
Tutorial: Creating an LLVM Toolchain for the Cpu0 Architecture

Release 12.0.14

Chen Chung-Shu

Dec 30, 2023

CONTENTS

1	About	3
1.1	Authors	3
1.2	Acknowledgments	3
1.3	Build steps	3
1.4	Revision history	4
1.5	Licensing	5
1.6	Outline of Chapters	5
2	Clang	7
2.1	Cpu0 target	7
2.2	Builtin functions	17
3	Cpu0 ELF linker	19
3.1	ELF to Hex	20
3.2	Create Cpu0 backend under LLD	35
3.3	Summary	59
4	Optimization	61
4.1	LLVM IR optimization	61
4.2	Project	65
5	Library	67
5.1	The theory of Floating point implementation	67
5.2	The dependence for Cpu0 based on Compiler-rt's builtin	69
5.3	C Library (Newlib)	71
5.4	Compiler-rt's builtins	75
6	LLDB	115
7	Appendix A: RISC-V	117
7.1	ISA	118
7.2	Mem	120
7.3	RISC compiler toolchain installation	120
7.4	Linker Command	124
7.5	Qemu simulator	125
7.6	Gem5 simulator	127
7.7	GDB	128
7.8	RISC-V Calling Convention	130
7.9	RVV	131
7.10	Atomic instructions	134
7.11	RISC-V+NPU for Deep Learning	134

7.12	Resources	134
7.13	Websites	135
7.14	To do:	135
8	Appendix B: Clang library and tools	137
8.1	Install llvm with clang tools	137
8.2	Using Clang as a Library	138
8.3	Using Clang as Tools	141
9	Appendix C: POCL	147
9.1	OpenCL	149
9.2	Run OpenCL	149
9.3	Open sources	149
9.4	POCL	151
9.5	Structure	153
9.6	GDB on POCL	153
9.7	Examples of Compiling and running on POCL	154
9.8	RISCV OpenCL	155
9.9	SYCL	156
9.10	DPC++	156
10	Resources	157
10.1	Build steps	157
10.2	Book example code	157
10.3	Alternate formats	157
10.4	Presentation files	157
10.5	Search this website	157
	Bibliography	159



Fig. 1: This book's flow

ABOUT

- *Authors*
- *Acknowledgments*
- *Build steps*
- *Revision history*
- *Licensing*
- *Outline of Chapters*

1.1 Authors

陳鍾樞

Chen Chung-Shu

gamma_chen@yahoo.com.tw

<http://jonathan2251.github.io/web/index.html>

1.2 Acknowledgments

I would like to thank Sean Silva, chisophugis@gmail.com, for his help, encouragement, and assistance with the Sphinx document generator. Without his help, this book would not have been finished and published online. Also thanking those corrections from readers who make the book more accurate.

1.3 Build steps

<https://github.com/Jonathan2251/lbt/blob/master/README.md>

1.4 Revision history

Version 12.0.15, not released yet.

Version 12.0.14, Released December 30, 2023.

libclc. opt.rst: llvm-link.gv. Add caption for reference of .gv. lib.rst: compiler-rt-dep.gv, compiler-rt-dep-short.gv. lib.rst: 16-bit a*b calculation.

Version 12.0.13, Released July 15, 2023.

riscv.rst: AnLLVMBasedNPUCompiler.key.

Version 12.0.12, Released April 4, 2023.

riscv.rst: The advantage of RISCv little endians.

Version 12.0.11, Released February 27, 2023.

riscv.rst: section Linker Command. pocl.rst, riscv.rst: books and document. some example-code on exlbt/riscv/. loop-convert2, Rewriter and PrintFunction in exlbt/clang-tools.

Version 12.0.10, Release November 13, 2022.

clang.rst:section Builtin functions. lib.rst:section Compiler-rt's builtins. pocl.rst. riscv.rst:table:: RISCv toolchain, sections Mem and Resources.

Version 12.0.9, Release September 24, 2022.

GDB section and refine Makefile bash

Version 12.0.8, Release September 8, 2022.

Appendix A: add sections ISA and RISCv Calling Convention. Add Appendix B: Clang-tools.

Version 12.0.7, Release August 16, 2022.

Appendix A: RISCv

Version 12.0.6, Released February 3, 2022.

scanf and printf.

Version 12.0.5, Released February 1, 2022.

Newlib/libm and complex type test for builtins-test.

Version 12.0.4, Released January 22, 2022.

sanitizer-printf for supporting printf("%lld") or "%llX", ..., etc. Pass test cases in compiler-rt-test/builtins/Unit include type float and double exclude complex.

Version 12.0.3, Released January 9, 2022.

Expand memory size of cpu0.v to 0x1000000, 24-bit. Add all compiler-rt-test/builtins/Unit/*.c.

Version 12.0.2, Release December 18, 2021.

Replace bash with Makefile. Add builtins-cpu0.c for clang regression test.

Version 12.0.1, Release December 12, 2021.

Add target Cpu0 to clang

Version 12.0.0, Release August 11, 2021.

Version 3.9.1, Released April 29, 2020

Enable tailcall test option in build-slinker.sh

Version 3.9.0, Released November 22, 2016

Porting to llvm 3.9.

Version 3.7.4, Released September 22, 2016

Split elf2hex-dlinker.cpp from elf2hex.cpp in exlbt/elf2hex.

Version 3.7.3, Released July 20, 2016

Refine code-block according sphinx lexers. Add search this book.

Version 3.7.2, Released June 29, 2016

Dynamic linker change display from ret \$t9 to jr \$t9. Move llvm-objdump -elf2hex to elf2hex. Upgrade sphinx to 1.4.4.

Version 3.7.1, Released November 7, 2015

Remove EM_CPU0_EL. Add IR blockaddress and indirectbr support. Add ch_9_3_detect_exception.cpp test. Change display “ret \$rx” to “jr \$rx” where \$rx is not \$lr. Add Phi node test.

Version 3.7.0, Released September 24, 2015

Porting to lld 3.7.

Version 3.6.2, Released May 4, 2015

Move some test from lbt to lbd. Remove warning in build Cpu0 code.

Version 3.6.1, Released March 22, 2015

Correct typing.

Version 3.6.0, Released March 8, 2015

Porting to lld 3.6.

1.5 Licensing

<http://llvm.org/docs/DeveloperPolicy.html#license>

1.6 Outline of Chapters

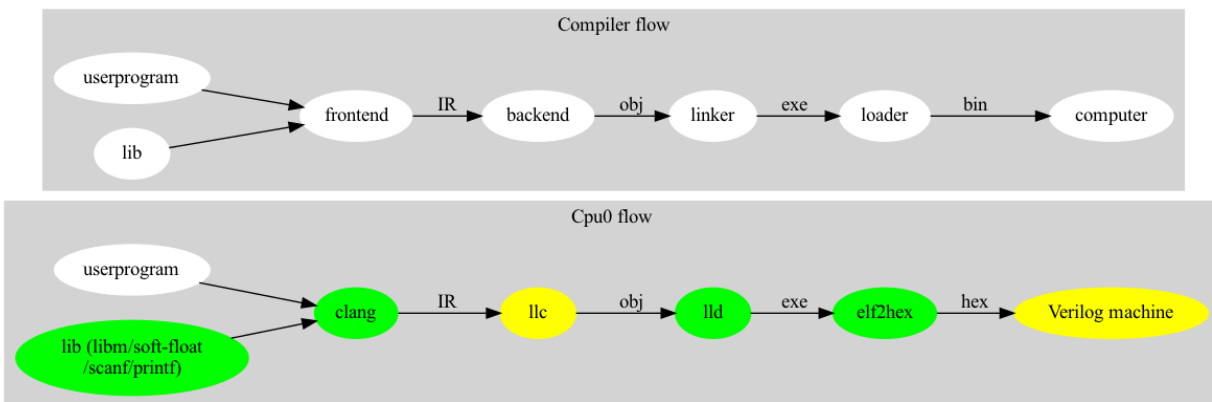


Fig. 1.1: Code generation and execution flow

The upper half of Fig. 1.1 is the work flow and software package of a computer program be generated and executed. IR stands for Intermediate Representation. The lower half is this book's work flow and software package of the toolchain extended implementation based on llvm. Except clang, the other blocks need to be extended for a new backend development. This book implement the green boxes part. The Cpu0 llvm backend can be find on <http://jonathan2251.github.io/lbd/index.html>.

