SYNOPSYS®

Understanding ABI Compatibility between Compilers targeting RISC-V

ABI Extractor

Luis Silva

February 1st, 2025

Agenda

- Introduction
- ABI Extractor
- Implementation details
- Conclusion
- QA

Agenda

- Introduction
  - Problem Statement
  - What is an Application Binary Interface?
  - Related Projects
- ABI Extractor
- Implementation details
- Conclusion
- QA

Problem Statement

Problem Statement

- Goal
  - We want to link RISC-V **object files from different compilers** and be sure the combination will work.

- Our focus
  - **Identify potential incompatibilities** in the ABI's used by different compilers (and their options)
  - **Identify deviations from the standard ABI**

What is an Application Binary Interface?

What is an Application Binary Interface?

- An ABI specifies the **interface** between binary program modules
  - (e.g., object files ; a program that depends on a library ; …)
- Defines **low-level details**
  - Properties of fundamental types
  - Calling convention
  - Main usage of registers
- Defines **higher level details**
  - C++ naming convention
  - Library interfaces
  - System Call interface
- Managed by compilers, operating systems or library developers.

Our focus: compatibility between compilers for a specific target

https://en.wikipedia.org/wiki/Application_binary_interface

What is an Application Binary Interface?

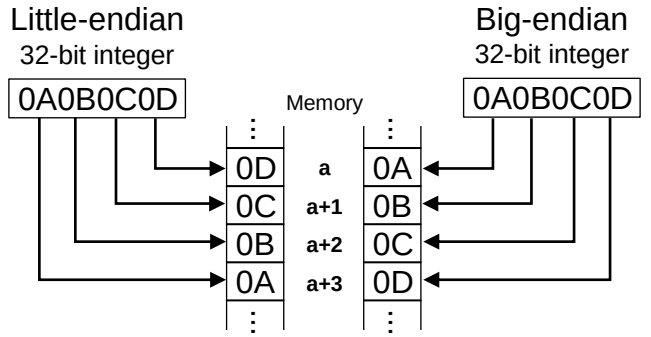
- Register usage
  - Specifies the **purpose** of registers
    - Stack pointer / Frame pointer /…
    - passing arguments
    - returning values
    - preserving state during function calls

Register	ABI Name	Description	Saver
x0	zero	Hard-wired zero	---
x1	ra	Return address	Caller
x2	sp	Stack pointer	Callee
x3	gp	Global pointer	---
x4	tp	Thread pointer	---
x5-7	t0-2	Temporaries	Caller
x8	s0/fp	Saved register/frame pointer	Callee
x9	s1	Saved register	Callee
x10-11	a0-1	Function arguments/return value	Caller
x12-17	a2-a7	Function arguments	Caller
x18-27	s2-11	Saved registers	Callee
x28-31	t3-6	Temporaries	Caller
f0-7	ft0-7	FP temporaries	Caller
f8-9	fs0-1	FP saved registers	Callee
f10-11	fa0-1	FP arguments/return values	Caller
f12-17	fa2-7	FP arguments	Caller
f18-27	fa2-11	FP saved registers	Callee
f28-31	ft8-11	FP temporaries	Caller

RISC-V ABIs Specification (Version 1.0): https://drive.google.com/file/d/1Ja_Tpp_5Me583CGVD-BIZMlgGBnlKU4R/view

What is an Application Binary Interface?

- Data layout
  - Specifies the **sizes** and **alignments** of fundamental C/C++ types
  - Specifies how **structs/unions/classes** are **laid out in memory**
  - Specifies how **bitfields** are laid out in a struct



		RV32		RV64	
C type	Description	Size	Alignment	Size	Alignment
char	Character value/byte	1	1	1	1
short	Short Integer	2	2	2	2
int	Integer	4	4	4	4
long	Long integer	4	4	8	8
long long	Long long integer	8	8	8	8
void*	Pointer	4	4	8	8
float	Single-precision float	4	4	4	4
double	Double-precision float	8	8	8	8
long double	Extended-precision float	16	16	16	16

RISC-V ABIs Specification (Version 1.0): https://drive.google.com/file/d/1Ja_Tpp_5Me583CGVD-BIZMlgGBnlKU4R/view
https://agilescientific.com/blog/2017/3/31/little-endian-is-legal

What is an Application Binary Interface?

