "__compcert_ais_annotations","",@note
    .ascii "# file:comment.c line:2 function:main\n"
    .byte 7,8
    .quad .L100
    .ascii " Critical Comment\n"
```

D.3 variable-O0.s

```
# File generated by CompCert 3.7
# Command line: variable.c -S -O0
    .text
    .align 16
    .globl main
```

```
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
  movl  $-10, %esi
.L100:
  movl  $98, %ecx
.L101:
  negl  %ecx
.L102:
  movl  $1, %r10d
.L103:
  movabsq $4294967296, %rax
.L104:
  movsd .L105(%rip), %xmm1 # 3.14159265358979312
  cvtsd2ss %xmm1, %xmm3
  cvtss2sd %xmm3, %xmm0
  movsd .L106(%rip), %xmm2 # 2.37840000000000007
  divsd %xmm2, %xmm0
  cqto
  shrq  $59, %rdx
  leaq  0(%rax,%rdx,1), %rax
  sarq  $5, %rax
  leal  0(%eax,%esi,1), %r9d
  leal  0(%r9d,%ecx,1), %r8d
  leal  0(%r8d,%r10d,1), %r10d
  cvttss2si %xmm3, %edx
  leal  0(%r10d,%edx,1), %r11d
  cvttsd2si %xmm0, %r8d
  leal  0(%r11d,%r8d,1), %ecx
  leal  0(%ecx,%edi,1), %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
  .section  .rodata.cst8,"aM",@progbits,8
```

```
    .align  8
.L105:  .quad 0x400921fb54442d18
.L106:  .quad 0x400306f694467382
    .section  "__compcert_ais_annotations","",@note
    .ascii "# file:variable.c line:3 function:main\n"
    .byte 7,8
    .quad .L100
    .ascii " reg(\"rsi\") = int\n"
    .ascii "# file:variable.c line:5 function:main\n"
    .byte 7,8
    .quad .L101
    .ascii " reg(\"rcx\") = char\n"
    .ascii "# file:variable.c line:7 function:main\n"
    .byte 7,8
    .quad .L102
    .ascii " reg(\"rcx\") = unsigned int\n"
    .ascii "# file:variable.c line:9 function:main\n"
    .byte 7,8
    .quad .L103
    .ascii " reg(\"r10\") = short\n"
    .ascii "# file:variable.c line:11 function:main\n"
    .byte 7,8
    .quad .L104
    .ascii " reg(\"rax\") = long\n"
```

D.4 variable-O3.s

```
# File generated by CompCert 3.7
# Command line: variable.c -S -O3
    .text
    .align  16
    .globl main
main:
    .cfi_startproc
    subq  $8, %rsp
    .cfi_adjust_cfa_offset  8
    leaq  16(%rsp), %rax
    movq  %rax, 0(%rsp)
```

```
.L100:
.L101:
.L102:
.L103:
.L104:
  leal  134217625(%edi), %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
  .section  "__compcert_ais_annotations","",@note
  .ascii "# file:variable.c line:3 function:main\n"
  .byte 7,8
  .quad .L100
  .ascii " -10 = int\n"
  .ascii "# file:variable.c line:5 function:main\n"
  .byte 7,8
  .quad .L101
  .ascii " 98 = char\n"
  .ascii "# file:variable.c line:7 function:main\n"
  .byte 7,8
  .quad .L102
  .ascii " -98 = unsigned int\n"
  .ascii "# file:variable.c line:9 function:main\n"
  .byte 7,8
  .quad .L103
  .ascii " 1 = short\n"
  .ascii "# file:variable.c line:11 function:main\n"
  .byte 7,8
  .quad .L104
  .ascii " 4294967296 = long\n"
```

D.5 volatile-O0.s

```
volatile.c:4: error: access to volatile variable 'x' for parameter
    '%e1' is not supported in ais annotations
1 error detected.
```

D.6 volatile-O3.s

```
volatile.c:4: error: access to volatile variable 'x' for parameter
    '%e1' is not supported in ais annotations
1 error detected.
```

D.7 loop-O0.s

```
# File generated by CompCert 3.7
# Command line: loop.c -S -O0 -o annotated/loop-O0.s
  .comm z, 4, 4
  .comm x, 4, 4
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
.L100:
.L101:
  xorl  %edx, %edx
  xorl  %edi, %edi
.L102:
.L103:
.L104:
  nop
.L105:
.L106:
.L107:
  nop
.L108:
.L109:
.L110:
  movl  z(%rip), %edx
  movq  %rdx, %rax
```

```
testl %eax, %eax
leal  1(%eax), %ecx
cmovl %rcx, %rax
sarl  $1, %eax
leal  0(,%eax,2), %edi
movq  %rdx, %rcx
subl  %edi, %ecx
testl %ecx, %ecx
jne .L108
movl  x(%rip), %edi
movl  z(%rip), %esi
jmp .L105
.cfi_endproc
.type main, @function
.size main, . - main
.section  "__compcert_ais_annotations","",@note
.ascii "# file:loop.c line:7 function:main\n"
.byte 7,8
.quad .L100
.ascii " L(mem("
.byte 7,8
.quad z
.ascii ", 4)) = true\n"
.ascii "# file:loop.c line:8 function:main\n"
.byte 7,8
.quad .L101
.ascii " L(mem("
.byte 7,8
.quad x
.ascii ", 4))= mem("
.byte 7,8
.quad z
.ascii ", 4) % 2 == 0\n"
.ascii "# file:loop.c line:11 function:main\n"
.byte 7,8
.quad .L102
.ascii " L(reg(\"rdi\"))= false\n"
.ascii "# file:loop.c line:14 function:main\n"
.byte 7,8
```

```
.quad .L103
.ascii " _P_0: reg(\"rdx\") % 2 == 0\n"
.ascii "# file:loop.c line:15 function:main\n"
.byte 7,8
.quad .L104
.ascii " _Gamma_0: reg(\"rdx\") -> LOW, reg(\"rdi\") -> LOW\n"
.ascii "# file:loop.c line:19 function:main\n"
.byte 7,8
.quad .L106
.ascii " _invariant: reg(\"rdx\") % 2 == 0 & reg(\"rdx\") <= mem("
.byte 7,8
.quad z
.ascii ", 4)\n"
.ascii "# file:loop.c line:20 function:main\n"
.byte 7,8
.quad .L107
.ascii " _Gamma: reg(\"rdx\") -> LOW, reg(\"rdi\") -> (reg(\"rdx\")
  == mem("
.byte 7,8
.quad z
.ascii ", 4)), mem("
.byte 7,8
.quad z
.ascii ", 4) -> LOW\n"
.ascii "# file:loop.c line:22 function:main\n"
.byte 7,8
.quad .L109
.ascii " _invariant: reg(\"rdx\") <= mem("
.byte 7,8
.quad z
.ascii ", 4)\n"
.ascii "# file:loop.c line:23 function:main\n"
.byte 7,8
.quad .L110
.ascii " _Gamma: reg(\"rdx\") -> LOW\n"
```

D.8 loop-O3.s

```
# File generated by CompCert 3.7
# Command line: loop.c -S -O3 -o annotated/loop-O3.s
  .comm z, 4, 4
  .comm x, 4, 4
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
.L100:
.L101:
  xorl  %edx, %edx
  xorl  %edi, %edi
.L102:
.L103:
.L104:
  nop
.L105:
.L106:
.L107:
  nop
.L108:
.L109:
.L110:
  movl  z(%rip), %edx
  movq  %rdx, %rax
  testl %eax, %eax
  leal  1(%eax), %ecx
  cmovl %rcx, %rax
  sarl  $1, %eax
  leal  0(,%eax,2), %edi
  movq  %rdx, %rcx
  subl  %edi, %ecx
  testl %ecx, %ecx
  jne .L108
```

```
movl  x(%rip), %edi
movq  %rdx, %rsi
jmp .L105
.cfi_endproc
.type main, @function
.size main, . - main
.section  "__compcert_ais_annotations","",@note
.ascii "# file:loop.c line:7 function:main\n"
.byte 7,8
.quad .L100
.ascii " L(mem("
.byte 7,8
.quad z
.ascii ", 4)) = true\n"
.ascii "# file:loop.c line:8 function:main\n"
.byte 7,8
.quad .L101
.ascii " L(mem("
.byte 7,8
.quad x
.ascii ", 4))= mem("
.byte 7,8
.quad z
.ascii ", 4) % 2 == 0\n"
.ascii "# file:loop.c line:11 function:main\n"
.byte 7,8
.quad .L102
.ascii " L(0)= false\n"
.ascii "# file:loop.c line:14 function:main\n"
.byte 7,8
.quad .L103
.ascii " _P_0: 0 % 2 == 0\n"
.ascii "# file:loop.c line:15 function:main\n"
.byte 7,8
.quad .L104
.ascii " _Gamma_0: 0 -> LOW, 0 -> LOW\n"
.ascii "# file:loop.c line:19 function:main\n"
.byte 7,8
.quad .L106
```

```
	.ascii " _invariant: reg(\"rdx\") % 2 == 0 & reg(\"rdx\") <= mem("
	.byte 7,8
	.quad z
	.ascii ", 4)\n"
	.ascii "# file:loop.c line:20 function:main\n"
	.byte 7,8
	.quad .L107
	.ascii " _Gamma: reg(\"rdx\") -> LOW, reg(\"rdi\") -> (reg(\"rdx\") == mem("
	.byte 7,8
	.quad z
	.ascii ", 4)), mem("
	.byte 7,8
	.quad z
	.ascii ", 4) -> LOW\n"
	.ascii "# file:loop.c line:22 function:main\n"
	.byte 7,8
	.quad .L109
	.ascii " _invariant: reg(\"rdx\") <= mem("
	.byte 7,8
	.quad z
	.ascii ", 4)\n"
	.ascii "# file:loop.c line:23 function:main\n"
	.byte 7,8
	.quad .L110
	.ascii " _Gamma: reg(\"rdx\") -> LOW\n"
```

D.9 rooster-O0.s

```
# File generated by CompCert 3.7
# Command line: rooster.c -S -O0 -o annotated/rooster-O0.s
	.comm rooster, 4, 4
	.comm drake, 4, 4
	.comm goose, 4, 4
	.data
	.align 4
count:
	.long 0
```

```
  .type count, @object
  .size count, . - count
  .text
  .align  16
  .globl fun
fun:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
.L100:
  leal  0(%edi,%esi,1), %r8d
  leal  0(%r8d,%edx,1), %eax
  testl %eax, %eax
  jl   .L101
  cmpl  %esi, %edi
  jl   .L102
  xorl  %r8d, %r8d
  jmp  .L103
.L102:
  cmpl  %edx, %esi
  setl  %r8b
  movzbl  %r8b, %r8d
.L103:
  cmpl  $0, %r8d
  je   .L104
.L105:
  cmpl  %esi, %edi
  je   .L104
  leal  1(%edi), %edi
  movl  count(%rip), %eax
  leal  1(%eax), %ecx
  movl  %ecx, count(%rip)
.L106:
  cmpl  %edx, %esi
  je   .L105
  leal  -1(%edx), %edx
  movl  count(%rip), %r9d
```

```
  leal  1(%r9d), %r8d
  movl  %r8d, count(%rip)
  jmp .L106
.L104:
  movl  count(%rip), %eax
.L101:
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type fun, @function
  .size fun, . - fun
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
.L107:
.L108:
  movl  $1, %eax
  movl  %eax, rooster(%rip)
  movl  $5, %eax
  movl  %eax, drake(%rip)
  movl  $10, %eax
  movl  %eax, goose(%rip)
  movl  rooster(%rip), %edi
  movl  drake(%rip), %esi
  movl  goose(%rip), %edx
  call  fun
  xorl  %eax, %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
  .section  "__compcert_ais_annotations","",@note
```