Table 1.1: Toolchain components^{2,3}

Component	LLVM	GNU ¹
C/C++ Compiler	clang/llvm	gcc
Assembler	llvm integrated assembler	as
Linker	ld.lld	ld.bfd ld.gold
Runtime	compiler-rt	libgcc ⁴
Unwinder	libunwind	libgcc_s
C++ library	libc++abi, libc++	libsupc++ libstdc++
Utils	llvm-ar, llvm-objdump etc.	ar, objdump etc.
C library	•	libc

The libgcc's Integer plus Soft float library^{4,5,6} are equal to functions of compiler-rt's builtins.

This book include:

1. Add Cpu0 target to clang.
2. The elf2hex extended from llvm-objump. Chapter 3.
3. Optimization. Chapter 4.
4. Porting C standard library from avr libc and software floating point library from LLVM compiler-rt.

With these implementation, reader can generate Cpu0 machine code through Cpu0 llvm backend compiler, linker and elf2hex, then see how it runs on your computer.

Clang:

Add Cpu0 target to clang.

Cpu0 ELF linker:

Develop ELF linker for Cpu0 backend based on lld project.

Optimization:

Backend independent optimaization.

Library:

Software floating point library and standard C library supporting.

² page 8 - 9 of https://archive.fosdem.org/2018/schedule/event/crosscompile/attachments/slides/2107/export/events/attachments/crosscompile/slides/2107/How_to_cross_compile_with_LLVM_based_tools.pdf

³ <https://bcain-llvm.readthedocs.io/projects/clang/en/latest/Toolchain/#compiler-runtime>

¹ https://en.wikipedia.org/wiki/GNU_Compiler_Collection#cite_note-55

⁴ <https://gcc.gnu.org/onlinedocs/gccint/Libgcc.html>

⁵ <https://gcc.gnu.org/onlinedocs/gccint/Integer-library-routines.html#Integer-library-routines>

⁶ <https://gcc.gnu.org/onlinedocs/gccint/Soft-float-library-routines.html#Soft-float-library-routines>

- *Cpu0 target*
- *Builtin functions*

This chapter add Cpu0 target to frontend clang.

2.1 Cpu0 target

exlbt/clang/include/clang/lib/Driver/CMakeLists.txt

ToolChains/Arch/Cpu0.cpp

exlbt/clang/lib/Driver/ToolChains/CommonArgs.cpp

```
#include "Arch/Cpu0.h"
...
case llvm::Triple::cpu0:
case llvm::Triple::cpu0el: {
   StringRef CPUName;
   StringRef ABIName;
    cpu0::getCpu0CPUAndABI(Args, T, CPUName, ABIName);
    return std::string(CPUName);
}
```

exlbt/clang/lib/Driver/ToolChains/Arch/Cpu0.h

```
//====- Cpu0.h - Cpu0-specific Tool Helpers -----*- C++ -*-====//
//
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
// See https://llvm.org/LICENSE.txt for license information.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
//
//=====
```

(continues on next page)

(continued from previous page)

```

#ifndef LLVM_CLANG_LIB_DRIVER_TOOLCHAINS_ARCH_CPU0_H
#define LLVM_CLANG_LIB_DRIVER_TOOLCHAINS_ARCH_CPU0_H

#include "clang/Driver/Driver.h"
#include "llvm/ADT/StringRef.h"
#include "llvm/ADT/Triple.h"
#include "llvm/Option/Option.h"
#include <string>
#include <vector>

namespace clang {
namespace driver {
namespace tools {

namespace cpu0 {

void getCpu0CPUAndABI(const llvm::opt::ArgList &Args,
                     const llvm::Triple &Triple, StringRef &CPUName,
                     StringRef &ABIName);

} // end namespace cpu0
} // end namespace target
} // end namespace driver
} // end namespace clang

#endif // LLVM_CLANG_LIB_DRIVER_TOOLCHAINS_ARCH_CPU0_H

```

exlbt/clang/lib/Driver/ToolChains/Arch/Cpu0.cpp

```

//===-- Cpu0.cpp - Tools Implementations -----*- C++ -*-===//
//
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
// See https://llvm.org/LICENSE.txt for license information.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
//
//===-----

```

```

#include "Cpu0.h"
#include "ToolChains/CommonArgs.h"
#include "clang/Driver/Driver.h"
#include "clang/Driver/DriverDiagnostic.h"
#include "clang/Driver/Options.h"
#include "llvm/ADT/StringSwitch.h"
#include "llvm/Option/ArgList.h"

using namespace clang::driver;
using namespace clang::driver::tools;
using namespace clang;
using namespace llvm::opt;

```

(continues on next page)

(continued from previous page)

```
// Get CPU and ABI names. They are not independent
// so we have to calculate them together.
void cpu0::getCpu0CPUAndABI(const ArgList &Args, const llvm::Triple &Triple,
                          StringRef &CPUName, StringRef &ABIName) {
    if (Arg *A = Args.getLastArg(clang::driver::options::OPT_march_EQ,
                                options::OPT_mcpu_EQ))
        CPUName = A->getValue();

    if (Arg *A = Args.getLastArg(options::OPT_mabi_EQ)) {
        ABIName = A->getValue();
        // Convert a GNU style Cpu0 ABI name to the name
        // accepted by LLVM Cpu0 backend.
        ABIName = llvm::StringSwitch<llvm::StringRef>(ABIName)
            .Case("32", "o32")
            .Default(ABIName);
    }

    // Setup default CPU and ABI names.
    if (CPUName.empty()) {
        switch (Triple.getArch()) {
        default:
            llvm_unreachable("Unexpected triple arch name");
        case llvm::Triple::cpu0:
        case llvm::Triple::cpu0el:
            CPUName = "cpu032II";
            break;
        }
    }

    if (ABIName.empty())
        ABIName = "o32";
}
```

exlbt/clang/include/clang/lib/Basic/CMakeLists.txt

Targets/Cpu0.cpp

exlbt/clang/include/clang/lib/Basic/Targets.cpp

```
#include "Targets/Cpu0.h"
...
case llvm::Triple::cpu0:
    switch (os) {
    case llvm::Triple::Linux:
        return new LinuxTargetInfo<Cpu0TargetInfo>(Triple, Opts);
    case llvm::Triple::RTEMS:
        return new RTEMSTargetInfo<Cpu0TargetInfo>(Triple, Opts);
    case llvm::Triple::FreeBSD:
        return new FreeBSDTargetInfo<Cpu0TargetInfo>(Triple, Opts);
    }
```

(continues on next page)

(continued from previous page)

```

case llvm::Triple::NetBSD:
    return new NetBSDTargetInfo<Cpu0TargetInfo>(Triple, Opts);
default:
    return new Cpu0TargetInfo(Triple, Opts);
}

case llvm::Triple::cpu0el:
    switch (os) {
    case llvm::Triple::Linux:
        return new LinuxTargetInfo<Cpu0TargetInfo>(Triple, Opts);
    case llvm::Triple::RTEMS:
        return new RTEMSTargetInfo<Cpu0TargetInfo>(Triple, Opts);
    case llvm::Triple::FreeBSD:
        return new FreeBSDTargetInfo<Cpu0TargetInfo>(Triple, Opts);
    case llvm::Triple::NetBSD:
        return new NetBSDTargetInfo<Cpu0TargetInfo>(Triple, Opts);
    default:
        return new Cpu0TargetInfo(Triple, Opts);
    }
}

```

exlbt/clang/lib/Basic/Targets/Cpu0.h

```

//====-- Cpu0.h - Declare Cpu0 target feature support -----*- C++ -*-====//
//
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
// See https://llvm.org/LICENSE.txt for license information.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
//
//====-
//
// This file declares Cpu0 TargetInfo objects.
//
//====-

#ifndef LLVM_CLANG_LIB_BASIC_TARGETS_CPU0_H
#define LLVM_CLANG_LIB_BASIC_TARGETS_CPU0_H

#include "clang/Basic/TargetInfo.h"
#include "clang/Basic/TargetOptions.h"
#include "llvm/ADT/Triple.h"
#include "llvm/Support/Compiler.h"

namespace clang {
namespace targets {

class LLVM_LIBRARY_VISIBILITY Cpu0TargetInfo : public TargetInfo {
    void setDataLayout() {
       StringRef Layout;

        if (ABI == "o32")

```

(continues on next page)