- Calling conventions
  - Which **registers** or **memory** locations are used to pass arguments
  - How a return value is passed (e.g., in a register or on the stack)
  - Who is responsible for saving used registers
  - The stack frame structure and how it is managed

```
1  extern void dump (short, short, short,
2                    short, short, short,
3                    short, short, short);
4
5  int
6  main (void)
7  {
8     dump (1,2,3,4,5,6,7,8,9);
9     return 123;
10  }
```

```
1  main:
2        addi    sp,sp,-32
3        sw      ra,28(sp)
4        sw      s0,24(sp)
5        addi    s0,sp,32
6        li      a5,9
7        sw      a5,0(sp)
8        li      a7,8
9        li      a6,7
10       li      a5,6
11       li      a4,5
12       li      a3,4
13       li      a2,3
14       li      a1,2
15       li      a0,1
16       call    dump(short, short, short,
short, short, short, short, short, short)
17       li      a5,123
18       mv      a0,a5
19       lw      ra,28(sp)
20       lw      s0,24(sp)
21       addi    sp,sp,32
22       jr      ra
```

https://godbolt.org/z/3Ehj31vh4

What is an Application Binary Interface?

- Libraries
  - Header file compatibility (different c/c++ library implementations)
  - Ensure functions from libraries can be called correctly, whether linked statically or dynamically, by following established conventions.
  - Specifies how functions are named and located in the binary.
  - Version compatibility.
- Exception handling (C++)
  - Defines how exceptions are raised, caught, and how errors are communicated between the caller and the handler.
- Systems
  - Allows programs to run on different OS versions or hardware without recompilation.

https://agilescientific.com/blog/2017/3/31/little-endian-is-legal

Related Projects

Related Projects – **ABI Dump**

- Part of ‘ABI Compliance Checker’
- Goal: **Extract ABI properties from libraries**
  - Symbol information
    - Parameter types
    - Return types
    - Register usage
  - Type information
    - Datatypes and their sizes
  - Undefined symbols

```c
int add (int a, int b)
{
      return a + b;
}
```

```
'Arch' => 'x86_64',
'GccVersion' => '12.2.0',
'Headers' => {},
'Language' => 'C',
'LibraryName' => 'libmath.so',
'LibraryVersion' => undef,
'NameSpaces' => {},
'Needed' => {},
'Sources' => {
    'test.c' => 1
},
'SymbolInfo' => {
    '50' => {
        'Line' => '3',
        'Param' => {
            '0' => {
                'name' => 'a',
                'type' => '105'
            },
            '1' => {
                'name' => 'b',
                'type' => '105'
            }
        },
        'Reg' => {
            '0' => 'rdi',
            '1' => 'rsi'
        },
        'Return' => '105',
        'ShortName' => 'add',
        'Source' => 'test.c'
    }
},
'SymbolVersion' => {},
'Symbols' => {
    'libmath.so' => {
        'add' => 1
    }
},
'Target' => 'unix',
'TypeInfo' => {
    '105' => {
        'Name' => 'int',
        'Size' => '4',
        'Type' => 'Intrinsic'
    }
},
'UndefinedSymbols' => {
    'libmath.so' => {
        '_ITM_deregisterTMCloneTable' => 0,
        '_ITM_registerTMCloneTable' => 0,
        '__cxa_finalize' => 0,
        '__gmon_start__' => 0
    }
},
```

- How: Analyze **DWARF** debug information.

https://github.com/lvc/abi-dumper

Related Projects – **ABI Compliance Checker**

- Goal: Ensure **backward** binary and source-level **compatibility of C/C++ library**
  - Changes in calling stack
  - V-Table changes
  - Removed symbols
  - Renamed symbols

- How: Analyze **DWARF** debug information (through **ABI Dump**)

https://lvc.github.io/abi-compliance-checker/

Related Projects – **libabigail**

- Goal: Detect **interface incompatibilities** in **ELF shared libraries**.

- How:
  - Analyze ELF shared libraries and their associated debug information (**DWARF**) to build internal representations of exported functions and variables.
  - **Identify changes** in function/variable symbols and their data types (e.g., return type changes).

https://rseng.github.io/rseng/software/libabigail
https://sourceware.org/libabigail/manual/libabigail-overview.html

Related Projects – **ABI Cafe**

- Goal: Identify **if two languages/compilers agree** on their ABIs.