```
	.ascii "# file:rooster.c line:6 function:fun\n"
	.byte 7,8
	.quad .L100
	.ascii " CRITICAL COMMENT\n"
	.ascii "# file:rooster.c line:26 function:main\n"
	.byte 7,8
	.quad .L107
	.ascii " L(mem("
	.byte 7,8
	.quad goose
	.ascii ", 4)) = medium\n"
	.ascii "# file:rooster.c line:27 function:main\n"
	.byte 7,8
	.quad .L108
	.ascii " EXCEPTIONAL\n"
```

D.10 rooster-O3.s

```
# File generated by CompCert 3.7
# Command line: rooster.c -S -O3 -o annotated/rooster-O3.s
	.comm rooster, 4, 4
	.comm drake, 4, 4
	.comm goose, 4, 4
	.data
	.align	4
count:
	.long 0
	.type count, @object
	.size count, . - count
	.text
	.align	16
	.globl fun
fun:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
```

```
.L100:
  leal   0(%edi,%esi,1), %r9d
  leal   0(%r9d,%edx,1), %eax
  testl %eax, %eax
  jl   .L101
  cmpl   %edx, %esi
  setl   %al
  movzbl   %al, %eax
  xorl   %r8d, %r8d
  cmpl   %esi, %edi
  cmovge   %r8, %rax
  cmpl   $0, %eax
  je   .L102
.L103:
  cmpl   %esi, %edi
  je   .L102
  leal   1(%edi), %edi
  movl   count(%rip), %ecx
  leal   1(%ecx), %r8d
  movl   %r8d, count(%rip)
.L104:
  cmpl   %edx, %esi
  je   .L103
  leal   -1(%edx), %edx
  movl   count(%rip), %r10d
  leal   1(%r10d), %r8d
  movl   %r8d, count(%rip)
  jmp  .L104
.L102:
  movl   count(%rip), %eax
.L101:
  addq   $8, %rsp
  ret
  .cfi_endproc
  .type fun, @function
  .size fun, . - fun
  .text
  .align  16
  .globl  main
```

```
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
.L105:
.L106:
  movl  $1, %eax
  movl  %eax, rooster(%rip)
  movl  $5, %eax
  movl  %eax, drake(%rip)
  movl  $10, %eax
  movl  %eax, goose(%rip)
  movl  $1, %edi
  movl  $5, %esi
  movl  $10, %edx
  call  fun
  xorl  %eax, %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
  .section  "__compcert_ais_annotations","",@note
  .ascii "# file:rooster.c line:6 function:fun\n"
  .byte 7,8
  .quad .L100
  .ascii " CRITICAL COMMENT\n"
  .ascii "# file:rooster.c line:26 function:main\n"
  .byte 7,8
  .quad .L105
  .ascii " L(mem("
  .byte 7,8
  .quad goose
  .ascii ", 4)) = medium\n"
  .ascii "# file:rooster.c line:27 function:main\n"
  .byte 7,8
  .quad .L106
```

```
	.ascii " EXCEPTIONAL\n"
```

D.11 password-O0.s

```
# File generated by CompCert 3.7
# Command line: password.c -S -O0 -o annotated/password-O0.s
	.section	.rodata
	.align	1
__stringlit_7:
	.ascii	"--- USERS ---\012\000"
	.type	__stringlit_7, @object
	.size	__stringlit_7, . - __stringlit_7
	.section	.rodata
	.align	1
__stringlit_6:
	.ascii	"passw0rd123\000"
	.type	__stringlit_6, @object
	.size	__stringlit_6, . - __stringlit_6
	.section	.rodata
	.align	1
__stringlit_4:
	.ascii	"!alice12!_veuje@@hak\000"
	.type	__stringlit_4, @object
	.size	__stringlit_4, . - __stringlit_4
	.section	.rodata
	.align	1
__stringlit_14:
	.ascii	"Password: \000"
	.type	__stringlit_14, @object
	.size	__stringlit_14, . - __stringlit_14
	.section	.rodata
	.align	1
__stringlit_18:
	.ascii	"Your balance: $%ld\012\000"
	.type	__stringlit_18, @object
	.size	__stringlit_18, . - __stringlit_18
	.section	.rodata
	.align	1
```

```
__stringlit_13:
  .ascii  "User < %s > does not exist.\012\000"
  .type __stringlit_13, @object
  .size __stringlit_13, . - __stringlit_13
  .section  .rodata
  .align  1
__stringlit_8:
  .ascii  " %021d. %s\012\000"
  .type __stringlit_8, @object
  .size __stringlit_8, . - __stringlit_8
  .section  .rodata
  .align  1
__stringlit_1:
  .ascii  "admin\000"
  .type __stringlit_1, @object
  .size __stringlit_1, . - __stringlit_1
  .section  .rodata
  .align  1
__stringlit_2:
  .ascii  "4dm1n__4eva\000"
  .type __stringlit_2, @object
  .size __stringlit_2, . - __stringlit_2
  .section  .rodata
  .align  1
__stringlit_3:
  .ascii  "alice\000"
  .type __stringlit_3, @object
  .size __stringlit_3, . - __stringlit_3
  .section  .rodata
  .align  1
__stringlit_11:
  .ascii  "Username: \000"
  .type __stringlit_11, @object
  .size __stringlit_11, . - __stringlit_11
  .section  .rodata
  .align  1
__stringlit_5:
  .ascii  "abdul\000"
  .type __stringlit_5, @object
```

```
  .size __stringlit_5, . - __stringlit_5
  .section  .rodata
  .align  1
__stringlit_17:
  .ascii  "Welcome, %s!\012\000"
  .type __stringlit_17, @object
  .size __stringlit_17, . - __stringlit_17
  .section  .rodata
  .align  1
__stringlit_12:
  .ascii  "%255s\000"
  .type __stringlit_12, @object
  .size __stringlit_12, . - __stringlit_12
  .section  .rodata
  .align  1
__stringlit_9:
  .ascii  "\012\000"
  .type __stringlit_9, @object
  .size __stringlit_9, . - __stringlit_9
  .section  .rodata
  .align  1
__stringlit_15:
  .ascii  "ERROR: incorrect password\012\000"
  .type __stringlit_15, @object
  .size __stringlit_15, . - __stringlit_15
  .section  .rodata
  .align  1
__stringlit_10:
  .ascii  "Welcome to BigBank Australia!\012\000"
  .type __stringlit_10, @object
  .size __stringlit_10, . - __stringlit_10
  .section  .rodata
  .align  1
__stringlit_16:
  .ascii  "Logged in as < %s >!\012\000"
  .type __stringlit_16, @object
  .size __stringlit_16, . - __stringlit_16
  .text
  .align  16
```

```
  .globl setup_users
setup_users:
  .cfi_startproc
  subq  $40, %rsp
  .cfi_adjust_cfa_offset  40
  leaq  48(%rsp), %rax
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  movq  %rbp, 16(%rsp)
  movq  %r12, 24(%rsp)
  movq  $528, %rdi
  call  malloc
  movq  %rax, %rbp
  leaq  8(%rbp), %rdi
  leaq  __stringlit_1(%rip), %rsi
  call  strcpy
  leaq  264(%rbp), %rdi
  leaq  __stringlit_2(%rip), %rsi
  call  strcpy
.L100:
  movq  $1000000, %r10
  movq  %r10, 520(%rbp)
  movq  $528, %rdi
  call  malloc
  movq  %rax, %r12
  leaq  8(%r12), %rdi
  leaq  __stringlit_3(%rip), %rsi
  call  strcpy
  leaq  264(%r12), %rdi
  leaq  __stringlit_4(%rip), %rsi
  call  strcpy
.L101:
  movq  $783, %r9
  movq  %r9, 520(%r12)
  movq  $528, %rdi
  call  malloc
  movq  %rax, %rbx
  leaq  8(%rbx), %rdi
  leaq  __stringlit_5(%rip), %rsi
```

```
  call  strcpy
  leaq  264(%rbx), %rdi
  leaq  __stringlit_6(%rip), %rsi
  call  strcpy
.L102:
  movq  $2, %r11
  movq  %r11, 520(%rbx)
  movq  %r12, 0(%rbp)
  movq  %rbx, 0(%r12)
  xorq  %r8, %r8
  movq  %r8, 0(%rbx)
  movq  %rbp, %rax
  movq  8(%rsp), %rbx
  movq  16(%rsp), %rbp
  movq  24(%rsp), %r12
  addq  $40, %rsp
  ret
  .cfi_endproc
  .type setup_users, @function
  .size setup_users, . - setup_users
  .text
  .align  16
  .globl print_users
print_users:
  .cfi_startproc
  subq  $24, %rsp
  .cfi_adjust_cfa_offset  24
  leaq  32(%rsp), %rax
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  movq  %rbp, 16(%rsp)
  movq  %rdi, %rbx
  leaq  __stringlit_7(%rip), %rdi
  movl  $0, %eax
  call  printf
  xorq  %rbp, %rbp
.L103:
  cmpq  $0, %rbx
  je  .L104
```

```
  leaq   1(%rbp), %rbp
  leaq   __stringlit_8(%rip), %rdi
  leaq   8(%rbx), %rdx
  movq   %rbp, %rsi
  movl   $0, %eax
  call   printf
  movq   0(%rbx), %rbx
  jmp  .L103
.L104:
  leaq   __stringlit_9(%rip), %rdi
  movl   $0, %eax
  call   printf
  movq   8(%rsp), %rbx
  movq   16(%rsp), %rbp
  addq   $24, %rsp
  ret
  .cfi_endproc
  .type print_users, @function
  .size print_users, . - print_users
  .text
  .align  16
  .globl getUser
getUser:
  .cfi_startproc
  subq   $24, %rsp
  .cfi_adjust_cfa_offset  24
  leaq   32(%rsp), %rax
  movq   %rax, 0(%rsp)
  movq   %rbx, 8(%rsp)
  movq   %rbp, 16(%rsp)
  movq   %rsi, %rbp
  movq   %rdi, %rbx
.L105:
  cmpq   $0, %rbx
  je   .L106
  leaq   8(%rbx), %rdi
  movq   %rbp, %rsi
  call   strcmp
  testl %eax, %eax
```

```
  je   .L107
  movq  0(%rbx), %rbx
  jmp .L105
.L106:
  xorq  %rbx, %rbx
.L107:
  movq  %rbx, %rax
  movq  8(%rsp), %rbx
  movq  16(%rsp), %rbp
  addq  $24, %rsp
  ret
  .cfi_endproc
  .type getUser, @function
  .size getUser, . - getUser
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $536, %rsp
  .cfi_adjust_cfa_offset  536
  leaq  544(%rsp), %rax
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  call  setup_users
  movq  %rax, %rbx
  leaq  __stringlit_10(%rip), %rdi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_11(%rip), %rdi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_12(%rip), %rdi
  leaq  16(%rsp), %rsi
  movl  $0, %eax
  call  __isoc99_scanf
  leaq  16(%rsp), %rsi
  movq  %rbx, %rdi
  call  getUser
```

```
  movq  %rax, %rbx
  cmpq  $0, %rbx
  jne .L108
  leaq  __stringlit_13(%rip), %rdi
  leaq  16(%rsp), %rsi
  movl  $0, %eax
  call  printf
  xorl  %eax, %eax
  jmp .L109
.L108:
  leaq  __stringlit_14(%rip), %rdi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_12(%rip), %rdi
  leaq  272(%rsp), %rsi
  movl  $0, %eax
  call  __isoc99_scanf
  leaq  264(%rbx), %rdi
  leaq  272(%rsp), %rsi
  call  strcmp
  testl %eax, %eax
  je  .L110
  leaq  __stringlit_15(%rip), %rdi
  movl  $0, %eax
  call  printf
  xorl  %eax, %eax
  jmp .L109
.L110:
  leaq  __stringlit_16(%rip), %rdi
  leaq  8(%rbx), %rsi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_9(%rip), %rdi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_17(%rip), %rdi
  leaq  8(%rbx), %rsi
  movl  $0, %eax
  call  printf
```