(continued from previous page)

```

    Layout = "m:m-p:32:32-i8:8:32-i16:16:32-i64:64-n32-S64";
else if (ABI == "n32")
    Layout = "m:e-p:32:32-i8:8:32-i16:16:32-i64:64-n32:64-S128";
else if (ABI == "n64")
    Layout = "m:e-i8:8:32-i16:16:32-i64:64-n32:64-S128";
else
    llvm_unreachable("Invalid ABI");

if (BigEndian)
    resetDataLayout(("E-" + Layout).str());
else
    resetDataLayout(("e-" + Layout).str());
}

static const Builtin::Info BuiltinInfo[];
std::string CPU;

protected:
    std::string ABI;
    enum Cpu0FloatABI { HardFloat, SoftFloat } FloatABI;

public:
    Cpu0TargetInfo(const llvm::Triple &Triple, const TargetOptions &Opt)
        : TargetInfo(Triple) {
        TheCXXABI.set(TargetCXXABI::GenericMIPS); // Cpu0 uses Mips ABI

        setABI("o32");

        CPU = "cpu032II";
    }

    StringRef getABI() const override { return ABI; }

    bool setABI(const std::string &Name) override {
        if (Name == "o32") {
            ABI = Name;
            return true;
        }
        return false;
    }

    bool isValidCPUName(StringRef Name) const override;

    bool setCPU(const std::string &Name) override {
        CPU = Name;
        return isValidCPUName(Name);
    }

    const std::string &getCPU() const { return CPU; }
    bool
    initFeatureMap(llvm::StringMap<bool> &Features, DiagnosticsEngine &Diags,
                  StringRef CPU,

```

(continues on next page)

(continued from previous page)

```

        const std::vector<std::string> &FeaturesVec) const override {
    if (CPU.empty())
        CPU = getCPU();
    if (CPU == "cpu032II")
        Features["HasCmp"] = Features["HasSlt"] = true;
    else if (CPU == "cpu032I")
        Features["HasCmp"] = true;
    else
        assert(0 && "incorrect CPU");
    return TargetInfo::initFeatureMap(Features, Diags, CPU, FeaturesVec);
}

unsigned getISARev() const;

void getTargetDefines(const LangOptions &Opts,
                     MacroBuilder &Builder) const override;

ArrayRef<Builtin::Info> getTargetBuiltins() const override;

bool hasFeature(StringRef Feature) const override;

BuiltinVaListKind getBuiltinVaListKind() const override {
    return TargetInfo::VoidPtrBuiltinVaList;
}

ArrayRef<const char *> getGCCRegNames() const override {
    static const char *const GCCRegNames[] = {
        // CPU register names
        // Must match second column of GCCRegAliases
        "$0", "$1", "$2", "$3", "$4", "$5", "$6", "$7", "$8", "$9", "$10",
        "$11", "$12", "$13", "$14", "$15",
        // Hi/lo and condition register names
        "hi", "lo",
    };
    return llvm::makeArrayRef(GCCRegNames);
}

bool validateAsmConstraint(const char *&Name,
                          TargetInfo::ConstraintInfo &Info) const override {
    switch (*Name) {
    default:
        return false;
    case 'r': // CPU registers.
    case 'd': // Equivalent to "r" unless generating MIPS16 code.
    case 'y': // Equivalent to "r", backward compatibility only.
    //case 'f': // floating-point registers.
    case 'c': // $6 for indirect jumps
    case 'l': // lo register
    case 'x': // hilo register pair
        Info.setAllowsRegister();
        return true;
    case 'I': // Signed 16-bit constant

```

(continues on next page)

(continued from previous page)

```

case 'J': // Integer 0
case 'K': // Unsigned 16-bit constant
case 'L': // Signed 32-bit constant, lower 16-bit zeros (for lui)
case 'M': // Constants not loadable via lui, addiu, or ori
case 'N': // Constant -1 to -65535
case 'O': // A signed 15-bit constant
case 'P': // A constant between 1 go 65535
    return true;
case 'R': // An address that can be used in a non-macro load or store
    Info.setAllowsMemory();
    return true;
case 'Z':
    if (Name[1] == 'C') { // An address usable by ll, and sc.
        Info.setAllowsMemory();
        Name++; // Skip over 'Z'.
        return true;
    }
    return false;
}
}

const char *getClobbers() const override {
    // In GCC, $1 is not widely used in generated code (it's used only in a few
    // specific situations), so there is no real need for users to add it to
    // the clobbers list if they want to use it in their inline assembly code.
    //
    // In LLVM, $1 is treated as a normal GPR and is always allocatable during
    // code generation, so using it in inline assembly without adding it to the
    // clobbers list can cause conflicts between the inline assembly code and
    // the surrounding generated code.
    //
    // Another problem is that LLVM is allowed to choose $1 for inline assembly
    // operands, which will conflict with the ".set at" assembler option (which
    // we use only for inline assembly, in order to maintain compatibility with
    // GCC) and will also conflict with the user's usage of $1.
    //
    // The easiest way to avoid these conflicts and keep $1 as an allocatable
    // register for generated code is to automatically clobber $1 for all inline
    // assembly code.
    //
    // FIXME: We should automatically clobber $1 only for inline assembly code
    // which actually uses it. This would allow LLVM to use $1 for inline
    // assembly operands if the user's assembly code doesn't use it.
    return "~{$1}";
}

bool handleTargetFeatures(std::vector<std::string> &Features,
                        DiagnosticsEngine &Diags) override {
    FloatABI = SoftFloat;

    for (const auto &Feature : Features) {

```

(continues on next page)

(continued from previous page)

```

    if (Feature == "+cpu032I")
        setCPU("cpu032I");
    else if (Feature == "+cpu032II")
        setCPU("cpu032II");
    else if (Feature == "+soft-float")
        FloatABI = SoftFloat;
}

setDataLayout();

return true;
}

ArrayRef<TargetInfo::GCCRegAlias> getGCCRegAliases() const override {
    static const TargetInfo::GCCRegAlias RegAliases[] = {
        {"at"}, "$1", {"v0"}, "$2", {"v1"}, "$3",
        {"a0"}, "$4", {"a1"}, "$5", {"t9"}, "$6",
        {"gp"}, "$11", {"fp"}, "$12", {"sp"}, "$13",
        {"lr"}, "$14", {"sw"}, "$15"
    };
    return llvm::makeArrayRef(RegAliases);
}

bool hasInt128Type() const override {
    return false;
}

unsigned getUnwindWordWidth() const override;

bool validateTarget(DiagnosticsEngine &Diags) const override;
bool hasExtIntType() const override { return true; }
};
} // namespace targets
} // namespace clang

#endif // LLVM_CLANG_LIB_BASIC_TARGETS_Cpu0_H

```

exlbt/clang/lib/Basic/Targets/Cpu0.cpp

```

//====-- Cpu0.cpp - Implement Cpu0 target feature support -----===//
//
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
// See https://llvm.org/LICENSE.txt for license information.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
//
//====-----===//
//
// This file implements Cpu0 TargetInfo objects.
//
//====-----===//

```

(continues on next page)

(continued from previous page)

```

#include "Cpu0.h"
#include "Targets.h"
#include "clang/Basic/Diagnostic.h"
#include "clang/Basic/MacroBuilder.h"
#include "clang/Basic/TargetBuiltins.h"
#include "llvm/ADT/StringSwitch.h"

using namespace clang;
using namespace clang::targets;

const Builtin::Info Cpu0TargetInfo::BuiltinInfo[] = {
#define BUILTIN(ID, TYPE, ATTRS) \
    {#ID, TYPE, ATTRS, nullptr, ALL_LANGUAGES, nullptr}, \
#define LIBBUILTIN(ID, TYPE, ATTRS, HEADER) \
    {#ID, TYPE, ATTRS, HEADER, ALL_LANGUAGES, nullptr}, \
#include "clang/Basic/BuiltinsCpu0.def"
};

static constexpr llvm::StringLiteral ValidCPUNames[] = {
    {"cpu032I"}, {"cpu032II"};

bool Cpu0TargetInfo::isValidCPUName(StringRef Name) const {
    return llvm::find(ValidCPUNames, Name) != std::end(ValidCPUNames);
}

unsigned Cpu0TargetInfo::getISARev() const {
    return llvm::StringSwitch<unsigned>(getCPU())
        .Case("cpu032I", 1)
        .Case("cpu032II", 2)
        .Default(0);
}

void Cpu0TargetInfo::getTargetDefines(const LangOptions &Opts,
                                      MacroBuilder &Builder) const {
    if (BigEndian) {
        DefineStd(Builder, "CPU0EB", Opts);
        Builder.defineMacro("_CPU0EB");
    } else {
        DefineStd(Builder, "CPU0EL", Opts);
        Builder.defineMacro("_CPU0EL");
    }

    Builder.defineMacro("__cpu0__");
    Builder.defineMacro("_cpu0");
    if (Opts.GNUMode)
        Builder.defineMacro("cpu0");

    if (ABI == "o32" || ABI == "s32") {
        Builder.defineMacro("__cpu0", "32");
        Builder.defineMacro("_CPU0_ISA", "_CPU0_ISA_CPU032");
    } else {