- How:
  - **Brute force testing** by generating random testcases focused on detecting ABI incompatibilities
    - *“If they agree, great!”*
    - *“If they don’t agree, even better, we just learned something!”*
  - Essentially an ABI fuzzer

https://github.com/Gankra/abi-cafe

Agenda

- Introduction
- ABI Extractor
  - Summary Report
  - Design Choices
- Implementation details
- Conclusion
- QA

Summary Report

Summary Report

RISC-V 32-bit ABI-Extractor

```
Datatype size test:
 - 1: char : signed char : unsigned char
 - 2: short
 - 4: int : long : void* : float
 - 8: long long : double
 - 16: long double

Stack direction test:
- The stack grows downwards.

Stack alignment test:
- Number of least significant 0 bits: 4
- Stack is aligned to 16 bytes.

Argument passing test:
- char : short : int : long : float
 - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
 - args 9   : [stack]
- long long : double
 - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
 - args 5   [low], [high]: [stack]
```

Summary Report

RISC-V 32-bit ABI-Extractor

```
Datatype size test:
 - 1: char : signed char : unsigned char
 - 2: short
 - 4: int : long : void* : float
 - 8: long long : double
 - 16: long double

Stack direction test:
- The stack grows downwards.

Stack alignment test:
- Number of least significant 0 bits: 4
- Stack is aligned to 16 bytes.

Argument passing test:
- char : short : int : long : float
 - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
 - args 9   : [stack]
- long long : double
 - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
 - args 5   [low], [high]: [stack]
```

Summary Report

RISC-V 32-bit ABI-Extractor

```
Datatype size test:
 - 1: char : signed char : unsigned char
 - 2: short
 - 4: int : long : void* : float
 - 8: long long : double
 - 16: long double

Stack direction test:
- The stack grows downwards.

Stack alignment test:
- Number of least significant 0 bits: 4
- Stack is aligned to 16 bytes.

Argument passing test:
- char : short : int : long : float
 - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
 - args 9   : [stack]
- long long : double
 - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
 - args 5   [low], [high]: [stack]
```

Summary Report

RISC-V 32-bit ABI-Extractor

```
Datatype size test:
 - 1: char : signed char : unsigned char
 - 2: short
 - 4: int : long : void* : float
 - 8: long long : double
 - 16: long double

Stack direction test:
- The stack grows downwards.

Stack alignment test:
- Number of least significant 0 bits: 4
- Stack is aligned to 16 bytes.

Argument passing test:
- char : short : int : long : float
 - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
 - args 9   : [stack]
- long long : double
 - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
 - args 5   [low], [high]: [stack]
```

Summary Report

RISC-V 32-bit ABI-Extractor

```
Datatype size test:
 - 1: char : signed char : unsigned char
 - 2: short
 - 4: int : long : void* : float
 - 8: long long : double
 - 16: long double

Stack direction test:
- The stack grows downwards.

Stack alignment test:
- Number of least significant 0 bits: 4
- Stack is aligned to 16 bytes.

Argument passing test:
- char : short : int : long : float
 - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
 - args 9   : [stack]
- long long : double
 - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
 - args 5   [low], [high]: [stack]
```

Summary Report

RISC-V 32-bit ABI-Extractor

```
Datatype size test:
 - 1: char : signed char : unsigned char
 - 2: short
 - 4: int : long : void* : float
 - 8: long long : double
 - 16: long double

Stack direction test:
- The stack grows downwards.

Stack alignment test:
- Number of least significant 0 bits: 4
- Stack is aligned to 16 bytes.

Argument passing test:
- char : short : int : long : float
 - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
 - args 9   : [stack]
- long long : double
 - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
 - args 5   [low], [high]: [stack]
```

```
Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
  - char : short : int : long : float : a0 , a1
  - long long : double [low], [high]: a0, a1
 - empty struct is ignored by C Compiler.

Endianess test:
- Wrote (as ull): 0123456789abcdef
- Read (as char): efcdab8967452301
- This system is little-endian.

Caller/callee-saved test:
- caller-saved: s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11
- callee-saved: t0, t1, t2, a0, a1, a2, a3, a4, a5, a6, a7, t3, t4, t5, t6

Return registers:
- char : short : int : long : float
  - passed in registers: a0
- long long : double
  - passed in registers [low], [high]: a0, a1
```