```
  leaq  __stringlit_18(%rip), %rdi
  movq  520(%rbx), %rsi
  movl  $0, %eax
  call  printf
  xorl  %eax, %eax
.L109:
  movq  8(%rsp), %rbx
  addq  $536, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
  .section  "__compcert_ais_annotations","",@note
  .ascii "# file:password.c line:20 function:setup_users\n"
  .byte 7,8
  .quad .L100
  .ascii " L((reg(\"rbp\") + 264)) = high\n"
  .ascii "# file:password.c line:26 function:setup_users\n"
  .byte 7,8
  .quad .L101
  .ascii " L((reg(\"r12\") + 264)) = high\n"
  .ascii "# file:password.c line:32 function:setup_users\n"
  .byte 7,8
  .quad .L102
  .ascii " L((reg(\"rbx\") + 264)) = high\n"
```

D.12 password-O3.s

```
# File generated by CompCert 3.7
# Command line: password.c -S -O3 -o annotated/password-O3.s
  .section  .rodata
  .align  1
__stringlit_7:
  .ascii  "--- USERS ---\012\000"
  .type __stringlit_7, @object
  .size __stringlit_7, . - __stringlit_7
  .section  .rodata
  .align  1
```

```
__stringlit_6:
  .ascii  "passw0rd123\000"
  .type __stringlit_6, @object
  .size __stringlit_6, . - __stringlit_6
  .section  .rodata
  .align  1
__stringlit_4:
  .ascii  "!alice12!_veuje@@hak\000"
  .type __stringlit_4, @object
  .size __stringlit_4, . - __stringlit_4
  .section  .rodata
  .align  1
__stringlit_14:
  .ascii  "Password: \000"
  .type __stringlit_14, @object
  .size __stringlit_14, . - __stringlit_14
  .section  .rodata
  .align  1
__stringlit_18:
  .ascii  "Your balance: $%ld\012\000"
  .type __stringlit_18, @object
  .size __stringlit_18, . - __stringlit_18
  .section  .rodata
  .align  1
__stringlit_13:
  .ascii  "User < %s > does not exist.\012\000"
  .type __stringlit_13, @object
  .size __stringlit_13, . - __stringlit_13
  .section  .rodata
  .align  1
__stringlit_8:
  .ascii  " %021d. %s\012\000"
  .type __stringlit_8, @object
  .size __stringlit_8, . - __stringlit_8
  .section  .rodata
  .align  1
__stringlit_1:
  .ascii  "admin\000"
  .type __stringlit_1, @object
```

```
  .size __stringlit_1, . - __stringlit_1
  .section  .rodata
  .align  1
__stringlit_2:
  .ascii  "4dm1n__4eva\000"
  .type __stringlit_2, @object
  .size __stringlit_2, . - __stringlit_2
  .section  .rodata
  .align  1
__stringlit_3:
  .ascii  "alice\000"
  .type __stringlit_3, @object
  .size __stringlit_3, . - __stringlit_3
  .section  .rodata
  .align  1
__stringlit_11:
  .ascii  "Username: \000"
  .type __stringlit_11, @object
  .size __stringlit_11, . - __stringlit_11
  .section  .rodata
  .align  1
__stringlit_5:
  .ascii  "abdul\000"
  .type __stringlit_5, @object
  .size __stringlit_5, . - __stringlit_5
  .section  .rodata
  .align  1
__stringlit_17:
  .ascii  "Welcome, %s!\012\000"
  .type __stringlit_17, @object
  .size __stringlit_17, . - __stringlit_17
  .section  .rodata
  .align  1
__stringlit_12:
  .ascii  "%255s\000"
  .type __stringlit_12, @object
  .size __stringlit_12, . - __stringlit_12
  .section  .rodata
  .align  1
```

```
__stringlit_9:
  .ascii  "\012\000"
  .type __stringlit_9, @object
  .size __stringlit_9, . - __stringlit_9
  .section  .rodata
  .align  1
__stringlit_15:
  .ascii  "ERROR: incorrect password\012\000"
  .type __stringlit_15, @object
  .size __stringlit_15, . - __stringlit_15
  .section  .rodata
  .align  1
__stringlit_10:
  .ascii  "Welcome to BigBank Australia!\012\000"
  .type __stringlit_10, @object
  .size __stringlit_10, . - __stringlit_10
  .section  .rodata
  .align  1
__stringlit_16:
  .ascii  "Logged in as < %s >!\012\000"
  .type __stringlit_16, @object
  .size __stringlit_16, . - __stringlit_16
  .text
  .align  16
  .globl setup_users
setup_users:
  .cfi_startproc
  subq  $40, %rsp
  .cfi_adjust_cfa_offset  40
  leaq  48(%rsp), %rax
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  movq  %rbp, 16(%rsp)
  movq  %r12, 24(%rsp)
  movq  $528, %rdi
  call  malloc
  movq  %rax, %rbp
  leaq  8(%rbp), %rdi
  leaq  __stringlit_1(%rip), %rsi
```

```
  call  strcpy
  leaq  264(%rbp), %rdi
  leaq  __stringlit_2(%rip), %rsi
  call  strcpy
.L100:
  movq  $1000000, %r10
  movq  %r10, 520(%rbp)
  movq  $528, %rdi
  call  malloc
  movq  %rax, %r12
  leaq  8(%r12), %rdi
  leaq  __stringlit_3(%rip), %rsi
  call  strcpy
  leaq  264(%r12), %rdi
  leaq  __stringlit_4(%rip), %rsi
  call  strcpy
.L101:
  movq  $783, %r9
  movq  %r9, 520(%r12)
  movq  $528, %rdi
  call  malloc
  movq  %rax, %rbx
  leaq  8(%rbx), %rdi
  leaq  __stringlit_5(%rip), %rsi
  call  strcpy
  leaq  264(%rbx), %rdi
  leaq  __stringlit_6(%rip), %rsi
  call  strcpy
.L102:
  movq  $2, %r11
  movq  %r11, 520(%rbx)
  movq  %r12, 0(%rbp)
  movq  %rbx, 0(%r12)
  xorq  %r8, %r8
  movq  %r8, 0(%rbx)
  movq  %rbp, %rax
  movq  8(%rsp), %rbx
  movq  16(%rsp), %rbp
  movq  24(%rsp), %r12
```

```
  addq  $40, %rsp
  ret
  .cfi_endproc
  .type setup_users, @function
  .size setup_users, . - setup_users
  .text
  .align  16
  .globl print_users
print_users:
  .cfi_startproc
  subq  $24, %rsp
  .cfi_adjust_cfa_offset  24
  leaq  32(%rsp), %rax
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  movq  %rbp, 16(%rsp)
  movq  %rdi, %rbp
  leaq  __stringlit_7(%rip), %rdi
  movl  $0, %eax
  call  printf
  xorq  %rbx, %rbx
.L103:
  cmpq  $0, %rbp
  je  .L104
  leaq  1(%rbx), %rbx
  leaq  __stringlit_8(%rip), %rdi
  leaq  8(%rbp), %rdx
  movq  %rbx, %rsi
  movl  $0, %eax
  call  printf
  movq  0(%rbp), %rbp
  jmp .L103
.L104:
  leaq  __stringlit_9(%rip), %rdi
  movl  $0, %eax
  call  printf
  movq  8(%rsp), %rbx
  movq  16(%rsp), %rbp
  addq  $24, %rsp
```

```
  ret
  .cfi_endproc
  .type print_users, @function
  .size print_users, . - print_users
  .text
  .align  16
  .globl getUser
getUser:
  .cfi_startproc
  subq  $24, %rsp
  .cfi_adjust_cfa_offset  24
  leaq  32(%rsp), %rax
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  movq  %rbp, 16(%rsp)
  movq  %rsi, %rbp
  movq  %rdi, %rbx
.L105:
  cmpq  $0, %rbx
  je   .L106
  leaq  8(%rbx), %rdi
  movq  %rbp, %rsi
  call  strcmp
  testl %eax, %eax
  je   .L107
  movq  0(%rbx), %rbx
  jmp .L105
.L106:
  xorq  %rbx, %rbx
.L107:
  movq  %rbx, %rax
  movq  8(%rsp), %rbx
  movq  16(%rsp), %rbp
  addq  $24, %rsp
  ret
  .cfi_endproc
  .type getUser, @function
  .size getUser, . - getUser
  .text
```

```
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $536, %rsp
  .cfi_adjust_cfa_offset  536
  leaq  544(%rsp), %rax
  movq  %rax, 0(%rsp)
  movq  %rbx, 8(%rsp)
  call  setup_users
  movq  %rax, %rbx
  leaq  __stringlit_10(%rip), %rdi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_11(%rip), %rdi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_12(%rip), %rdi
  leaq  16(%rsp), %rsi
  movl  $0, %eax
  call  __isoc99_scanf
  leaq  16(%rsp), %rsi
  movq  %rbx, %rdi
  call  getUser
  movq  %rax, %rbx
  cmpq  $0, %rbx
  jne .L108
  leaq  __stringlit_13(%rip), %rdi
  leaq  16(%rsp), %rsi
  movl  $0, %eax
  call  printf
  xorl  %eax, %eax
  jmp .L109
.L108:
  leaq  __stringlit_14(%rip), %rdi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_12(%rip), %rdi
  leaq  272(%rsp), %rsi
```

```
  movl  $0, %eax
  call  __isoc99_scanf
  leaq  264(%rbx), %rdi
  leaq  272(%rsp), %rsi
  call  strcmp
  testl %eax, %eax
  je    .L110
  leaq  __stringlit_15(%rip), %rdi
  movl  $0, %eax
  call  printf
  xorl  %eax, %eax
  jmp   .L109
.L110:
  leaq  __stringlit_16(%rip), %rdi
  leaq  8(%rbx), %rsi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_9(%rip), %rdi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_17(%rip), %rdi
  leaq  8(%rbx), %rsi
  movl  $0, %eax
  call  printf
  leaq  __stringlit_18(%rip), %rdi
  movq  520(%rbx), %rsi
  movl  $0, %eax
  call  printf
  xorl  %eax, %eax
.L109:
  movq  8(%rsp), %rbx
  addq  $536, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
  .section  "__compcert_ais_annotations","",@note
  .ascii "# file:password.c line:20 function:setup_users\n"
  .byte 7,8
```

```
	.quad .L100
	.ascii " L((reg(\"rbp\") + 264)) = high\n"
	.ascii "# file:password.c line:26 function:setup_users\n"
	.byte 7,8
	.quad .L101
	.ascii " L((reg(\"r12\") + 264)) = high\n"
	.ascii "# file:password.c line:32 function:setup_users\n"
	.byte 7,8
	.quad .L102
	.ascii " L((reg(\"rbx\") + 264)) = high\n"
```

D.13 deadStoreElimination-O0.s

```
# File generated by CompCert 3.7
# Command line: deadStoreElimination.c -S -O0 -o annotated/deadStoreElimination-O0.s
	.text
	.align	16
	.globl deadStore
deadStore:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
	movl	$43981, %ecx
.L100:
	nop
.L101:
	cmpl	%esi, %edi
	jle	.L102
	leal	-1(%edi), %edi
	jmp	.L101
.L102:
	leal	0(%edi,%esi,1), %eax
	addq	$8, %rsp
	ret
	.cfi_endproc
```