```

(continues on next page)

(continued from previous page)

```

    llvm_unreachable("Invalid ABI.");
}

const std::string ISARev = std::to_string(getISARev());

if (!ISARev.empty())
    Builder.defineMacro("__cpu0_isa_rev", ISARev);

if (ABI == "o32") {
    Builder.defineMacro("__cpu0_o32");
    Builder.defineMacro("_ABI032", "1");
    Builder.defineMacro("_CPU0_SIM", "_ABI032");
} else if (ABI == "s32") {
    Builder.defineMacro("__cpu0_n32");
    Builder.defineMacro("_ABIS32", "2");
    Builder.defineMacro("_CPU0_SIM", "_ABIN32");
} else
    llvm_unreachable("Invalid ABI.");

Builder.defineMacro("__REGISTER_PREFIX__", "");

switch (FloatABI) {
case HardFloat:
    llvm_unreachable("HardFloat is not support in Cpu0");
    break;
case SoftFloat:
    Builder.defineMacro("__cpu0_soft_float", Twine(1));
    break;
}
}

bool Cpu0TargetInfo::hasFeature(StringRef Feature) const {
    return llvm::StringSwitch<bool>(Feature)
        .Case("cpu0", true)
        .Default(false);
}

ArrayRef<Builtin::Info> Cpu0TargetInfo::getTargetBuiltins() const {
    return llvm::makeArrayRef(BuiltinInfo, clang::Cpu0::LastTSBuiltin -
                               Builtin::FirstTSBuiltin);
}

unsigned Cpu0TargetInfo::getUnwindWordWidth() const {
    return llvm::StringSwitch<unsigned>(ABI)
        .Cases("o32", "s32", 32)
        .Default(getPointerWidth(0));
}

bool Cpu0TargetInfo::validateTarget(DiagnosticsEngine &Diags) const {
    if (CPU != "cpu032I" && CPU != "cpu032II") {
        Diags.Report(diag::err_target_unknown_cpu) << ABI << CPU;
        return false;
    }
}

```

(continues on next page)

(continued from previous page)

```

}

return true;
}

```

```

chungshu@ChungShudeMacBook-Air input % ~/llvm/test/build/bin/clang --help-hidden|grep_
↪cpu0
-cpu032II          Equivalent to -march=cpu032II
-cpu032I           Equivalent to -march=cpu032I

```

2.2 Builtin functions

Builtin-function is a function may map to one or more HW instructions for speedup. Cpu0 port compiler-rt's builtins in the section Compiler-rt's builtins of chapter Library¹. Built-in kernels are kernels that are specific to a particular device and provide a mechanism to allow an application developer to leverage special hardware that may be present on the device².

exlbt/clang/include/clang/Basic/TargetBuiltins.h

```

/// CPU0 builtins
namespace Cpu0 {
    enum {
        LastTIBuiltin = clang::Builtin::FirstTSBuiltin-1,
#define BUILTIN(ID, TYPE, ATTRS) BI##ID,
#include "clang/Basic/BuiltinsCpu0.def"
        LastTSBuiltin
    };
}

```

exlbt/clang/include/clang/Basic/BuiltinsCpu0.def

```

//===-- BuiltinsCpu0.def - Cpu0 Builtin function database -----*- C++ -*-===//
//
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
// See https://llvm.org/LICENSE.txt for license information.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
//
//===-----
//
// This file defines the CPU0-specific builtin function database. Users of
// this file must define the BUILTIN macro to make use of this information.
//
//===-----

// The format of this database matches clang/Basic/Builtins.def.

```

(continues on next page)

¹ <http://jonathan2251.github.io/lbt/lib.html#compiler-rt-s-builtins>

² Chapter 5.4 of book "Heterogeneous computing with OpenCL 2.0, 3rd edition"

(continued from previous page)

```
// Reference:
// https://github.com/Jonathan2251/lbd/blob/master/lbdex/llvm/modify/llvm/include/llvm/
// IR/IntrinsicsCpu0.td
BUILTIN(__builtin_cpu0_gcd, "iii", "n")

#undef BUILTIN
```

```
chungshu@ChungShudeMacBook-Air CodeGen % pwd
/Users/chungshu/llvm/test/clang/test/CodeGen
chungshu@ChungShudeMacBook-Air CodeGen % ~/llvm/test/build/bin/llvm-lit builtins-cpu0.c
..
-- Testing: 1 tests, 1 workers --
    PASS: Clang :: CodeGen/builtins-cpu0.c (1 of 1)

Testing Time: 0.14s
    Passed: 1
```

CPU0 ELF LINKER

- *ELF to Hex*
- *Create Cpu0 backend under LLD*
 - *LLD introduction*
 - *Static linker*
 - *Dynamic linker*
- *Summary*
 - *Create a new backend base on LLVM*
 - *Contribute back to Open Source through working and learning*

LLD changes quickly and the figures of this chapter is not up to date. Like llvm, lld linker include a couple of target in ELF format handling. The term Cpu0 backend used in this chapter can refer to the ELF format handling for Cpu0 target machine under lld, llvm compiler backend, or both. But supposing readers will easy knowing what it refer to.

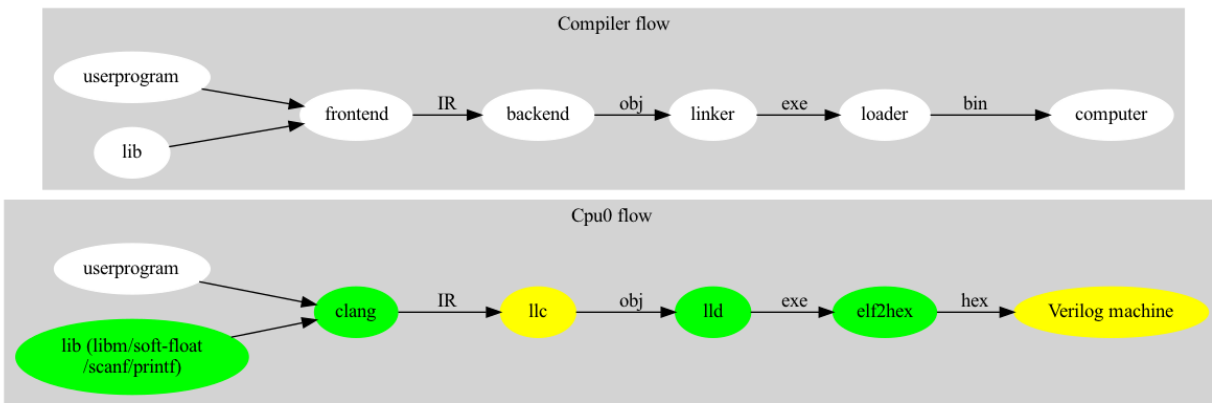


Fig. 3.1: Code generation and execution flow

As depicted in Fig. 3.1 of chapter About. Beside llvm backend, we implement ELF linker and elf2hex to run on Cpu0 verilog simulator. This chapter extends lld to support Cpu0 backend as well as elf2hex to replace Cpu0 loader. After link with lld, the program with global variables can be allocated in ELF file format layout. Meaning the relocation records of global variables is resolved. In addition, elf2hex is implemented for supporting generate Hex file from ELF. With these two tools supported, the global variables exists in section .data and .rodata can be accessed and transfered to Hex file which feeds to Verilog Cpu0 machine and run on your PC/Laptop.

As the previous chapters mentioned, Cpu0 has two relocation models for static link and dynamic link, respectively, which controlled by option `-relocation-model` in `llc`. This chapter supports the static link.

About `lld` please refer LLD web site here¹ and LLD install requirement on Linux here². Currently, `lld` can be built by: `gcc` and `clang` compiler on Ubuntu. On iMac, `lld` can be built by `clang` with the Xcode version as the next sub section. If you run with Virtual Machine (VM), please keep your physical memory size setting over 1GB to avoid insufficient memory link error.