Summary Report

RISC-V 32-bit ABI-Extractor

```
Datatype size test:
 - 1: char : signed char : unsigned char
 - 2: short
 - 4: int : long : void* : float
 - 8: long long : double
 - 16: long double

Stack direction test:
- The stack grows downwards.

Stack alignment test:
- Number of least significant 0 bits: 4
- Stack is aligned to 16 bytes.

Argument passing test:
- char : short : int : long : float
 - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
 - args 9   : [stack]
- long long : double
 - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
 - args 5   [low], [high]: [stack]
```

```
Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
  - char : short : int : long : float : a0 , a1
  - long long : double [low], [high]: a0, a1
 - empty struct is ignored by C Compiler.

Endianess test:
 - Wrote (as ull): 0123456789abcdef
 - Read (as char): efcdab8967452301
 - This system is little-endian.

Caller/callee-saved test:
 - caller-saved: s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11
 - callee-saved: t0, t1, t2, a0, a1, a2, a3, a4, a5, a6, a7, t3, t4, t5, t6

Return registers:
 - char : short : int : long : float
   - passed in registers: a0
 - long long : double
   - passed in registers [low], [high]: a0, a1
```

Summary Report

RISC-V 32-bit ABI-Extractor

```
Datatype size test:
 - 1: char : signed char : unsigned char
 - 2: short
 - 4: int : long : void* : float
 - 8: long long : double
 - 16: long double

Stack direction test:
- The stack grows downwards.

Stack alignment test:
- Number of least significant 0 bits: 4
- Stack is aligned to 16 bytes.

Argument passing test:
- char : short : int : long : float
 - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
 - args 9   : [stack]
- long long : double
 - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
 - args 5   [low], [high]: [stack]
```

```
Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
  - char : short : int : long : float : a0 , a1
  - long long : double [low], [high]: a0, a1
 - empty struct is ignored by C Compiler.

Endianess test:
 - Wrote (as ull): 0123456789abcdef
 - Read (as char): efcdab8967452301
 - This system is little-endian.

Caller/callee-saved test:
 - caller-saved: s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11
 - callee-saved: t0, t1, t2, a0, a1, a2, a3, a4, a5, a6, a7, t3, t4, t5, t6

Return registers:
 - char : short : int : long : float
   - passed in registers: a0
 - long long : double
   - passed in registers [low], [high]: a0, a1
```

Summary Report

RISC-V 32-bit ABI-Extractor

```
Datatype size test:
 - 1: char : signed char : unsigned char
 - 2: short
 - 4: int : long : void* : float
 - 8: long long : double
 - 16: long double

Stack direction test:
- The stack grows downwards.

Stack alignment test:
- Number of least significant 0 bits: 4
- Stack is aligned to 16 bytes.

Argument passing test:
- char : short : int : long : float
 - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
 - args 9   : [stack]
- long long : double
 - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
 - args 5   [low], [high]: [stack]
```

```
Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
  - char : short : int : long : float : a0 , a1
  - long long : double [low], [high]: a0, a1
 - empty struct is ignored by C Compiler.

Endianess test:
- Wrote (as ull): 0123456789abcdef
- Read (as char): efcdab8967452301
- This system is little-endian.

Caller/callee-saved test:
- caller-saved: s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11
- callee-saved: t0, t1, t2, a0, a1, a2, a3, a4, a5, a6, a7, t3, t4, t5, t6

Return registers:
- char : short : int : long : float
  - passed in registers: a0
- long long : double
  - passed in registers [low], [high]: a0, a1
```

Design Choices

Design choices

- We decided to **not** use DWARF
  - Different compilers might encode information in a different way in DWARF.
  - Only works with compilers that generate DWARF info.
  - Missing or incomplete DWARF data can lead to wrong ABI extraction.
  - Compilers can use their own DWARF extensions.

- We assume availability of:
  - A **compiler** (compiler, assembler and linker).
  - A **simulator**.

Design choices - implementation

- Developed a **Python-based** infrastructure
  - Using generated C test cases that run on a simulator to extract compiler behavior.
  - Produces a summary report on the findings.

- Easy to extend
  - A **Base framework** for generating a C test, running it and extracting the output is provided.
  - Easy to add **new analyzers**.
  - Possibility **to share information** between analyzers.
  - Support for using a **feedback loop**:
    Some test cases are dynamically generated based on results from a previous test run.
    - E.g., finding the boundary where structs are passed in memory instead of through registers.