```
	.type deadStore, @function
	.size deadStore, . - deadStore
	.text
	.align	16
	.globl main
main:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
	movq	%rax, 0(%rsp)
	movl	$2, %esi
	call	deadStore
	xorl	%eax, %eax
	addq	$8, %rsp
	ret
	.cfi_endproc
	.type main, @function
	.size main, . - main
	.section	"__compcert_ais_annotations","",@note
	.ascii "# file:deadStoreElimination.c line:3 function:deadStore\n"
	.byte 7,8
	.quad .L100
	.ascii " L(reg(\"rcx\")) = high\n"
```

D.14 deadStoreElimination-O3.s

```
# File generated by CompCert 3.7
# Command line: deadStoreElimination.c -S -O3 -o
    annotated/deadStoreElimination-O3.s
	.text
	.align	16
	.globl deadStore
deadStore:
	.cfi_startproc
	subq	$8, %rsp
	.cfi_adjust_cfa_offset	8
	leaq	16(%rsp), %rax
```

```
  movq  %rax, 0(%rsp)
.L100:
  nop
.L101:
  cmpl  %esi, %edi
  jle .L102
  leal  -1(%edi), %edi
  jmp .L101
.L102:
  leal  0(%edi,%esi,1), %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type deadStore, @function
  .size deadStore, . - deadStore
  .text
  .align  16
  .globl main
main:
  .cfi_startproc
  subq  $8, %rsp
  .cfi_adjust_cfa_offset  8
  leaq  16(%rsp), %rax
  movq  %rax, 0(%rsp)
  movl  $2, %esi
  call  deadStore
  xorl  %eax, %eax
  addq  $8, %rsp
  ret
  .cfi_endproc
  .type main, @function
  .size main, . - main
  .section  "__compcert_ais_annotations","",@note
  .ascii "# file:deadStoreElimination.c line:3 function:deadStore\n"
  .byte 7,8
  .quad .L100
  .ascii " L(43981) = high\n"
```

D.15 pread-O0.s

```
pread.c:6: error: access to volatile variable 'z' for parameter '%e1'
    is not supported in ais annotations
pread.c:7: error: access to volatile variable 'x' for parameter '%e1'
    is not supported in ais annotations
pread.c:7: error: access to volatile variable 'z' for parameter '%e2'
    is not supported in ais annotations
pread.c:20: error: access to volatile variable 'z' for parameter '%e2'
    is not supported in ais annotations
pread.c:21: error: access to volatile variable 'z' for parameter '%e3'
    is not supported in ais annotations
pread.c:21: error: access to volatile variable 'z' for parameter '%e3'
    is not supported in ais annotations
pread.c:23: error: access to volatile variable 'z' for parameter '%e2'
    is not supported in ais annotations
7 errors detected.
```

D.16 pread-O3.s

```
pread.c:6: error: access to volatile variable 'z' for parameter '%e1'
    is not supported in ais annotations
pread.c:7: error: access to volatile variable 'x' for parameter '%e1'
    is not supported in ais annotations
pread.c:7: error: access to volatile variable 'z' for parameter '%e2'
    is not supported in ais annotations
pread.c:20: error: access to volatile variable 'z' for parameter '%e2'
    is not supported in ais annotations
pread.c:21: error: access to volatile variable 'z' for parameter '%e3'
    is not supported in ais annotations
pread.c:21: error: access to volatile variable 'z' for parameter '%e3'
    is not supported in ais annotations
pread.c:23: error: access to volatile variable 'z' for parameter '%e2'
    is not supported in ais annotations
7 errors detected.
```

Appendix E

Inline Assembly Annotated C Programs

E.1 comment.c

```c
int main() {
    asm("# CRITICAL COMMENT");
    return 0;
}
```

E.2 variable.c

```c
int main(int argc, char* argv[]) {
    // a = int
    // b = char
    // c = unsigned int
    // d = short
    // e = long
    // x = float
    // y = double
    int a = -10;
    asm("# annotation: %0 = int" : : "X"(a));
    char b = 'b';
    asm("# annotation: %0 = char" : : "X"(b));
```

```
    unsigned int c = -b;
    asm("# annotation: %0 = unsigned int" : : "X"(c));
    short d = 0x1;
    asm("# annotation: %0 = short" : : "X"(d));
    long e = 4294967296;
    asm("# annotation: %0 = long" : : "X"(e));
    float x = 3.141592653589793;
    asm("# annotation: %0 = float" : : "X"(x));
    double y = x / 2.3784;
    asm("# annotation: %0 = double" : : "X"(y));
    return (int)(e / 32) + (int)a + (int)c + (int)d + (int) x + (int) y + argc;
}
```

E.3 volatile.c

```
volatile int x;

int main() {
    asm("# annotation: %0 = High" : : "X"(x));
    return x + 1;
}
```

E.4 loop.c

```
int z;
int x;

// security policies
// {L(z)=true}
// {L(x)=z % 2 == 0}

// predicates on initial state
// {_P_0: r1 % 2 == 0}
// {_Gamma_0: r1 -> LOW, r2 -> LOW}

int main() {
    int r1 = 0;
```

```
    // {L(r2)=False}
    int r2 = 0;

    while(1) {
    do {
        // {_invariant: r1 % 2 == 0 /\ r1 <= z}
        // {_Gamma: r1 -> LOW, r2 -> (r1 == z), z -> LOW}
        do {
            // {_invariant: r1 <= z}
            // {_Gamma: r1 -> LOW}
            r1 = z;
        } while (r1 %2 != 0);
            r2 = x;
        } while (z != r1);
    }
    return r2;
}
```

E.5 rooster.c

```
int rooster;
int drake;
// MEDIUM
int goose;

int fun(int a, int b, int c) {
    // CRITICAL COMMENT
    static int count = 0;
    int sum = a + b + c;
    if (sum < 0) {
        return sum;
    }
    if (a < b && b < c) {
        while (a != b) {
            a++;
            count++;
            while (b != c) {
                c--;
```

```
                count++;
            }
        }
    }
    return count;
}

int main(void) {
    // EXCEPTIONAL
    rooster = 1;
    drake = 5;
    goose = 10;
    int result;
    result = fun(rooster,drake,goose);
    return 0;
}
```

E.6 password.c

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

#define BUFF_LEN 256

typedef struct user_t user_t;

struct user_t {
    user_t* next;
    char name[BUFF_LEN];
    // L(password) = High
    char password[BUFF_LEN];
    size_t balance;
};

user_t* setup_users() {
    user_t* user_admin = malloc(sizeof(user_t));
    strcpy(user_admin->name, "admin");
```

```
    strcpy(user_admin->password, "4dm1n__4eva");
    user_admin->balance = 1000000;

    user_t* user_alice = malloc(sizeof(user_t));
    strcpy(user_alice->name, "alice");
    strcpy(user_alice->password, "!alice12!_veuje@@hak");
    user_alice->balance = 783;

    user_t* user_abdul = malloc(sizeof(user_t));
    strcpy(user_abdul->name, "abdul");
    strcpy(user_abdul->password, "passw0rd123");
    user_abdul->balance = 2;

    user_admin->next = user_alice;
    user_alice->next = user_abdul;
    user_abdul->next = NULL;

    return user_admin;
}

void print_users(user_t* users) {
    printf("--- USERS ---\n");
    size_t count = 0;
    while (users != NULL) {
        printf(" %02ld. %s\n", ++count, users->name);
        users = users->next;
    }
    printf("\n");
}

user_t* getUser(user_t* user_list, char* name) {
    while (user_list != NULL) {
        if (strcmp(user_list->name, name) == 0) {
            return user_list;
        }
        user_list = user_list->next;
    }
    return NULL;
}
```

```

int main() {
    user_t* users = setup_users();

    printf("Welcome to BigBank Australia!\n");

    char username[BUFF_LEN];
    printf("Username: ");
    scanf("%255s", username);

    user_t* user = getUser(users, username);
    if (user == NULL) {
        printf("User < %s > does not exist.\n", username);
        return 0;
    }

    char password[BUFF_LEN];
    printf("Password: ");
    scanf("%255s", password);
    if (strcmp(user->password, password) != 0) {
        printf("ERROR: incorrect password\n");
        return 0;
    }

    printf("Logged in as < %s >!\n", user->name);
    printf("\n");
    printf("Welcome, %s!\n", user->name);
    printf("Your balance: $%ld\n", user->balance);
}
```

E.7 deadStoreElimination.c

```
int deadStore(int i, int n) {
    int key = 0xabcd;
    // L(key) = high

    // do some work
    int result = 0;
```

```
    while (i > n) {
        result += key;
        i--;
    }

    // clear out our secret key
    key = 0;
    return i + n;
}

int main(int argc, char *argv[]) {
    deadStore(argc, 2);
}
```

E.8 pread.c

```
volatile int z;
volatile int x;

// security policies
// {L(z)=true}
// {L(x)=z % 2 == 0}

// predicates on initial state
// {_P_0: r1 % 2 == 0}
// {_Gamma_0: r1 -> LOW, r2 -> LOW}

int main() {
    int r1 = 0;
    // {L(r2)=False}
    int r2 = 0;

    while(1) {
    do {
        // {_invariant: r1 % 2 == 0 /\ r1 <= z}
        // {_Gamma: r1 -> LOW, r2 -> (r1 == z), z -> LOW}
        do {
            // {_invariant: r1 <= z}
```

```
            // {_Gamma: r1 -> LOW}
            r1 = z;
        } while (r1 %2 != 0);
            r2 = x;
        } while (z != r1);
    }
    return r2;
}
```

Appendix F

Inline Assembly Annotated Assembly Output

F.1 comment-O0.s

```
	.text
	.file	"comment.c"
	.globl	main                    # -- Begin function main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset %rbp, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register %rbp
	movl	$0, -4(%rbp)
	#APP
	# CRITICAL COMMENT
	#NO_APP
	xorl	%eax, %eax
	popq	%rbp
```

```
	.cfi_def_cfa %rsp, 8
	retq
.Lfunc_end0:
	.size	main, .Lfunc_end0-main
	.cfi_endproc
                                        # -- End function
	.ident	"clang version 10.0.0-4ubuntu1 "
	.section	".note.GNU-stack","",@progbits
	.addrsig
```

F.2 comment-O3.s

```
	.text
	.file	"comment.c"
	.globl	main                    # -- Begin function main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
	#APP
	# CRITICAL COMMENT
	#NO_APP
	xorl	%eax, %eax
	retq
.Lfunc_end0:
	.size	main, .Lfunc_end0-main
	.cfi_endproc
                                        # -- End function
	.ident	"clang version 10.0.0-4ubuntu1 "
	.section	".note.GNU-stack","",@progbits
	.addrsig
```