3.1 ELF to Hex

As follows,

`exlbt/elf2hex/CMakeLists.txt`

```
# elf2hex.cpp needs backend related functions, like
# LLVMInitializeCpu0TargetInfo and LLVMInitializeCpu0Disassembler ... etc.
# Set LLVM_LINK_COMPONENTS then it can link them during the link stage.
set(LLVM_LINK_COMPONENTS
#   AllTargetsAsmPrinters
  AllTargetsDescs
  AllTargetsDisassemblers
  AllTargetsInfos
  BinaryFormat
  CodeGen
  DebugInfoDWARF
  DebugInfoPDB
  Demangle
  MC
  MCDisassembler
  Object
  Support
  Symbolize
)

add_llvm_tool(elf2hex
  elf2hex.cpp
)

if(HAVE_LIBXAR)
  target_link_libraries(elf2hex PRIVATE ${XAR_LIB})
endif()

if(LLVM_INSTALL_BINUTILS_SYMLINKS)
  add_llvm_tool_symlink(elf2hex elf2hex)
endif()
```

¹ <http://lld.llvm.org/>

² http://lld.llvm.org/getting_started.html#on-unix-like-systems

exlbt/elf2hex/elf2hex.h

```

//
//                               The LLVM Compiler Infrastructure
//
// This file is distributed under the University of Illinois Open Source
// License. See LICENSE.TXT for details.
//
//====-----

#ifndef LLVM_TOOLS_ELF2HEX_ELF2HEX_H
#define LLVM_TOOLS_ELF2HEX_ELF2HEX_H

#include "llvm/DebugInfo/DIContext.h"
#include "llvm/MC/MCDisassembler/MCDisassembler.h"
#include "llvm/MC/MCInstPrinter.h"
#include "llvm/Support/CommandLine.h"
#include "llvm/Support/Compiler.h"
#include "llvm/Support/DataTypes.h"
#include "llvm/Object/Archive.h"

#include <stdio.h>
#include "llvm/Support/raw_ostream.h"

#define BOOT_SIZE 16

#define DLINK
// #define ELF2HEX_DEBUG

namespace llvm {
namespace elf2hex {

using namespace object;

class HexOut {
public:
    virtual void ProcessDisAsmInstruction(MCInst inst, uint64_t Size,
                                         ArrayRef<uint8_t> Bytes, const ObjectFile *Obj) = 0;
    virtual void ProcessDataSection(SectionRef Section) {};
    virtual ~HexOut() {};
};

// Split HexOut from Reader::DisassembleObject() for separating hex output
// functions.
class VerilogHex : public HexOut {
public:
    VerilogHex(std::unique_ptr<MCInstPrinter>& instructionPrinter,
               std::unique_ptr<const MCSubtargetInfo>& subTargetInfo,
               const ObjectFile *Obj);
    void ProcessDisAsmInstruction(MCInst inst, uint64_t Size,
                                  ArrayRef<uint8_t> Bytes, const ObjectFile *Obj) override;
    void ProcessDataSection(SectionRef Section) override;

```

(continues on next page)

(continued from previous page)

```

private:
    void PrintBootSection(uint64_t textOffset, uint64_t isrAddr, bool isLittleEndian);
    void Fill0s(uint64_t startAddr, uint64_t endAddr);
    void PrintDataSection(SectionRef Section);
    std::unique_ptr<MCInstPrinter>& IP;
    std::unique_ptr<const MCSubtargetInfo>& STI;
    uint64_t lastDumpAddr;
    unsigned si;
    StringRef sectionName;
};

class Reader {
public:
    void DisassembleObject(const ObjectFile *Obj,
                          std::unique_ptr<MCDisassembler>& DisAsm,
                          std::unique_ptr<MCInstPrinter>& IP,
                          std::unique_ptr<const MCSubtargetInfo>& STI);
    StringRef CurrentSymbol();
    SectionRef CurrentSection();
    unsigned CurrentSi();
    uint64_t CurrentIndex();

private:
    SectionRef _section;
    std::vector<std::pair<uint64_t, StringRef> > Symbols;
    unsigned si;
    uint64_t Index;
};

} // end namespace elf2hex
} // end namespace llvm

//using namespace llvm;

#endif

```

exlbt/elf2hex/elf2hex.cpp

```

//====-- llvm-objdump.cpp - Object file dumping utility for llvm -----===//
//
//                               The LLVM Compiler Infrastructure
//
// This file is distributed under the University of Illinois Open Source
// License. See LICENSE.TXT for details.
//
//====-----
//
// This program is a utility that works like binutils "objdump", that is, it
// dumps out a plethora of information about an object file depending on the

```

(continues on next page)

(continued from previous page)

```

// flags.
//
// The flags and output of this program should be near identical to those of
// binutils objdump.
//
//===-----
#define ELF2HEX

#include "elf2hex.h"
#include "llvm/MC/MCAsmInfo.h"
#include "llvm/MC/MCContext.h"
#include "llvm/MC/MCInst.h"
#include "llvm/MC/MCInstrAnalysis.h"
#include "llvm/MC/MCInstrInfo.h"
#include "llvm/MC/MCObjectFileInfo.h"
#include "llvm/MC/MCTargetOptions.h"
#include "llvm/Object/MachO.h"
#include "llvm/Support/InitLLVM.h"
#include "llvm/Support/TargetRegistry.h"
#include "llvm/Support/TargetSelect.h"

using namespace llvm;
using namespace llvm::object;

staticStringRef ToolName;
staticStringRef CurrInputFile;

// copy from llvm-objdump.cpp
LLVM_ATTRIBUTE_NORETURN void reportError(StringRef File,
                                         const Twine &Message) {
    outs().flush();
    WithColor::error(errs(), ToolName) << "'" << File << "': " << Message << "\n";
    exit(1);
}

// copy from llvm-objdump.h
template <typename T, typename... Ts>
T unwrapOrError(Expected<T> EO, Ts &&... Args) {
    if (EO)
        return std::move(*EO);
    assert(0 && "error in unwrapOrError()");
}

// copy from llvm-objdump.cpp
static cl::OptionCategory Elf2hexCat("elf2hex Options");

static cl::list<std::string> InputFileNames(cl::Positional,
                                           cl::desc("<input object files>"),
                                           cl::ZeroOrMore,
                                           cl::cat(Elf2hexCat));

std::string TripleName = "";

```

(continues on next page)

(continued from previous page)

```

static const Target *getTarget(const ObjectFile *Obj) {
    // Figure out the target triple.
    Triple TheTriple("unknown-unknown-unknown");
    TheTriple = Obj->makeTriple();

    // Get the target specific parser.
    std::string Error;
    const Target *TheTarget = TargetRegistry::lookupTarget("", TheTriple,
                                                            Error);
    if (!TheTarget)
        reportError(Obj->getFileName(), "can't find target: " + Error);

    // Update the triple name and return the found target.
    TripleName = TheTriple.getTriple();
    return TheTarget;
}

bool isRelocAddressLess(RelocationRef A, RelocationRef B) {
    return A.getOffset() < B.getOffset();
}

void error(std::error_code EC) {
    if (!EC)
        return;
    WithColor::error(errs(), ToolName)
        << "reading file: " << EC.message() << ".\n";
    errs().flush();
    exit(1);
}

static void getName(llvm::object::SectionRef const &Section, StringRef Name) {
    Name = unwrapOrError(Section.getName(), CurrInputFile);
#ifdef ELF2HEX_DEBUG
    llvm::dbgs() << Name << "\n";
#endif
}

static cl::opt<bool>
LittleEndian("le",
cl::desc("Little endian format"));

#ifdef ELF2HEX_DEBUG
// Modified from PrintSectionHeaders()
uint64_t GetSectionHeaderStartAddress(const ObjectFile *Obj,
    StringRef sectionName) {
    // outs() << "Sections:\n"
    //      "Idx Name          Size      Address      Type\n";
    std::error_code ec;
    unsigned i = 0;
    for (const SectionRef &Section : Obj->sections()) {

```

(continues on next page)

(continued from previous page)

```

    error(ec);
   StringRef Name;
    error(getName(Section, Name));
    uint64_t Address;
    Address = Section.getAddress();
    uint64_t Size;
    Size = Section.getSize();
    bool Text;
    Text = Section.isText();
    if (Name == sectionName)
        return Address;
    else
        return 0;
    ++i;
}
return 0;
}
#endif

// Reference from llvm::printSymbolTable of llvm-objdump.cpp
uint64_t GetSymbolAddress(const ObjectFile *o, StringRef SymbolName) {
    for (const SymbolRef &Symbol : o->symbols()) {
        Expected<uint64_t> AddressOrError = Symbol.getAddress();
        if (!AddressOrError)
            reportError(o->getFileName(), SymbolName);
        uint64_t Address = *AddressOrError;
        Expected<SymbolRef::Type> TypeOrError = Symbol.getType();
        if (!TypeOrError)
            reportError(o->getFileName(), SymbolName);
        SymbolRef::Type Type = *TypeOrError;
        section_iterator Section = unwrapOrError(Symbol.getSection(), CurrInputFile);
        StringRef Name;
        if (Type == SymbolRef::ST_Debug && Section != o->section_end()) {
            if (Expected<StringRef> NameOrErr = Section->getName())
                Name = *NameOrErr;
            else
                consumeError(NameOrErr.takeError());
        } else {
            Name = unwrapOrError(Symbol.getName(), o->getFileName());
        }
        if (Name == SymbolName)
            return Address;
    }
    return 0;
}

uint64_t SectionOffset(const ObjectFile *o, StringRef secName) {
    for (const SectionRef &Section : o->sections()) {
        StringRef Name;
        uint64_t BaseAddr;
        Name = unwrapOrError(Section.getName(), o->getFileName());
        unwrapOrError(Section.getContents(), o->getFileName());
    }
}