Agenda

- Introduction
- ABI Extractor
- Implementation details
  - Architecture Dump Information
  - Struct Argument Passing Test Case
  - Example of ambiguous results
  - Extendibility / Portability
  - Future Extensions
- Conclusion
- QA

Architecture Dump Information

Architecture Dump Information

- A **common** way to extract information from the target:
  - Content of all registers
  - Content of the stack
- Used for:
  - **Calling convention**: the location of known **special values** helps the analyzers deduce how the compiler passes arguments / returns values.
  - Identifying **caller/callee** saved registers

```
                                  // Header info
                  Current Stack 0x80002710
Size of pointer (aka register) 0x00000004
      Number of register banks 0x1
                      Bank0: ID 0x0
                    Bank0: Size 0x4
    Bank0: Number of registers 0x20
        Dump of register bank0 // regs_bank0
                   (zero) x0 0x0
                     (ra) x1 0x16e
                     (sp) x2 0x80002710
                     (gp) x3 0x8000080c
                     (tp) x4 0x80000008
                     (t0) x5 0x0
                     (t1) x6 0x20
                         ... (...)
      Stack: 32 entries // Start of stack dump
‘address’ : ‘value’ 0x80002710 : 0x0
                         ... 0x80002714 : 0x0
                             0x80002718 : 0x80002740
                             0x8000271c : 0x3ae0
                             0x80002720 : 0x2f706d74
                             (...)
```

Struct Argument Passing Test Case

Example

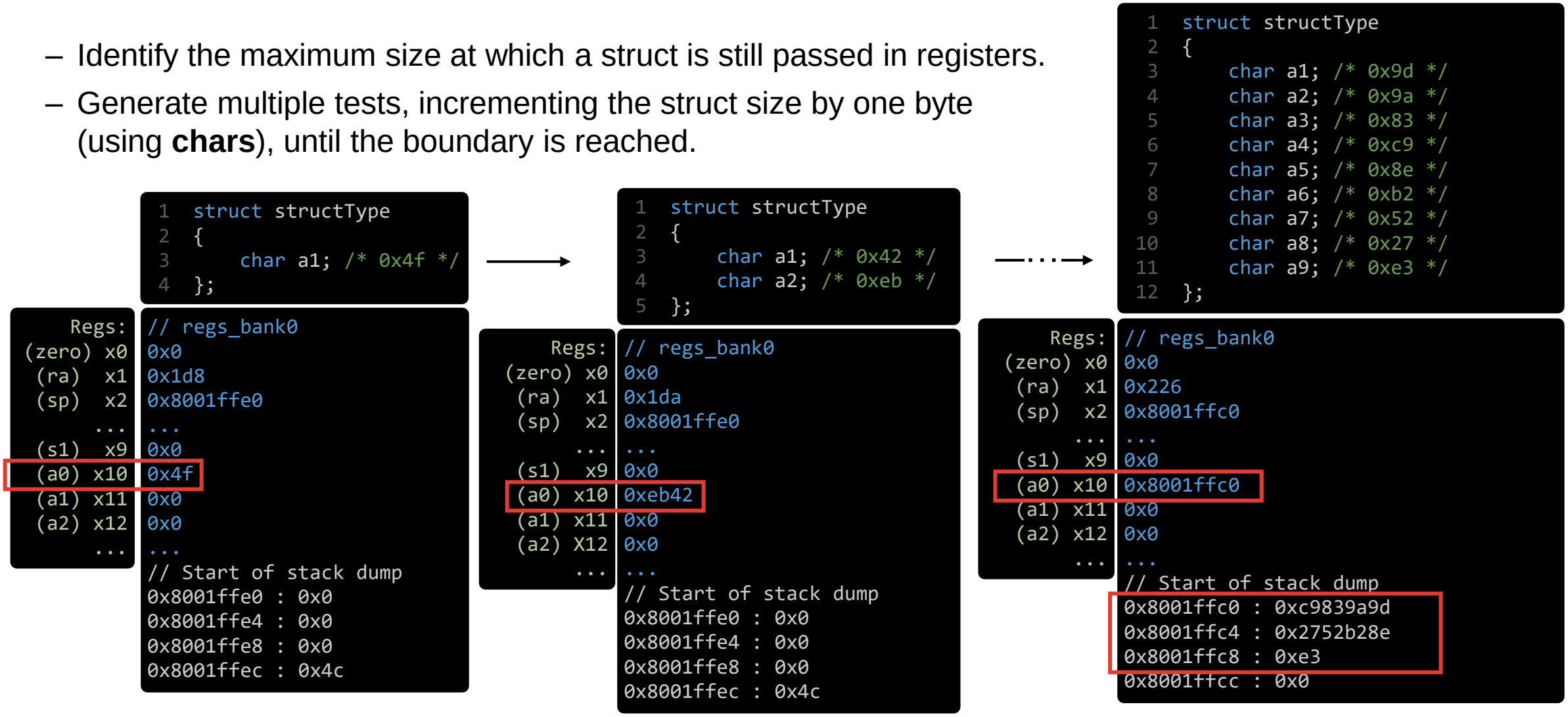
Struct Argument Passing Test Case

- Understand how structs are passed
  - Examine how non-empty and empty structs are passed.
  - Determine which registers are used.
  - Identify under which conditions a struct is passed by reference.
    - ***"Aggregates larger than 2×XLEN bits are passed by reference and are replaced in the argument list with the address,…"***