F.3 variable-O0.s

```
	.text
	.file	"variable.c"
	.section	.rodata.cst8,"aM",@progbits,8
```

```
  .p2align  3                 # -- Begin function main
.LCPI0_0:
  .quad 4612538099476886402      # double 2.3784000000000001
  .section  .rodata.cst4,"aM",@progbits,4
  .p2align  2
.LCPI0_1:
  .long 1078530011               # float 3.14159274
  .text
  .globl  main
  .p2align  4, 0x90
  .type main,@function
main:                                       # @main
  .cfi_startproc
# %bb.0:
  pushq %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset %rbp, -16
  movq  %rsp, %rbp
  .cfi_def_cfa_register %rbp
  movl  $0, -4(%rbp)
  movl  %edi, -8(%rbp)
  movq  %rsi, -16(%rbp)
  movl  $-10, -20(%rbp)
  movl  -20(%rbp), %eax
  #APP
  # annotation: %eax = int
  #NO_APP
  movb  $98, -21(%rbp)
  movb  -21(%rbp), %cl
  #APP
  # annotation: %cl = char
  #NO_APP
  xorl  %eax, %eax
  movsbl  -21(%rbp), %edx
  subl  %edx, %eax
  movl  %eax, -28(%rbp)
  movl  -28(%rbp), %eax
  #APP
  # annotation: %eax = unsigned int
```

```
#NO_APP
movw   $1, -30(%rbp)
movw   -30(%rbp), %r8w
#APP
# annotation: %r8w = short
#NO_APP
movabsq $4294967296, %rsi        # imm = 0x100000000
movq   %rsi, -40(%rbp)
movq   -40(%rbp), %rsi
#APP
# annotation: %rsi = long
#NO_APP
movss .LCPI0_1(%rip), %xmm0    # xmm0 = mem[0],zero,zero,zero
movss %xmm0, -44(%rbp)
movss -44(%rbp), %xmm0         # xmm0 = mem[0],zero,zero,zero
#APP
# annotation: %xmm0 = float
#NO_APP
movsd .LCPI0_0(%rip), %xmm0    # xmm0 = mem[0],zero
movss -44(%rbp), %xmm1         # xmm1 = mem[0],zero,zero,zero
cvtss2sd  %xmm1, %xmm1
divsd %xmm0, %xmm1
movsd %xmm1, -56(%rbp)
movsd -56(%rbp), %xmm0         # xmm0 = mem[0],zero
#APP
# annotation: %xmm0 = double
#NO_APP
movq   -40(%rbp), %rax
cqto
movl   $32, %esi
idivq %rsi
                                       # kill: def $eax killed $eax
 killed $rax
addl   -20(%rbp), %eax
addl   -28(%rbp), %eax
movswl   -30(%rbp), %edi
addl   %edi, %eax
cvttss2si -44(%rbp), %edi
addl   %edi, %eax
```

```
	cvttsd2si	-56(%rbp), %edi
	addl	%edi, %eax
	addl	-8(%rbp), %eax
	popq	%rbp
	.cfi_def_cfa %rsp, 8
	retq
.Lfunc_end0:
	.size	main, .Lfunc_end0-main
	.cfi_endproc
                                        # -- End function
	.ident	"clang version 10.0.0-4ubuntu1 "
	.section	".note.GNU-stack","",@progbits
	.addrsig
```

F.4 variable-O3.s

```
	.text
	.file	"variable.c"
	.section	.rodata.cst4,"aM",@progbits,4
	.p2align	2               # -- Begin function main
.LCPI0_0:
	.long	1078530011              # float 3.14159274
	.section	.rodata.cst8,"aM",@progbits,8
	.p2align	3
.LCPI0_1:
	.quad	4608627556095693531     # double 1.3208849398808329
	.text
	.globl	main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
                                        # kill: def $edi killed $edi def $rdi
	#APP
	# annotation: $-10 = int
	#NO_APP
```

```
	#APP
	# annotation: $98 = char
	#NO_APP
	#APP
	# annotation: $-98 = unsigned int
	#NO_APP
	#APP
	# annotation: $1 = short
	#NO_APP
	#APP
	# annotation: $4294967296 = long
	#NO_APP
	movss .LCPI0_0(%rip), %xmm0    # xmm0 = mem[0],zero,zero,zero
	#APP
	# annotation: %xmm0 = float
	#NO_APP
	movsd .LCPI0_1(%rip), %xmm0    # xmm0 = mem[0],zero
	#APP
	# annotation: %xmm0 = double
	#NO_APP
	leal  134217625(%rdi), %eax
	retq
.Lfunc_end0:
	.size main, .Lfunc_end0-main
	.cfi_endproc
                                        # -- End function
	.ident  "clang version 10.0.0-4ubuntu1 "
	.section  ".note.GNU-stack","",@progbits
	.addrsig
```

F.5 volatile-O0.s

```
	.text
	.file "volatile.c"
	.globl  main                    # -- Begin function main
	.p2align  4, 0x90
	.type main,@function
main:                                   # @main
```

```
	.cfi_startproc
# %bb.0:
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset %rbp, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register %rbp
	movl	$0, -4(%rbp)
	movl	x, %eax
	#APP
	# annotation: %eax = High
	#NO_APP
	movl	x, %eax
	addl	$1, %eax
	popq	%rbp
	.cfi_def_cfa %rsp, 8
	retq
.Lfunc_end0:
	.size	main, .Lfunc_end0-main
	.cfi_endproc
                                        # -- End function
	.type	x,@object               # @x
	.comm	x,4,4
	.ident	"clang version 10.0.0-4ubuntu1 "
	.section	".note.GNU-stack","",@progbits
	.addrsig
	.addrsig_sym x
```

F.6 volatile-O3.s

```
	.text
	.file	"volatile.c"
	.globl	main                    # -- Begin function main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
```

```
	movl	x(%rip), %eax
	#APP
	# annotation: %eax = High
	#NO_APP
	movl	x(%rip), %eax
	addl	$1, %eax
	retq
.Lfunc_end0:
	.size	main, .Lfunc_end0-main
	.cfi_endproc
                                        # -- End function
	.type	x,@object               # @x
	.comm	x,4,4
	.ident	"clang version 10.0.0-4ubuntu1 "
	.section	".note.GNU-stack","",@progbits
	.addrsig
	.addrsig_sym x
```

F.7 loop-O0.s

```
	.text
	.file	"loop.c"
	.globl	main                    # -- Begin function main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset %rbp, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register %rbp
	movl	$0, -4(%rbp)
	movl	$0, -8(%rbp)
	movl	$0, -12(%rbp)
.LBB0_1:                                # =>This Loop Header: Depth=1
                                        #     Child Loop BB0_2 Depth 2
```

```
                                                    #       Child Loop BB0_3 Depth 3
  jmp .LBB0_2
.LBB0_2:                                            #   Parent Loop BB0_1 Depth=1
                                                    # =>  This Loop Header: Depth=2
                                                    #       Child Loop BB0_3 Depth 3
  jmp .LBB0_3
.LBB0_3:                                            #   Parent Loop BB0_1 Depth=1
                                                    #     Parent Loop BB0_2 Depth=2
                                                    # =>    This Inner Loop Header: Depth=3
  movl  z, %eax
  movl  %eax, -8(%rbp)
# %bb.4:                                            #   in Loop: Header=BB0_3 Depth=3
  movl  -8(%rbp), %eax
  cltd
  movl  $2, %ecx
  idivl %ecx
  cmpl  $0, %edx
  jne .LBB0_3
# %bb.5:                                            #   in Loop: Header=BB0_2 Depth=2
  movl  x, %eax
  movl  %eax, -12(%rbp)
# %bb.6:                                            #   in Loop: Header=BB0_2 Depth=2
  movl  z, %eax
  cmpl  -8(%rbp), %eax
  jne .LBB0_2
# %bb.7:                                            #   in Loop: Header=BB0_1 Depth=1
  jmp .LBB0_1
.Lfunc_end0:
  .size main, .Lfunc_end0-main
  .cfi_endproc
                                                    # -- End function
  .type z,@object                           # @z
```

```
	.comm	z,4,4
	.type	x,@object               # @x
	.comm	x,4,4
	.ident	"clang version 10.0.0-4ubuntu1 "
	.section	".note.GNU-stack","",@progbits
	.addrsig
	.addrsig_sym z
	.addrsig_sym x
```

F.8 loop-O3.s

```
	.text
	.file	"loop.c"
	.globl	main                    # -- Begin function main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
	testb	$1, z(%rip)
	jne	.LBB0_2
	.p2align	4, 0x90
.LBB0_1:                                # =>This Inner Loop Header: Depth=1
	jmp	.LBB0_1
	.p2align	4, 0x90
.LBB0_2:                                # =>This Inner Loop Header: Depth=1
	jmp	.LBB0_2
.Lfunc_end0:
	.size	main, .Lfunc_end0-main
	.cfi_endproc
                                        # -- End function
	.type	z,@object               # @z
	.comm	z,4,4
	.type	x,@object               # @x
	.comm	x,4,4
	.ident	"clang version 10.0.0-4ubuntu1 "
```

```
	.section	".note.GNU-stack","",@progbits
	.addrsig
```

F.9 rooster-O0.s

```
	.text
	.file "rooster.c"
	.globl	fun                    # -- Begin function fun
	.p2align	4, 0x90
	.type	fun,@function
fun:                                    # @fun
	.cfi_startproc
# %bb.0:
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset %rbp, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register %rbp
	movl	%edi, -8(%rbp)
	movl	%esi, -12(%rbp)
	movl	%edx, -16(%rbp)
	movl	-8(%rbp), %eax
	addl	-12(%rbp), %eax
	addl	-16(%rbp), %eax
	movl	%eax, -20(%rbp)
	cmpl	$0, -20(%rbp)
	jge	.LBB0_2
# %bb.1:
	movl	-20(%rbp), %eax
	movl	%eax, -4(%rbp)
	jmp	.LBB0_12
.LBB0_2:
	movl	-8(%rbp), %eax
	cmpl	-12(%rbp), %eax
	jge	.LBB0_11
# %bb.3:
	movl	-12(%rbp), %eax
	cmpl	-16(%rbp), %eax
```

```
  jge .LBB0_11
# %bb.4:
  jmp .LBB0_5
.LBB0_5:                                # =>This Loop Header: Depth=1
                                        #     Child Loop BB0_7 Depth 2
  movl  -8(%rbp), %eax
  cmpl  -12(%rbp), %eax
  je  .LBB0_10
# %bb.6:                                #   in Loop: Header=BB0_5
   Depth=1
  movl  -8(%rbp), %eax
  addl  $1, %eax
  movl  %eax, -8(%rbp)
  movl  fun.count, %eax
  addl  $1, %eax
  movl  %eax, fun.count
.LBB0_7:                                #   Parent Loop BB0_5 Depth=1
                                        # =>  This Inner Loop Header:
   Depth=2
  movl  -12(%rbp), %eax
  cmpl  -16(%rbp), %eax
  je  .LBB0_9
# %bb.8:                                #   in Loop: Header=BB0_7
   Depth=2
  movl  -16(%rbp), %eax
  addl  $-1, %eax
  movl  %eax, -16(%rbp)
  movl  fun.count, %eax
  addl  $1, %eax
  movl  %eax, fun.count
  jmp .LBB0_7
.LBB0_9:                                #   in Loop: Header=BB0_5
   Depth=1
  jmp .LBB0_5
.LBB0_10:
  jmp .LBB0_11
.LBB0_11:
  movl  fun.count, %eax
  movl  %eax, -4(%rbp)
```

```
.LBB0_12:
  movl  -4(%rbp), %eax
  popq  %rbp
  .cfi_def_cfa %rsp, 8
  retq
.Lfunc_end0:
  .size fun, .Lfunc_end0-fun
  .cfi_endproc
                                        # -- End function
  .globl  main                    # -- Begin function main
  .p2align  4, 0x90
  .type main,@function
main:                                   # @main
  .cfi_startproc
# %bb.0:
  pushq %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset %rbp, -16
  movq  %rsp, %rbp
  .cfi_def_cfa_register %rbp
  subq  $16, %rsp
  movl  $0, -4(%rbp)
  movl  $1, rooster
  movl  $5, drake
  movl  $10, goose
  movl  rooster, %edi
  movl  drake, %esi
  movl  goose, %edx
  callq fun
  xorl  %ecx, %ecx
  movl  %eax, -8(%rbp)
  movl  %ecx, %eax
  addq  $16, %rsp
  popq  %rbp
  .cfi_def_cfa %rsp, 8
  retq
.Lfunc_end1:
  .size main, .Lfunc_end1-main
  .cfi_endproc
```

```
                                        # -- End function
  .type fun.count,@object       # @fun.count
  .local  fun.count
  .comm fun.count,4,4
  .type rooster,@object         # @rooster
  .comm rooster,4,4
  .type drake,@object           # @drake
  .comm drake,4,4
  .type goose,@object           # @goose
  .comm goose,4,4
  .ident  "clang version 10.0.0-4ubuntu1 "
  .section  ".note.GNU-stack","",@progbits
  .addrsig
  .addrsig_sym fun
  .addrsig_sym fun.count
  .addrsig_sym rooster
  .addrsig_sym drake
  .addrsig_sym goose
```