```

(continues on next page)

(continued from previous page)

```

    BaseAddr = Section.getAddress();

    if (Name == secName)
        return BaseAddr;
    }
    return 0;
}

using namespace llvm::elf2hex;

Reader reader;

VerilogHex::VerilogHex(std::unique_ptr<MCInstPrinter>& instructionPointer,
    std::unique_ptr<const MCSubtargetInfo>& subTargetInfo, const ObjectFile *Obj) :
    IP(instructionPointer), STI(subTargetInfo) {
    lastDumpAddr = 0;
#ifdef ELF2HEX_DEBUG
    //uint64_t startAddr = GetSectionHeaderStartAddress(Obj, "_start");
    //errs() << format("_start address:%08" PRIx64 "\n", startAddr);
#endif
    uint64_t isrAddr = GetSymbolAddress(Obj, "ISR");
    errs() << format("ISR address:%08" PRIx64 "\n", isrAddr);

    //uint64_t pltOffset = SectionOffset(Obj, ".plt");
    uint64_t textOffset = SectionOffset(Obj, ".text");
    PrintBootSection(textOffset, isrAddr, LittleEndian);
    lastDumpAddr = BOOT_SIZE;
    Fill0s(lastDumpAddr, 0x100);
    lastDumpAddr = 0x100;
}

void VerilogHex::PrintBootSection(uint64_t textOffset, uint64_t isrAddr,
    bool isLittleEndian) {
    uint64_t offset = textOffset - 4;

    // isr instruction at 0x8 and PC counter point to next instruction
    uint64_t isrOffset = isrAddr - 8 - 4;
    if (isLittleEndian) {
        outs() << "/*      0:*/      ";
        outs() << format("%02" PRIx64 " ", (offset & 0xff));
        outs() << format("%02" PRIx64 " ", (offset & 0xff00) >> 8);
        outs() << format("%02" PRIx64 " ", (offset & 0xff0000) >> 16);
        outs() << " 36";
        outs() << "                                /*      jmp      0x";
        outs() << format("%02" PRIx64 "%02" PRIx64 "%02" PRIx64 " */\n",
            (offset & 0xff0000) >> 16, (offset & 0xff00) >> 8, (offset & 0xff));
        outs() <<
            "/*      4:*/      04 00 00 36                                /
↪ *      jmp      4 */\n";
        offset -= 8;
        outs() << "/*      8:*/      ";
        outs() << format("%02" PRIx64 " ", (isrOffset & 0xff));
    }
}

```

(continues on next page)

(continued from previous page)

```

outs() << format("%02" PRIx64 " ", (isrOffset & 0xff00) >> 8);
outs() << format("%02" PRIx64 "", (isrOffset & 0xff0000) >> 16);
outs() << " 36";
outs() << "                                /*      jmp      0x";
outs() << format("%02" PRIx64 "%02" PRIx64 "%02" PRIx64 " */\n",
    (isrOffset & 0xff0000) >> 16, (isrOffset & 0xff00) >> 8, (isrOffset & 0xff));
outs() <<
    "/*      c:*/      fc ff ff 36                                /
→ *      jmp      -4 */\n";
}
else {
    outs() << "/*      0:*/      36 ";
    outs() << format("%02" PRIx64 " ", (offset & 0xff0000) >> 16);
    outs() << format("%02" PRIx64 " ", (offset & 0xff00) >> 8);
    outs() << format("%02" PRIx64 "", (offset & 0xff));
    outs() << "                                /*      jmp      0x";
    outs() << format("%02" PRIx64 "%02" PRIx64 "%02" PRIx64 " */\n",
        (offset & 0xff0000) >> 16, (offset & 0xff00) >> 8, (offset & 0xff));
    outs() <<
        "/*      4:*/      36 00 00 04                                /
→ *      jmp      4 */\n";
    offset -= 8;
    outs() << "/*      8:*/      36 ";
    outs() << format("%02" PRIx64 " ", (isrOffset & 0xff0000) >> 16);
    outs() << format("%02" PRIx64 " ", (isrOffset & 0xff00) >> 8);
    outs() << format("%02" PRIx64 "", (isrOffset & 0xff));
    outs() << "                                /*      jmp      0x";
    outs() << format("%02" PRIx64 "%02" PRIx64 "%02" PRIx64 " */\n",
        (isrOffset & 0xff0000) >> 16, (isrOffset & 0xff00) >> 8, (isrOffset & 0xff));
    outs() <<
        "/*      c:*/      36 ff ff fc                                /
→ *      jmp      -4 */\n";
}
}

// Fill /*address*/ 00 00 00 00 [startAddr..endAddr] from startAddr to endAddr.
// Include startAddr and endAddr.
void VerilogHex::Fill0s(uint64_t startAddr, uint64_t endAddr) {
    std::size_t addr;

    assert((startAddr <= endAddr) && "startAddr must <= BaseAddr");
    // Fill /*address*/ 00 00 00 00 for 4 bytes alignment (1 Cpu0 word size)
    for (addr = startAddr; addr < endAddr; addr += 4) {
        outs() << format("/%8" PRIx64 " */", addr);
        outs() << format("%02" PRIx64 " ", 0) << format("%02" PRIx64 " ", 0) \
            << format("%02" PRIx64 " ", 0) << format("%02" PRIx64 " ", 0) << '\n';
    }

    return;
}

void VerilogHex::ProcessDisAsmInstruction(MCInst inst, uint64_t Size,

```

(continues on next page)

(continued from previous page)

```

                                ArrayRef<uint8_t> Bytes, const ObjectFile *Obj) {
    SectionRef Section = reader.CurrentSection();
   StringRef Name;
   StringRef Contents;
    Name = unwrapOrError(Section.getName(), Obj->getFileName());
    unwrapOrError(Section.getContents(), Obj->getFileName());
    uint64_t SectionAddr = Section.getAddress();
    uint64_t Index = reader.CurrentIndex();
#ifdef ELF2HEX_DEBUG
    errs() << format("SectionAddr + Index = %8" PRIx64 "\n", SectionAddr + Index);
    errs() << format("lastDumpAddr %8" PRIx64 "\n", lastDumpAddr);
#endif
    if (lastDumpAddr < SectionAddr) {
        Fill0s(lastDumpAddr, SectionAddr - 1);
        lastDumpAddr = SectionAddr;
    }

    // print section name when meeting it first time
    if (sectionName != Name) {
        StringRef SegmentName = "";
        if (const MachOObjectFile *MachO =
            dyn_cast<const MachOObjectFile>(Obj)) {
            DataRefImpl DR = Section.getRawDataRefImpl();
            SegmentName = MachO->getSectionFinalSegmentName(DR);
        }
        outs() << "/*" << "Disassembly of section ";
        if (!SegmentName.empty())
            outs() << SegmentName << ", ";
        outs() << Name << ':' << "*/";
        sectionName = Name;
    }

    if (si != reader.CurrentSi()) {
        // print function name in section .text just before the first instruction
        // is printed
        outs() << '\n' << "/*" << reader.CurrentSymbol() << "*/\n";
        si = reader.CurrentSi();
    }

    // print instruction address
    outs() << format("/%8" PRIx64 "*/", SectionAddr + Index);

    // print instruction in hex format
    outs() << "\t";
    dumpBytes(Bytes.slice(Index, Size), outs());

    outs() << "/*";
    // print disassembly instruction to outs()
    IP->printInst(&inst, 0, "", *STI, outs());
    outs() << "*/";
    outs() << "\n";

```

(continues on next page)