RISC-V ABIs Specification (Version 1.0): https://drive.google.com/file/d/1Ja_Tpp_5Me583CGVD-BIZMlgGBnlKU4R/view

Struct Argument Passing Test Case

- Identify the maximum size at which a struct is still passed in registers.
- Generate multiple tests, incrementing the struct size by one byte (using **chars**), until the boundary is reached.



=> The boundary is reached at 8 bytes (= two registers) in size.

Struct Argument Passing Test Case

- Size of a struct greater than 8 bytes are expected to be passed by reference.
- For a 32-bit architecture, an **int** is 4 bytes.
  - A struct of 2 **int** (8 bytes) is passed by registers.
  - By adding a char, it is expected to be passed by reference.

```
1  struct structType
2  {
3      int a1;
4      int a2;
5  };
```

```
1  struct structType
2  {
3      int a1;
4      int a2;
5      char a3;
6  };
```

- If the expected boundary is not reached, increase the boundary by one.

```
1  struct structType
2  {
3      int a1;
4      int a2;
5      int a3;
6  };
```

```
1  struct structType
2  {
3      int a1;
4      int a2;
5      int a3;
6      char a4;
7  };
```

Struct Argument Passing Test Case

– Summary of the test case:

```
Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
 - char : short : int : long : float : a0 , a1
 - long long : double [low], [high]: a0, a1
```

– Enabling hardware floating-point leads in the following:

```
Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
   - char : short : int : long : a0, a1
   - long long [low], [high]: a0, a1
   - float : fa0, fa1
- sizeof(S) <= 16 : passed in registers
- sizeof(S) >  16 : passed by ref: [stack]
   - double : fa0, fa1
```

– Why?

Struct Argument Passing Test Case

According to the RISC-V ABI:

– ***"A struct containing two floating-point reals is passed in two floating-point registers, if neither real is more than ABI_FLEN bits wide and at least two floating-registers are available."***

```
1  struct structType
2  {
3      float a1;
4      float a2;
5  };
```

```
1  struct structType
2  {
3      double a1;
4      double a2;
5  };
```

```
1  struct structType
2  {
3      float a1;
4      double a2;
5  };
```

```
1  struct structType
2  {
3      double a1;
4      float a2;
5  };
```

– They will have the same behavior. An increase of the number of members results in the struct being passed by reference.

```
Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
  - char : short : int : long : a0, a1
  - long long [low], [high]: a0, a1
  - float : fa0, fa1
- sizeof(S) <= 16 : passed in registers
- sizeof(S) >  16 : passed by ref: [stack]
  - double : fa0, fa1
- members <= 2 : passed in registers
- members >  2 : passed by ref: [stack]
  - float,float : double,double : float,double : double,float
```

RISC-V ABIs Specification (Version 1.0): https://drive.google.com/file/d/1Ja_Tpp_5Me583CGVD-BIZMlgGBnlKU4R/view

Struct Argument Passing Test Case

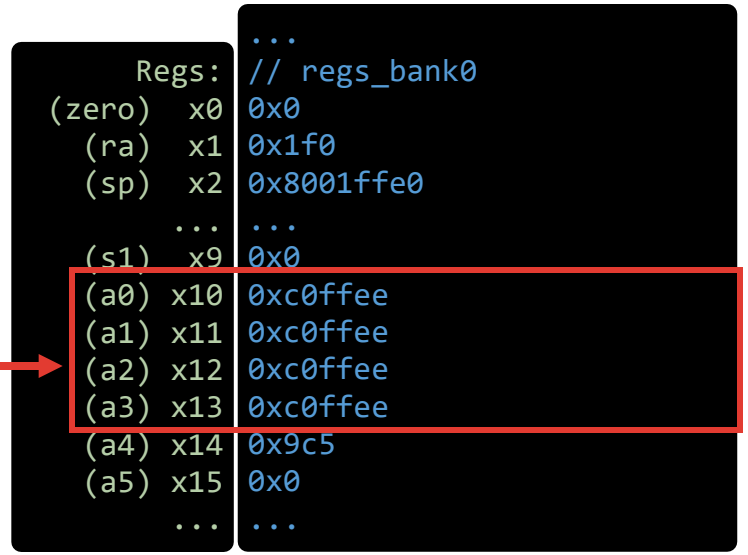
If an empty struct is ignored by C compiler.