F.10 rooster-O3.s

```
  .text
  .file "rooster.c"
  .globl  fun                     # -- Begin function fun
  .p2align  4, 0x90
  .type fun,@function
fun:                                    # @fun
  .cfi_startproc
# %bb.0:
                                        # kill: def $edx killed $edx
   def $rdx
                                        # kill: def $esi killed $esi
   def $rsi
                                        # kill: def $edi killed $edi
   def $rdi
  leal  (%rsi,%rdi), %eax
  addl  %edx, %eax
  js  .LBB0_9
```

```
# %bb.1:
	movl	fun.count(%rip), %eax
	cmpl	%esi, %edi
	jge	.LBB0_9
# %bb.2:
	cmpl	%edx, %esi
	jge	.LBB0_9
# %bb.3:
	leal	1(%rdi), %ecx
	cmpl	%esi, %ecx
	jne	.LBB0_5
# %bb.4:
	subl	%esi, %eax
	addl	%edx, %eax
	addl	$1, %eax
	jmp	.LBB0_8
.LBB0_5:
	addl	$-1, %esi
	.p2align	4, 0x90
.LBB0_6:                                # =>This Inner Loop Header: Depth=1
	addl	$-1, %esi
	cmpl	%esi, %edi
	jne	.LBB0_6
# %bb.7:
	addl	%edx, %eax
	subl	%esi, %eax
.LBB0_8:
	movl	%eax, fun.count(%rip)
.LBB0_9:
                                        # kill: def $eax killed $eax killed $rax
	retq
.Lfunc_end0:
	.size	fun, .Lfunc_end0-fun
	.cfi_endproc
                                        # -- End function
	.globl	main                    # -- Begin function main
	.p2align	4, 0x90
```

```
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
	movl	$1, rooster(%rip)
	movl	$5, drake(%rip)
	movl	$10, goose(%rip)
	addl	$9, fun.count(%rip)
	xorl	%eax, %eax
	retq
.Lfunc_end1:
	.size	main, .Lfunc_end1-main
	.cfi_endproc
                                        # -- End function
	.type	fun.count,@object       # @fun.count
	.local	fun.count
	.comm	fun.count,4,4
	.type	rooster,@object         # @rooster
	.comm	rooster,4,4
	.type	drake,@object           # @drake
	.comm	drake,4,4
	.type	goose,@object           # @goose
	.comm	goose,4,4
	.ident	"clang version 10.0.0-4ubuntu1 "
	.section	".note.GNU-stack","",@progbits
	.addrsig
```

F.11 password-O0.s

```
	.text
	.file	"password.c"
	.globl	setup_users             # -- Begin function setup_users
	.p2align	4, 0x90
	.type	setup_users,@function
setup_users:                            # @setup_users
	.cfi_startproc
# %bb.0:
	pushq	%rbp
```

```
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
    movq   %rsp, %rbp
    .cfi_def_cfa_register %rbp
    subq   $80, %rsp
    movl   $528, %edi                   # imm = 0x210
    callq  malloc
    movq   %rax, -8(%rbp)
    movq   -8(%rbp), %rax
    addq   $8, %rax
    movl   $.L.str, %esi
    movq   %rax, %rdi
    callq  strcpy
    movq   -8(%rbp), %rcx
    addq   $264, %rcx                   # imm = 0x108
    movl   $.L.str.1, %esi
    movq   %rcx, %rdi
    movq   %rax, -32(%rbp)              # 8-byte Spill
    callq  strcpy
    movq   -8(%rbp), %rcx
    movq   $1000000, 520(%rcx)          # imm = 0xF4240
    movl   $528, %edi                   # imm = 0x210
    movq   %rax, -40(%rbp)              # 8-byte Spill
    callq  malloc
    movq   %rax, -16(%rbp)
    movq   -16(%rbp), %rax
    addq   $8, %rax
    movl   $.L.str.2, %esi
    movq   %rax, %rdi
    callq  strcpy
    movq   -16(%rbp), %rcx
    addq   $264, %rcx                   # imm = 0x108
    movl   $.L.str.3, %esi
    movq   %rcx, %rdi
    movq   %rax, -48(%rbp)              # 8-byte Spill
    callq  strcpy
    movq   -16(%rbp), %rcx
    movq   $783, 520(%rcx)              # imm = 0x30F
    movl   $528, %edi                   # imm = 0x210
```

```
  movq  %rax, -56(%rbp)          # 8-byte Spill
  callq malloc
  movq  %rax, -24(%rbp)
  movq  -24(%rbp), %rax
  addq  $8, %rax
  movl  $.L.str.4, %esi
  movq  %rax, %rdi
  callq strcpy
  movq  -24(%rbp), %rcx
  addq  $264, %rcx               # imm = 0x108
  movl  $.L.str.5, %esi
  movq  %rcx, %rdi
  movq  %rax, -64(%rbp)          # 8-byte Spill
  callq strcpy
  movq  -24(%rbp), %rcx
  movq  $2, 520(%rcx)
  movq  -16(%rbp), %rcx
  movq  -8(%rbp), %rdx
  movq  %rcx, (%rdx)
  movq  -24(%rbp), %rcx
  movq  -16(%rbp), %rdx
  movq  %rcx, (%rdx)
  movq  -24(%rbp), %rcx
  movq  $0, (%rcx)
  movq  -8(%rbp), %rcx
  movq  %rax, -72(%rbp)          # 8-byte Spill
  movq  %rcx, %rax
  addq  $80, %rsp
  popq  %rbp
  .cfi_def_cfa %rsp, 8
  retq
.Lfunc_end0:
  .size setup_users, .Lfunc_end0-setup_users
  .cfi_endproc
                                        # -- End function
  .globl  print_users            # -- Begin function print_users
  .p2align  4, 0x90
  .type print_users,@function
print_users:                            # @print_users
```

```
  .cfi_startproc
# %bb.0:
  pushq %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset %rbp, -16
  movq  %rsp, %rbp
  .cfi_def_cfa_register %rbp
  subq  $16, %rsp
  movq  %rdi, -8(%rbp)
  movabsq $.L.str.6, %rdi
  movb  $0, %al
  callq printf
  movq  $0, -16(%rbp)
.LBB1_1:                                # =>This Inner Loop Header:
   Depth=1
  cmpq  $0, -8(%rbp)
  je   .LBB1_3
# %bb.2:                                #   in Loop: Header=BB1_1
   Depth=1
  movq  -16(%rbp), %rax
  addq  $1, %rax
  movq  %rax, -16(%rbp)
  movq  -8(%rbp), %rcx
  addq  $8, %rcx
  movabsq $.L.str.7, %rdi
  movq  %rax, %rsi
  movq  %rcx, %rdx
  movb  $0, %al
  callq printf
  movq  -8(%rbp), %rcx
  movq  (%rcx), %rcx
  movq  %rcx, -8(%rbp)
  jmp .LBB1_1
.LBB1_3:
  movabsq $.L.str.8, %rdi
  movb  $0, %al
  callq printf
  addq  $16, %rsp
  popq  %rbp
```

```
	.cfi_def_cfa %rsp, 8
	retq
.Lfunc_end1:
	.size print_users, .Lfunc_end1-print_users
	.cfi_endproc
                                        # -- End function
	.globl	getUser                 # -- Begin function getUser
	.p2align	4, 0x90
	.type getUser,@function
getUser:                                # @getUser
	.cfi_startproc
# %bb.0:
	pushq %rbp
	.cfi_def_cfa_offset 16
	.cfi_offset %rbp, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register %rbp
	subq	$32, %rsp
	movq	%rdi, -16(%rbp)
	movq	%rsi, -24(%rbp)
.LBB2_1:                                # =>This Inner Loop Header: Depth=1
	cmpq	$0, -16(%rbp)
	je	.LBB2_5
# %bb.2:                                #   in Loop: Header=BB2_1 Depth=1
	movq	-16(%rbp), %rax
	addq	$8, %rax
	movq	-24(%rbp), %rsi
	movq	%rax, %rdi
	callq strcmp
	cmpl	$0, %eax
	jne .LBB2_4
# %bb.3:
	movq	-16(%rbp), %rax
	movq	%rax, -8(%rbp)
	jmp .LBB2_6
.LBB2_4:                                #   in Loop: Header=BB2_1 Depth=1
```

```
  movq  -16(%rbp), %rax
  movq  (%rax), %rax
  movq  %rax, -16(%rbp)
  jmp .LBB2_1
.LBB2_5:
  movq  $0, -8(%rbp)
.LBB2_6:
  movq  -8(%rbp), %rax
  addq  $32, %rsp
  popq  %rbp
  .cfi_def_cfa %rsp, 8
  retq
.Lfunc_end2:
  .size getUser, .Lfunc_end2-getUser
  .cfi_endproc
                                        # -- End function
  .globl  main                    # -- Begin function main
  .p2align  4, 0x90
  .type main,@function
main:                                   # @main
  .cfi_startproc
# %bb.0:
  pushq %rbp
  .cfi_def_cfa_offset 16
  .cfi_offset %rbp, -16
  movq  %rsp, %rbp
  .cfi_def_cfa_register %rbp
  subq  $576, %rsp              # imm = 0x240
  movl  $0, -4(%rbp)
  callq setup_users
  movq  %rax, -16(%rbp)
  movabsq $.L.str.9, %rdi
  movb  $0, %al
  callq printf
  movabsq $.L.str.10, %rdi
  movl  %eax, -548(%rbp)        # 4-byte Spill
  movb  $0, %al
  callq printf
  leaq  -272(%rbp), %rsi
```

```
  movabsq $.L.str.11, %rdi
  movl   %eax, -552(%rbp)         # 4-byte Spill
  movb   $0, %al
  callq __isoc99_scanf
  leaq   -272(%rbp), %rsi
  movq   -16(%rbp), %rdi
  movl   %eax, -556(%rbp)         # 4-byte Spill
  callq getUser
  movq   %rax, -280(%rbp)
  cmpq   $0, -280(%rbp)
  jne .LBB3_2
# %bb.1:
  leaq   -272(%rbp), %rsi
  movabsq $.L.str.12, %rdi
  movb   $0, %al
  callq printf
  movl   $0, -4(%rbp)
  jmp .LBB3_5
.LBB3_2:
  movabsq $.L.str.13, %rdi
  movb   $0, %al
  callq printf
  leaq   -544(%rbp), %rsi
  movabsq $.L.str.11, %rdi
  movl   %eax, -560(%rbp)         # 4-byte Spill
  movb   $0, %al
  callq __isoc99_scanf
  leaq   -544(%rbp), %rsi
  movq   -280(%rbp), %rcx
  addq   $264, %rcx               # imm = 0x108
  movq   %rcx, %rdi
  movl   %eax, -564(%rbp)         # 4-byte Spill
  callq strcmp
  cmpl   $0, %eax
  je   .LBB3_4
# %bb.3:
  movabsq $.L.str.14, %rdi
  movb   $0, %al
  callq printf
```

```
  movl  $0, -4(%rbp)
  jmp .LBB3_5
.LBB3_4:
  movq  -280(%rbp), %rax
  addq  $8, %rax
  movabsq $.L.str.15, %rdi
  movq  %rax, %rsi
  movb  $0, %al
  callq printf
  movabsq $.L.str.8, %rdi
  movl  %eax, -568(%rbp)        # 4-byte Spill
  movb  $0, %al
  callq printf
  movq  -280(%rbp), %rcx
  addq  $8, %rcx
  movabsq $.L.str.16, %rdi
  movq  %rcx, %rsi
  movl  %eax, -572(%rbp)        # 4-byte Spill
  movb  $0, %al
  callq printf
  movq  -280(%rbp), %rcx
  movq  520(%rcx), %rsi
  movabsq $.L.str.17, %rdi
  movl  %eax, -576(%rbp)        # 4-byte Spill
  movb  $0, %al
  callq printf
.LBB3_5:
  movl  -4(%rbp), %eax
  addq  $576, %rsp              # imm = 0x240
  popq  %rbp
  .cfi_def_cfa %rsp, 8
  retq
.Lfunc_end3:
  .size main, .Lfunc_end3-main
  .cfi_endproc
                                        # -- End function
  .type .L.str,@object          # @.str
  .section  .rodata.str1.1,"aMS",@progbits,1
.L.str:
```

```
	.asciz	"admin"
	.size	.L.str, 6

	.type	.L.str.1,@object        # @.str.1
.L.str.1:
	.asciz	"4dm1n__4eva"
	.size	.L.str.1, 12

	.type	.L.str.2,@object        # @.str.2
.L.str.2:
	.asciz	"alice"
	.size	.L.str.2, 6

	.type	.L.str.3,@object        # @.str.3
.L.str.3:
	.asciz	"!alice12!_veuje@@hak"
	.size	.L.str.3, 21

	.type	.L.str.4,@object        # @.str.4
.L.str.4:
	.asciz	"abdul"
	.size	.L.str.4, 6

	.type	.L.str.5,@object        # @.str.5
.L.str.5:
	.asciz	"passw0rd123"
	.size	.L.str.5, 12

	.type	.L.str.6,@object        # @.str.6
.L.str.6:
	.asciz	"--- USERS ---\n"
	.size	.L.str.6, 15

	.type	.L.str.7,@object        # @.str.7
.L.str.7:
	.asciz	" %02ld. %s\n"
	.size	.L.str.7, 12

	.type	.L.str.8,@object        # @.str.8
```

```
.L.str.8:
  .asciz  "\n"
  .size .L.str.8, 2

  .type .L.str.9,@object        # @.str.9
.L.str.9:
  .asciz  "Welcome to BigBank Australia!\n"
  .size .L.str.9, 31

  .type .L.str.10,@object       # @.str.10
.L.str.10:
  .asciz  "Username: "
  .size .L.str.10, 11

  .type .L.str.11,@object       # @.str.11
.L.str.11:
  .asciz  "%255s"
  .size .L.str.11, 6

  .type .L.str.12,@object       # @.str.12
.L.str.12:
  .asciz  "User < %s > does not exist.\n"
  .size .L.str.12, 29

  .type .L.str.13,@object       # @.str.13
.L.str.13:
  .asciz  "Password: "
  .size .L.str.13, 11

  .type .L.str.14,@object       # @.str.14
.L.str.14:
  .asciz  "ERROR: incorrect password\n"
  .size .L.str.14, 27

  .type .L.str.15,@object       # @.str.15
.L.str.15:
  .asciz  "Logged in as < %s >!\n"
  .size .L.str.15, 22