(continued from previous page)

```

// In section .plt or .text, the Contents.size() maybe < (SectionAddr + Index + 4)
if (Contents.size() < (SectionAddr + Index + 4))
    lastDumpAddr = SectionAddr + Index + 4;
else
    lastDumpAddr = SectionAddr + Contents.size();
}

void VerilogHex::ProcessDataSection(SectionRef Section) {
    std::string Error;
    StringRef Name;
    StringRef Contents;
    uint64_t BaseAddr;
    uint64_t size;
    getName(Section, Name);
    unwrapOrError(Section.getContents(), CurrInputFile);
    BaseAddr = Section.getAddress();

#ifdef ELF2HEX_DEBUG
    errs() << format("BaseAddr = %8" PRIx64 "\n", BaseAddr);
    errs() << format("lastDumpAddr %8" PRIx64 "\n", lastDumpAddr);
#endif
    if (lastDumpAddr < BaseAddr) {
        Fill0s(lastDumpAddr, BaseAddr - 1);
        lastDumpAddr = BaseAddr;
    }
    if ((Name == ".bss" || Name == ".sbss") && Contents.size() > 0) {
        size = (Contents.size() + 3)/4*4;
        Fill0s(BaseAddr, BaseAddr + size - 1);
        lastDumpAddr = BaseAddr + size;
        return;
    }
    else {
        PrintDataSection(Section);
    }
}

void VerilogHex::PrintDataSection(SectionRef Section) {
    std::string Error;
    StringRef Name;
    uint64_t BaseAddr;
    uint64_t size;
    getName(Section, Name);
    StringRef Contents = unwrapOrError(Section.getContents(), CurrInputFile);
    BaseAddr = Section.getAddress();

    if (Contents.size() <= 0) {
        return;
    }
    size = (Contents.size()+3)/4*4;

    outs() << "/*Contents of section " << Name << ":\n";
    // Dump out the content as hex and printable ascii characters.

```

(continues on next page)

(continued from previous page)

```

for (std::size_t addr = 0, end = Contents.size(); addr < end; addr += 16) {
    outs() << format("/%8" PRIx64 " */", BaseAddr + addr);
    // Dump line of hex.
    for (std::size_t i = 0; i < 16; ++i) {
        if (i != 0 && i % 4 == 0)
            outs() << ' ';
        if (addr + i < end)
            outs() << hexdigit((Contents[addr + i] >> 4) & 0xF, true)
                << hexdigit(Contents[addr + i] & 0xF, true) << " ";
    }
    // Print ascii.
    outs() << "/*" << " ";
    for (std::size_t i = 0; i < 16 && addr + i < end; ++i) {
        if (std::isprint(static_cast<unsigned char>(Contents[addr + i]) & 0xFF))
            outs() << Contents[addr + i];
        else
            outs() << ".";
    }
    outs() << "*/" << "\n";
}
for (std::size_t i = Contents.size(); i < size; i++) {
    outs() << "\00 ";
}
outs() << "\n";
#ifdef ELF2HEX_DEBUG
errs() << "Name " << Name << " BaseAddr ";
errs() << format("%8" PRIx64 " Contents.size() ", BaseAddr);
errs() << format("%8" PRIx64 " size ", Contents.size());
errs() << format("%8" PRIx64 " \n", size);
#endif
// save the end address of this section to lastDumpAddr
lastDumpAddr = BaseAddr + size;
}

StringRef Reader::CurrentSymbol() {
    return Symbols[si].second;
}

SectionRef Reader::CurrentSection() {
    return _section;
}

unsigned Reader::CurrentSi() {
    return si;
}

uint64_t Reader::CurrentIndex() {
    return Index;
}

// Porting from DisassembleObject() of llvm-objdump.cpp
void Reader::DisassembleObject(const ObjectFile *Obj

```

(continues on next page)

(continued from previous page)

```

/*, bool InlineRelocs*/ , std::unique_ptr<MCDisassembler>& DisAsm,
std::unique_ptr<MCInstPrinter>& IP,
std::unique_ptr<const MCSubtargetInfo>& STI) {
  VerilogHex hexOut(IP, STI, Obj);
  std::error_code ec;
  for (const SectionRef &Section : Obj->sections()) {
    _section = Section;
    uint64_t BaseAddr;
    unwrapOnError(Section.getContents(), Obj->getFileName());
    BaseAddr = Section.getAddress();
    uint64_t SectSize = Section.getSize();
    if (!SectSize)
      continue;

    if (BaseAddr < 0x100)
      continue;

#ifdef ELF2HEX_DEBUG
    StringRef SectionName = unwrapOnError(Section.getName(), Obj->getFileName());
    errs() << "SectionName " << SectionName << format(" BaseAddr %8" PRIx64 "\n",
↳BaseAddr);
#endif

    bool text;
    text = Section.isText();
    if (!text) {
      hexOut.ProcessDataSection(Section);
      continue;
    }
    // It's .text section
    uint64_t SectionAddr;
    SectionAddr = Section.getAddress();

    // Make a list of all the symbols in this section.
    for (const SymbolRef &Symbol : Obj->symbols()) {
      if (Section.containsSymbol(Symbol)) {
        Expected<uint64_t> AddressOrErr = Symbol.getAddress();
        error(errorToErrorCode(AddressOrErr.takeError()));
        uint64_t Address = *AddressOrErr;
        Address -= SectionAddr;
        if (Address >= SectSize)
          continue;

        Expected<StringRef> Name = Symbol.getName();
        error(errorToErrorCode(Name.takeError()));
        Symbols.push_back(std::make_pair(Address, *Name));
      }
    }

    // Sort the symbols by address, just in case they didn't come in that way.
    array_pod_sort(Symbols.begin(), Symbols.end());
#ifdef ELF2HEX_DEBUG

```

(continues on next page)

(continued from previous page)

```

    for (unsigned si = 0, se = Symbols.size(); si != se; ++si) {
        errs() << '\n' << "/" << Symbols[si].first << " " << Symbols[si].second << "*/\n";
    }
#endif

    // Make a list of all the relocations for this section.
    std::vector<RelocationRef> Rels;

    // Sort relocations by address.
    std::sort(Rels.begin(), Rels.end(), isRelocAddressLess);

   StringRef name;
    getName(Section, name);

    // If the section has no symbols just insert a dummy one and disassemble
    // the whole section.
    if (Symbols.empty())
        Symbols.push_back(std::make_pair(0, name));

    SmallString<40> Comments;
    raw_svector_ostream CommentStream(Comments);

    ArrayRef<uint8_t> Bytes = arrayRefFromStringRef(
        unwrapOrError(Section.getContents(), Obj->getFileName()));
#if 0
    Section.getContents();
    ArrayRef<uint8_t> Bytes(reinterpret_cast<const uint8_t *>(BytesStr.data()),
        BytesStr.size());
#endif
    uint64_t Size;
    SectSize = Section.getSize();

    // Disassemble symbol by symbol.
    unsigned se;
    for (si = 0, se = Symbols.size(); si != se; ++si) {
        uint64_t Start = Symbols[si].first;
        uint64_t End;
        // The end is either the size of the section or the beginning of the next
        // symbol.
        if (si == se - 1)
            End = SectSize;
        // Make sure this symbol takes up space.
        else if (Symbols[si + 1].first != Start)
            End = Symbols[si + 1].first - 1;
        else {
            continue;
        }

        for (Index = Start; Index < End; Index += Size) {
            MCInst Inst;
            if (DisAsm->getInstruction(Inst, Size, Bytes.slice(Index),

```

(continues on next page)

(continued from previous page)

```

        SectionAddr + Index, CommentStream)) {
    hexOut.ProcessDisasmInstruction(Inst, Size, Bytes, Obj);
} else {
    errs() << ToolName << ": warning: invalid instruction encoding\n";
    if (Size == 0)
        Size = 1; // skip illegible bytes
    }
} // for
} // for
}
}

// Porting from disassembleObject() of llvm-objdump.cpp
static void Elf2Hex(const ObjectFile *Obj) {

    const Target *TheTarget = getTarget(Obj);

    // Package up features to be passed to target/subtarget
    SubtargetFeatures Features = Obj->getFeatures();

    std::unique_ptr<const MCRegisterInfo> MRI(TheTarget->createMCRegInfo(TripleName));
    if (!MRI)
        report_fatal_error("error: no register info for target " + TripleName);

    // Set up disassembler.
    MCTargetOptions MCOptions;
    std::unique_ptr<const MCAsmInfo> AsmInfo(
        TheTarget->createMCAsmInfo(*MRI, TripleName, MCOptions));
    if (!AsmInfo)
        report_fatal_error("error: no assembly info for target " + TripleName);

    std::unique_ptr<const MCSubtargetInfo> STI(
        TheTarget->createMCSubtargetInfo(TripleName, "", Features.getString()));
    if (!STI)
        report_fatal_error("error: no subtarget info for target " + TripleName);

    std::unique_ptr<const MCInstrInfo> MII(TheTarget->createMCInstrInfo());
    if (!MII)
        report_fatal_error("error: no instruction info for target " + TripleName);

    MCObjectFileInfo MOFI;
    MCContext Ctx(AsmInfo.get(), MRI.get(), &MOFI);
    // FIXME: for now initialize MCObjectFileInfo with default values
    MOFI.InitMCObjectFileInfo(Triple(TripleName), false, Ctx);

    std::unique_ptr<MCDisassembler> DisAsm(
        TheTarget->createMCDisassembler(*STI, Ctx));
    if (!DisAsm)
        report_fatal_error("error: no disassembler for target " + TripleName);

    std::unique_ptr<const MCInstrAnalysis> MIA(
        TheTarget->createMCInstrAnalysis(MII.get()));