– Make use of a **sentinel** value (a marker to indicate the end of the data structure).

– Place the empty struct **between** sentinel values for each possible argument position.

– All registers should contain the sentinel values, **indicating the empty struct was ignored**.

```
 1  struct emptyStruct {};
 2  extern void dump();
 3
 4  int main (void) {
 5      int I = 0xc0ffee;
 6      struct emptyStruct S;
 7      dump(S, I);                          /* 1st arg. */
 8      dump(I, S, I);                       /* 2nd arg. */
 9      dump(I, I, S, I);                    /* 3rd arg. */
10      dump(I, I, I, S, I);                 /* 4th arg. */
11      dump(I, I, I, I, S, I);              /* 5th arg. */
12      dump(I, I, I, I, I, S, I);           /* 6th arg. */
13      dump(I, I, I, I, I, I, S, I);        /* 7th arg. */
14      dump(I, I, I, I, I, I, I, S, I);     /* 8th arg. */
15  };
```



Struct Argument Passing Test Case

Software Floating-Point

```
Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
  - char : short : int : long : float : a0, a1
  - long long : double [low], [high]: a0, a1
- empty struct is ignored by C compiler.
```

Hardware Floating-Point

```
Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
  - char : short : int : long : a0, a1
  - long long [low], [high]: a0, a1
  - float : fa0, fa1
- sizeof(S) <= 16 : passed in registers
- sizeof(S) >  16 : passed by ref: [stack]
  - double : fa0, fa1
- members <= 2 : passed in registers
- members >  2 : passed by ref: [stack]
  - float,float : double,double : float,double : double,float
- empty struct is ignored by C compiler.
```

Example of ambiguous results

Example of ambiguous results

GCC

```
Argument passing test:
- char : short : int : long : float
  - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
  - args 9   : [stack]
- long long : double
  - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
  - args 5   [low], [high]: [stack]

Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
  - char : short : int : long : float : a0, a1
  - long long : double [low], [high]: a0, a1
- empty struct is ignored by C compiler.

 Endianess test:
 - Wrote (as ull):  0123456789abcdef
 - Read  (as char): efcdab8967452301
 - This system is little-endian.

 Caller/callee-saved test:
 - caller-saved s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11
 - callee-saved t0, t1, t2, a0, a1, a2, a3, a4, a5, a6, a7, t3, t4, t5, t6
```

CLANG

```
Argument passing test:
- char : short : int : long : float
  - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
  - args 9   : [stack]
>   - WARNING: multiple value occurrences detected in (t0, [stack])
- long_long : double
  - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
  - args 5   [low], [high]: [stack]
>   - WARNING: multiple value occurrences detected in (t0, [stack])

Struct argument passing test:
- sizeof(S) <= 8 : passed in registers
- sizeof(S) >  8 : passed by ref: [stack]
  - char : short : int : long : float : a0, a1
  - long long : double [low], [high]: a0, a1
- empty struct is ignored by C compiler.

Endianess test:
- Wrote (as ull):  0123456789abcdef
- Read  (as char): efcdab8967452301
- This system is little-endian.

Caller/callee-saved test:
- caller-saved s0, s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11
- callee-saved t0, t1, t2, a0, a1, a2, a3, a4, a5, a6, a7, t3, t4, t5, t6
```

Example of ambiguous results

CLANG

```
  Argument passing test:
  - char : short : int : long : float
    - args 1-8 : a0 a1 a2 a3 a4 a5 a6 a7
    - args 9   : [stack]
>   - WARNING: multiple value occurrences detected in (t0, [stack])
  - long_long : double
    - args 1-4 [low], [high]: [a0, a1] [a2, a3] [a4, a5] [a6, a7]
    - args 5   [low], [high]: [stack]
>   - WARNING: multiple value occurrences detected in (t0, [stack])
```

```c
extern void dump(char, char, char, char, char, char, char, char);

int main(void) {
    dump(0xb5, 0x37, 0x3a, 0x6a, 0xa7, 0xee, 0x28, 0x74, 0xde);
    return 0;
}
```

```
main:
        addi    sp, sp, -16
        sw      ra, 12(sp)
        li      t0, 222
        li      a0, 181
        li      a1, 55
        li      a2, 58
        li      a3, 106
        li      a4, 167
        li      a5, 238
        li      a6, 40
        li      a7, 116
        sw      t0, 0(sp)
        call    dump
        li      a0, 0
        lw      ra, 12(sp)
        addi    sp, sp, 16
        ret
```