```

```
	.type	.L.str.16,@object	# @.str.16
.L.str.16:
	.asciz	"Welcome, %s!\n"
	.size	.L.str.16, 14

	.type	.L.str.17,@object	# @.str.17
.L.str.17:
	.asciz	"Your balance: $%ld\n"
	.size	.L.str.17, 20

	.ident	"clang version 10.0.0-4ubuntu1 "
	.section	".note.GNU-stack","",@progbits
	.addrsig
	.addrsig_sym setup_users
	.addrsig_sym malloc
	.addrsig_sym strcpy
	.addrsig_sym printf
	.addrsig_sym getUser
	.addrsig_sym strcmp
	.addrsig_sym __isoc99_scanf
```

F.12 password-O3.s

```
	.text
	.file	"password.c"
	.globl	setup_users	# -- Begin function setup_users
	.p2align	4, 0x90
	.type	setup_users,@function
setup_users:	# @setup_users
	.cfi_startproc
# %bb.0:
	pushq	%r14
	.cfi_def_cfa_offset 16
	pushq	%rbx
	.cfi_def_cfa_offset 24
	pushq	%rax
	.cfi_def_cfa_offset 32
	.cfi_offset %rbx, -24
```

```
.cfi_offset %r14, -16
movl  $528, %edi               # imm = 0x210
callq malloc
movq  %rax, %r14
movl  $1768776801, 8(%rax)     # imm = 0x696D6461
movw  $110, 12(%rax)
movabsq $3773839939640058932, %rax # imm = 0x345F5F6E316D6434
movq  %rax, 264(%r14)
movl  $6387301, 272(%r14)      # imm = 0x617665
movq  $1000000, 520(%r14)      # imm = 0xF4240
movl  $528, %edi               # imm = 0x210
callq malloc
movq  %rax, %rbx
movl  $1667853409, 8(%rax)     # imm = 0x63696C61
movw  $101, 12(%rax)
movabsq $30224922890495338, %rax # imm = 0x6B61684040656A
movq  %rax, 277(%rbx)
movups  .L.str.3(%rip), %xmm0
movups  %xmm0, 264(%rbx)
movq  $783, 520(%rbx)          # imm = 0x30F
movl  $528, %edi               # imm = 0x210
callq malloc
movl  $1969513057, 8(%rax)     # imm = 0x75646261
movw  $108, 12(%rax)
movabsq $7237900840733991280, %rcx # imm = 0x6472307773736170
movq  %rcx, 264(%rax)
movl  $3355185, 272(%rax)      # imm = 0x333231
movq  $2, 520(%rax)
movq  %rbx, (%r14)
movq  %rax, (%rbx)
movq  $0, (%rax)
movq  %r14, %rax
addq  $8, %rsp
.cfi_def_cfa_offset 24
popq  %rbx
.cfi_def_cfa_offset 16
popq  %r14
.cfi_def_cfa_offset 8
retq
```

```
.Lfunc_end0:
  .size setup_users, .Lfunc_end0-setup_users
  .cfi_endproc
                                        # -- End function
  .globl  print_users             # -- Begin function print_users
  .p2align  4, 0x90
  .type print_users,@function
print_users:                            # @print_users
  .cfi_startproc
# %bb.0:
  pushq %r14
  .cfi_def_cfa_offset 16
  pushq %rbx
  .cfi_def_cfa_offset 24
  pushq %rax
  .cfi_def_cfa_offset 32
  .cfi_offset %rbx, -24
  .cfi_offset %r14, -16
  movq  %rdi, %r14
  movl  $.Lstr, %edi
  callq puts
  testq %r14, %r14
  je  .LBB1_3
# %bb.1:
  movl  $1, %ebx
  .p2align  4, 0x90
.LBB1_2:                                # =>This Inner Loop Header: Depth=1
  leaq  8(%r14), %rdx
  movl  $.L.str.7, %edi
  movq  %rbx, %rsi
  xorl  %eax, %eax
  callq printf
  movq  (%r14), %r14
  addq  $1, %rbx
  testq %r14, %r14
  jne .LBB1_2
.LBB1_3:
  movl  $10, %edi
```

```
  addq  $8, %rsp
  .cfi_def_cfa_offset 24
  popq  %rbx
  .cfi_def_cfa_offset 16
  popq  %r14
  .cfi_def_cfa_offset 8
  jmp putchar                 # TAILCALL
.Lfunc_end1:
  .size print_users, .Lfunc_end1-print_users
  .cfi_endproc
                                        # -- End function
  .globl  getUser                 # -- Begin function getUser
  .p2align  4, 0x90
  .type getUser,@function
getUser:                                # @getUser
  .cfi_startproc
# %bb.0:
  pushq %r14
  .cfi_def_cfa_offset 16
  pushq %rbx
  .cfi_def_cfa_offset 24
  pushq %rax
  .cfi_def_cfa_offset 32
  .cfi_offset %rbx, -24
  .cfi_offset %r14, -16
  testq %rdi, %rdi
  je  .LBB2_4
# %bb.1:
  movq  %rsi, %r14
  movq  %rdi, %rbx
  .p2align  4, 0x90
.LBB2_2:                                # =>This Inner Loop Header:
   Depth=1
  leaq  8(%rbx), %rdi
  movq  %r14, %rsi
  callq strcmp
  testl %eax, %eax
  je  .LBB2_5
# %bb.3:                                #   in Loop: Header=BB2_2
```

```
   Depth=1
  movq   (%rbx), %rbx
  testq %rbx, %rbx
  jne  .LBB2_2
.LBB2_4:
  xorl   %ebx, %ebx
.LBB2_5:
  movq   %rbx, %rax
  addq   $8, %rsp
  .cfi_def_cfa_offset 24
  popq   %rbx
  .cfi_def_cfa_offset 16
  popq   %r14
  .cfi_def_cfa_offset 8
  retq
.Lfunc_end2:
  .size getUser, .Lfunc_end2-getUser
  .cfi_endproc
                                        # -- End function
  .globl  main                          # -- Begin function main
  .p2align  4, 0x90
  .type main,@function
main:                                   # @main
  .cfi_startproc
# %bb.0:
  pushq %r15
  .cfi_def_cfa_offset 16
  pushq %r14
  .cfi_def_cfa_offset 24
  pushq %rbx
  .cfi_def_cfa_offset 32
  subq   $512, %rsp                   # imm = 0x200
  .cfi_def_cfa_offset 544
  .cfi_offset %rbx, -32
  .cfi_offset %r14, -24
  .cfi_offset %r15, -16
  movl   $528, %edi                   # imm = 0x210
  callq malloc
  movq   %rax, %rbx
```

```
    movl   $1768776801, 8(%rax)     # imm = 0x696D6461
    movw   $110, 12(%rax)
    movabsq $3773839939640058932, %rax # imm = 0x345F5F6E316D6434
    movq   %rax, 264(%rbx)
    movl   $6387301, 272(%rbx)      # imm = 0x617665
    movq   $1000000, 520(%rbx)      # imm = 0xF4240
    movl   $528, %edi               # imm = 0x210
    callq  malloc
    movq   %rax, %r14
    movl   $1667853409, 8(%rax)     # imm = 0x63696C61
    movw   $101, 12(%rax)
    movabsq $3022492289049533 8, %rax # imm = 0x6B61684040656A
    movq   %rax, 277(%r14)
    movups  .L.str.3(%rip), %xmm0
    movups  %xmm0, 264(%r14)
    movq   $783, 520(%r14)          # imm = 0x30F
    movl   $528, %edi               # imm = 0x210
    callq  malloc
    movl   $1969513057, 8(%rax)     # imm = 0x75646261
    movw   $108, 12(%rax)
    movabsq $7237900840733991280, %rcx # imm = 0x6472307773736170
    movq   %rcx, 264(%rax)
    movl   $3355185, 272(%rax)      # imm = 0x333231
    movq   $2, 520(%rax)
    movq   %r14, (%rbx)
    movq   %rax, (%r14)
    movq   $0, (%rax)
    movl   $.Lstr.18, %edi
    callq  puts
    movl   $.L.str.10, %edi
    xorl   %eax, %eax
    callq  printf
    movq   %rsp, %rsi
    movl   $.L.str.11, %edi
    xorl   %eax, %eax
    callq  __isoc99_scanf
    testq  %rbx, %rbx
    je     .LBB3_4
# %bb.1:
```

```
  movq  %rsp, %r15
  .p2align  4, 0x90
.LBB3_2:                                # =>This Inner Loop Header: Depth=1
  leaq  8(%rbx), %r14
  movq  %r14, %rdi
  movq  %r15, %rsi
  callq strcmp
  testl %eax, %eax
  je    .LBB3_5
# %bb.3:                                #   in Loop: Header=BB3_2 Depth=1
  movq  (%rbx), %rbx
  testq %rbx, %rbx
  jne .LBB3_2
.LBB3_4:
  movq  %rsp, %rsi
  movl  $.L.str.12, %edi
.LBB3_8:
  xorl  %eax, %eax
  callq printf
  jmp .LBB3_9
.LBB3_5:
  movl  $.L.str.13, %edi
  xorl  %eax, %eax
  callq printf
  leaq  256(%rsp), %r15
  movl  $.L.str.11, %edi
  movq  %r15, %rsi
  xorl  %eax, %eax
  callq __isoc99_scanf
  leaq  264(%rbx), %rdi
  movq  %r15, %rsi
  callq strcmp
  testl %eax, %eax
  je    .LBB3_7
# %bb.6:
  movl  $.Lstr.19, %edi
  callq puts
```

```
.LBB3_9:
	xorl	%eax, %eax
	addq	$512, %rsp                     # imm = 0x200
	.cfi_def_cfa_offset 32
	popq	%rbx
	.cfi_def_cfa_offset 24
	popq	%r14
	.cfi_def_cfa_offset 16
	popq	%r15
	.cfi_def_cfa_offset 8
	retq
.LBB3_7:
	.cfi_def_cfa_offset 544
	movl	$.L.str.15, %edi
	movq	%r14, %rsi
	xorl	%eax, %eax
	callq	printf
	movl	$10, %edi
	callq	putchar
	movl	$.L.str.16, %edi
	movq	%r14, %rsi
	xorl	%eax, %eax
	callq	printf
	movq	520(%rbx), %rsi
	movl	$.L.str.17, %edi
	jmp	.LBB3_8
.Lfunc_end3:
	.size	main, .Lfunc_end3-main
	.cfi_endproc
                                        # -- End function
	.type	.L.str,@object          # @.str
	.section	.rodata.str1.1,"aMS",@progbits,1
.L.str:
	.asciz	"admin"
	.size	.L.str, 6

	.type	.L.str.1,@object        # @.str.1
.L.str.1:
	.asciz	"4dm1n__4eva"
```

```
  .size .L.str.1, 12

  .type .L.str.2,@object          # @.str.2
.L.str.2:
  .asciz  "alice"
  .size .L.str.2, 6

  .type .L.str.3,@object          # @.str.3
.L.str.3:
  .asciz  "!alice12!_veuje@@hak"
  .size .L.str.3, 21

  .type .L.str.4,@object          # @.str.4
.L.str.4:
  .asciz  "abdul"
  .size .L.str.4, 6

  .type .L.str.5,@object          # @.str.5
.L.str.5:
  .asciz  "passw0rd123"
  .size .L.str.5, 12

  .type .L.str.7,@object          # @.str.7
.L.str.7:
  .asciz  " %02ld. %s\n"
  .size .L.str.7, 12

  .type .L.str.10,@object         # @.str.10
.L.str.10:
  .asciz  "Username: "
  .size .L.str.10, 11

  .type .L.str.11,@object         # @.str.11
.L.str.11:
  .asciz  "%255s"
  .size .L.str.11, 6

  .type .L.str.12,@object         # @.str.12
.L.str.12:
```

```
  .asciz  "User < %s > does not exist.\n"
  .size .L.str.12, 29

  .type .L.str.13,@object       # @.str.13
.L.str.13:
  .asciz  "Password: "
  .size .L.str.13, 11

  .type .L.str.15,@object       # @.str.15
.L.str.15:
  .asciz  "Logged in as < %s >!\n"
  .size .L.str.15, 22

  .type .L.str.16,@object       # @.str.16
.L.str.16:
  .asciz  "Welcome, %s!\n"
  .size .L.str.16, 14

  .type .L.str.17,@object       # @.str.17
.L.str.17:
  .asciz  "Your balance: $%ld\n"
  .size .L.str.17, 20

  .type .Lstr,@object           # @str
.Lstr:
  .asciz  "--- USERS ---"
  .size .Lstr, 14

  .type .Lstr.18,@object        # @str.18
.Lstr.18:
  .asciz  "Welcome to BigBank Australia!"
  .size .Lstr.18, 30

  .type .Lstr.19,@object        # @str.19
.Lstr.19:
  .asciz  "ERROR: incorrect password"
  .size .Lstr.19, 26

  .ident  "clang version 10.0.0-4ubuntu1 "
```

```
    .section  ".note.GNU-stack","",@progbits
    .addrsig
```

F.13 deadStoreElimination-O0.s

```
    .text
    .file "deadStoreElimination.c"
    .globl  deadStore                # -- Begin function deadStore
    .p2align  4, 0x90
    .type deadStore,@function
deadStore:                               # @deadStore
    .cfi_startproc
# %bb.0:
    pushq %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset %rbp, -16
    movq  %rsp, %rbp
    .cfi_def_cfa_register %rbp
    movl  %edi, -4(%rbp)
    movl  %esi, -8(%rbp)
    movl  $43981, -12(%rbp)       # imm = 0xABCD
    movl  $0, -16(%rbp)
.LBB0_1:                                 # =>This Inner Loop Header:
     Depth=1
    movl  -4(%rbp), %eax
    cmpl  -8(%rbp), %eax
    jle .LBB0_3
# %bb.2:                                 #   in Loop: Header=BB0_1
     Depth=1
    movl  -12(%rbp), %eax
    addl  -16(%rbp), %eax
    movl  %eax, -16(%rbp)
    movl  -4(%rbp), %eax
    addl  $-1, %eax
    movl  %eax, -4(%rbp)
    jmp .LBB0_1
.LBB0_3:
    movl  $0, -12(%rbp)
```

```
	movl	-4(%rbp), %eax
	addl	-8(%rbp), %eax
	popq	%rbp
	.cfi_def_cfa %rsp, 8
	retq
.Lfunc_end0:
	.size	deadStore, .Lfunc_end0-deadStore
	.cfi_endproc
                                        # -- End function
	.globl	main                    # -- Begin function main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset %rbp, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register %rbp
	subq	$32, %rsp
	movl	%edi, -4(%rbp)
	movq	%rsi, -16(%rbp)
	movl	-4(%rbp), %edi
	movl	$2, %esi
	callq	deadStore
	xorl	%ecx, %ecx
	movl	%eax, -20(%rbp)         # 4-byte Spill
	movl	%ecx, %eax
	addq	$32, %rsp
	popq	%rbp
	.cfi_def_cfa %rsp, 8
	retq
.Lfunc_end1:
	.size	main, .Lfunc_end1-main
	.cfi_endproc
                                        # -- End function
	.ident	"clang version 10.0.0-4ubuntu1 "
	.section	".note.GNU-stack","",@progbits
```

```
	.addrsig
	.addrsig_sym deadStore
```

F.14 deadStoreElimination-O3.s

```
	.text
	.file	"deadStoreElimination.c"
	.globl	deadStore               # -- Begin function deadStore
	.p2align	4, 0x90
	.type	deadStore,@function
deadStore:                              # @deadStore
	.cfi_startproc
# %bb.0:
                                        # kill: def $esi killed $esi def $rsi
                                        # kill: def $edi killed $edi def $rdi
	cmpl	%edi, %esi
	cmovlel	%esi, %edi
	leal	(%rdi,%rsi), %eax
	retq
.Lfunc_end0:
	.size	deadStore, .Lfunc_end0-deadStore
	.cfi_endproc
                                        # -- End function
	.globl	main                    # -- Begin function main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
	xorl	%eax, %eax
	retq
.Lfunc_end1:
	.size	main, .Lfunc_end1-main
	.cfi_endproc
                                        # -- End function
	.ident	"clang version 10.0.0-4ubuntu1 "
```

```
	.section	".note.GNU-stack","",@progbits
	.addrsig
```

F.15 pread-O0.s

```
	.text
	.file	"pread.c"
	.globl	main                    # -- Begin function main
	.p2align	4, 0x90
	.type	main,@function
main:                                   # @main
	.cfi_startproc
# %bb.0:
	pushq	%rbp
	.cfi_def_cfa_offset 16
	.cfi_offset %rbp, -16
	movq	%rsp, %rbp
	.cfi_def_cfa_register %rbp
	movl	$0, -4(%rbp)
	movl	$0, -8(%rbp)
	movl	$0, -12(%rbp)
.LBB0_1:                                # =>This Loop Header: Depth=1
                                        #     Child Loop BB0_2 Depth 2
                                        #       Child Loop BB0_3 Depth 3
	jmp	.LBB0_2
.LBB0_2:                                #   Parent Loop BB0_1 Depth=1
                                        # =>  This Loop Header: Depth=2
                                        #       Child Loop BB0_3 Depth 3
	jmp	.LBB0_3
.LBB0_3:                                #   Parent Loop BB0_1 Depth=1
                                        #     Parent Loop BB0_2 Depth=2
                                        # =>    This Inner Loop Header: Depth=3
	movl	z, %eax
	movl	%eax, -8(%rbp)
# %bb.4:                                #   in Loop: Header=BB0_3
```

```
 Depth=3
	movl	-8(%rbp), %eax
	cltd
	movl	$2, %ecx
	idivl	%ecx
	cmpl	$0, %edx
	jne	.LBB0_3
# %bb.5:                                #   in Loop: Header=BB0_2 Depth=2
	movl	x, %eax
	movl	%eax, -12(%rbp)
# %bb.6:                                #   in Loop: Header=BB0_2 Depth=2
	movl	z, %eax
	cmpl	-8(%rbp), %eax
	jne	.LBB0_2
# %bb.7:                                #   in Loop: Header=BB0_1 Depth=1
	jmp	.LBB0_1
.Lfunc_end0:
	.size	main, .Lfunc_end0-main
	.cfi_endproc
                                        # -- End function
	.type	z,@object               # @z
	.comm	z,4,4
	.type	x,@object               # @x
	.comm	x,4,4
	.ident	"clang version 10.0.0-4ubuntu1 "
	.section	".note.GNU-stack","",@progbits
	.addrsig
	.addrsig_sym z
	.addrsig_sym x
```

F.16 pread-O3.s

```
	.text
	.file	"pread.c"
	.globl	main                    # -- Begin function main
```

```
  .p2align  4, 0x90
  .type main,@function
main:                                   # @main
  .cfi_startproc
# %bb.0:
  .p2align  4, 0x90
.LBB0_1:                                # =>This Inner Loop Header: Depth=1
  movl  z(%rip), %eax
  testb $1, %al
  jne .LBB0_1
# %bb.2:                                #   in Loop: Header=BB0_1 Depth=1
  movl  x(%rip), %eax
  movl  z(%rip), %eax
  jmp .LBB0_1
.Lfunc_end0:
  .size main, .Lfunc_end0-main
  .cfi_endproc
                                        # -- End function
  .type z,@object               # @z
  .comm z,4,4
  .type x,@object               # @x
  .comm x,4,4
  .ident  "clang version 10.0.0-4ubuntu1 "
  .section  ".note.GNU-stack","",@progbits
  .addrsig
  .addrsig_sym z
  .addrsig_sym x
```