https://godbolt.org/z/Ed8afxWPv

Extendibility / Portability

Extendibility

- Bash wrappers
  - Compilation

gcc-riscv32/cc-wrapper

```
#!/bin/bash

# CONFIG_FPU indicates if the compiler supports hardware floating-point.
riscv32-unknown-elf-gcc -march=rv32gc -mabi=ilp32 –DCONFIG_FPU=0 "$@"
```

clang-riscv32/cc-wrapper

```
#!/bin/bash

# CONFIG_FPU indicates if the compiler supports hardware floating-point.
clang -march=rv32gc -mabi=ilp32 –DCONFIG_FPU=0 "$@"
```

```
$ tree scripts/wrapper/
scripts/wrapper/
├── cc
│   ├── clang-riscv32
│   │   ├── as-wrapper
│   │   ├── cc-wrapper
│   │   └── ld-wrapper
│   └── gcc-riscv32
│       ├── as-wrapper
│       ├── cc-wrapper
│       └── ld-wrapper
└── sim
    └── qemu-riscv32
        └── sim-wrapper

```

  - Simulator/Emulator

gcc-riscv32/ld-wrapper

```
#!/bin/bash

qemu-riscv32 –r 5.10 “$@”
```

Portability

- **Target-specific description (Python class)**
  - Specify register banks.

- Target-specific **assembly** file
  - Defines how to save register values into memory.
  - Reset registers to a known value.

```python
# RISCV target specific
class RISCV(TargetArch):
    def __init__(self):
        super().__init__()
        self.Registers = {
            "regs_bank0": [
                "zero", "ra", "sp", "gp", "tp",
                "t0", "t1", "t2", "s0", "s1",
                "a0", "a1", "a2", "a3", "a4",
                "a5", "a6", "a7", "s2", "s3",
                "s4", "s5", "s6", "s7", "s8",
                "s9", "s10", "s11", "t3", "t4",
                "t5", "t6"
            ],
            "regs_bank1": [
                "ft0", "ft1", "ft2", "ft3", "ft4",
                "ft5", "ft6", "ft7", "fs0", "fs1",
                "fa0", "fa1", "fa2", "fa3", "fa4",
                "fa5", "fa6", "fa7", "fs2", "fs3",
                "fs4", "fs5", "fs6", "fs7", "fs8",
                "fs9", "fs10", "fs11", "ft8", "ft9",
                "ft10", "ft11"
            ]
        }
```

Future Extensions

List if potential future extensions

- Support 64-bit
- Long double
- RVV and vector types
  - Handling calling conventions: signedness, size, and alignment.
- Internal function dependencies.
  - Support for `long long` multiplication, float and double simulations.
  - Managing side-effect symbols influencing initialization behavior.
- Thread local storage (TLS)
  - Ensure compatibility with systems like Zephyr.
- ELF header mismatches
  - Resolving discrepancies between header and actual content.
- Relocations
  - Addressing relocations behavior differences across linkers.
- C++ feature support
  - Extra types: `wchar_t`, `bool`, `classes`.
  - Inheritance and v-table layout.
  - Exception Handling.

Agenda

- Introduction
- ABI Extractor
- Implementation details
- Conclusion
  - Conclusion
  - Where to find ABI-Extractor
  - We are hiring
- QA

Conclusion

Conclusion

- ABI-Extractor
  - Is able to extract ABI properties for a compiler…
  - … and dump them in a human readable format.
  - Simplifies the comparison of extracted ABI's.
  - Helps with the validation of the ABI implementation of a compiler.

Where to find ABI-Extractor

- GitHub repository:
  - https://github.com/foss-for-synopsys-dwc-arc-processors/compiler-abi-extractor
  - Licensed under **GPLv3**

  - NOTE: Currently a dummy repository.
    - Subscribe to this issue to get notified as soon as the repository is made public.
      - https://github.com/foss-for-synopsys-dwc-arc-processors/compiler-abi-extractor/issues/1

We Are Hiring !

- https://careers.synopsys.com/
  - Explore our career opportunities and apply.
  - Use "**llvm**" or "**gcc**" as search keywords to find relevant positions.
  - Feel free to reach out if you're interested.

SYNOPSYS®

Thank you

Agenda

- Introduction
- ABI Extractor
- Implementation details
- Conclusion
- QA


SYNOPSYS®