```

(continues on next page)

(continued from previous page)

```

int AsmPrinterVariant = AsmInfo->getAssemblerDialect();
std::unique_ptr<MCInstPrinter> IP(TheTarget->createMCInstPrinter(
    Triple(TripleName), AsmPrinterVariant, *AsmInfo, *MII, *MRI));
if (!IP)
    report_fatal_error("error: no instruction printer for target " +
                       TripleName);

std::error_code EC;
reader.DisassembleObject(Obj, DisAsm, IP, STI);
}

static void DumpObject(const ObjectFile *o) {
    outs() << "/*";
    outs() << o->getFileName()
        << ":\tfile format " << o->getFileFormatName() << "*/";
    outs() << "\n\n";

    Elf2Hex(o);
}

/// @brief Open file and figure out how to dump it.
static void DumpInput(StringRef file) {
    CurrInputFile = file;
    // Attempt to open the binary.
    Expected<OwningBinary<Binary>> BinaryOrErr = createBinary(file);
    if (!BinaryOrErr)
        reportError(file, "no this file");

    Binary &Binary = *BinaryOrErr.get().getBinary();

    if (ObjectFile *o = dyn_cast<ObjectFile>(&Binary))
        DumpObject(o);
    else
        reportError(file, "invalid_file_type");
}

int main(int argc, char **argv) {
    // Print a stack trace if we signal out.
    //sys::PrintStackTraceOnErrorSignal(argv[0]);
    //PrettyStackTraceProgram X(argc, argv);
    //llvm_shutdown_obj Y; // Call llvm_shutdown() on exit.

    using namespace llvm;
    InitLLVM X(argc, argv);

    // Initialize targets and assembly printers/parsers.
    llvm::InitializeAllTargetInfos();
    llvm::InitializeAllTargetMCs();
    llvm::InitializeAllDisassemblers();

    // Register the target printer for --version.

```

(continues on next page)

(continued from previous page)

```

cl::AddExtraVersionPrinter(TargetRegistry::printRegisteredTargetsForVersion);

cl::ParseCommandLineOptions(argc, argv, "llvm object file dumper\n");
// TripleName = Triple::normalize(TripleName);

ToolName = argv[0];

// Defaults to a.out if no filenames specified.
if (InputFilenames.size() == 0)
    InputFilenames.push_back("a.out");

std::for_each(InputFilenames.begin(), InputFilenames.end(),
              DumpInput);

return EXIT_SUCCESS;
}

```

In order to support command, **llvm-objdump -d** and **llvm-objdump -t**, for Cpu0, the code add to `llvm-objdump.cpp` as follows,

exlbt/llvm-objdump/llvm-objdump.cpp

```
case ELF::EM_CPU0: //Cpu0
```

3.2 Create Cpu0 backend under LLD

3.2.1 LLD introduction

In general, linker do the Relocation Records Resolve as Chapter ELF support depicted, and optimization for those cannot finish in compiler stage. One of the optimization opportunities in linker is Dead Code Stripping which is explained as follows,

Dead code stripping - example (modified from llvm lto document web)

a.h

```

extern int foo1(void);
extern void foo2(void);
extern int foo4(void);

```

a.cpp

```
#include "a.h"

static signed int i = 0;

void foo2(void) {
    i = -1;
}

static int foo3() {
    return (10+foo4());
}

int foo1(void) {
    int data = 0;

    if (i < 0)
        data = foo3();

    data = data + 42;
    return data;
}
```

ch13_1.cpp

```
#include "a.h"

void ISR() {
    asm("ISR:");
    return;
}

int foo4(void) {
    return 5;
}

int main() {
    return foo1();
}
```

Above code can be reduced to [Fig. 3.2](#) to perform mark and swip in graph for Dead Code Stripping.

As above example, the foo2() is an isolated node without any reference. It's dead code and can be removed in linker optimization. We test this example by Makefile.ch13_1 and find foo2() cannot be removed. There are two possibilities for this situation. One is we do not trigger lld dead code stripping optimization in command (the default is not do it). The other is lld hasn't implemented it yet at this point. It's reasonable since the lld is in its early stages of development. We didn't dig it more, since the Cpu0 backend tutorial just need a linker to finish Relocation Records Resolve and see how it runs on PC.

Remind, llvm-linker is the linker works on IR level linker optimization. Sometime when you got the obj file only (if



Fig. 3.2: Atom classified (from lld web)

you have a.o in this case), the native linker (such as lld) have the opportunity to do Dead Code Stripping while the IR linker hasn't.

3.2.2 Static linker

Let's run the static linker first and explain it next.

File `printf-stdarg.c`⁷ come from internet download which is GPL2 license. GPL2 is more restricted than LLVM license.

exlbt/input/printf-stdarg-1.c

```

/*
Copyright 2001, 2002 Georges Menie (www.menie.org)
stdarg version contributed by Christian Ettinger

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

```

(continues on next page)

⁷ https://github.com/atgreen/FreeRTOS/blob/master/Demo/CORTEX_STM32F103_Primer_GCC/printf-stdarg.c

(continued from previous page)

```

    You should have received a copy of the GNU Lesser General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

/*
    putchar is the only external dependency for this file,
    if you have a working putchar, leave it commented out.
    If not, uncomment the define below and
    replace outbyte(c) by your own function call.

#define putchar(c) outbyte(c)
*/

// gcc printf-stdarg-1.c
// ./a.out

#include <stdio.h>

#define TEST_PRINTF

#ifdef TEST_PRINTF
int main(void)
{
    char *ptr = "Hello world!";
    char *np = 0;
    int i = 5;
    unsigned int bs = sizeof(int)*8;
    int mi;
    char buf[80];

    mi = (1 << (bs-1)) + 1;
    printf("%s\n", ptr);
    printf("printf test\n");
    printf("%s is null pointer\n", np);
    printf("%d = 5\n", i);
    printf("%d = - max int\n", mi);
    printf("char %c = 'a'\n", 'a');
    printf("hex %x = ff\n", 0xff);
    printf("hex %02x = 00\n", 0);
    printf("signed %d = unsigned %u = hex %x\n", -3, -3, -3);
    printf("%d %s(s)%", 0, "message");
    printf("\n");
    printf("%d %s(s) with %%\n", 0, "message");
    sprintf(buf, "justif: \"%-10s\"\n", "left"); printf("%s", buf);
    sprintf(buf, "justif: \"%10s\"\n", "right"); printf("%s", buf);
    sprintf(buf, " 3: %04d zero padded\n", 3); printf("%s", buf);
    sprintf(buf, " 3: %-4d left justif.\n", 3); printf("%s", buf);
    sprintf(buf, " 3: %4d right justif.\n", 3); printf("%s", buf);
    sprintf(buf, "-3: %04d zero padded\n", -3); printf("%s", buf);
    sprintf(buf, "-3: %-4d left justif.\n", -3); printf("%s", buf);

```

(continues on next page)

(continued from previous page)

```

    sprintf(buf, "-3: %4d right justif.\n", -3); printf("%s", buf);

    return 0;
}

/*
 * if you compile this file with
 * gcc -Wall $(YOUR_C_OPTIONS) -DTEST_PRINTF -c printf.c
 * you will get a normal warning:
 * printf.c:214: warning: spurious trailing '%' in format
 * this line is testing an invalid % at the end of the format string.
 *
 * this should display (on 32bit int machine) :
 *
 * Hello world!
 * printf test
 * (null) is null pointer
 * 5 = 5
 * -2147483647 = - max int
 * char a = 'a'
 * hex ff = ff
 * hex 00 = 00
 * signed -3 = unsigned 4294967293 = hex ffffffff
 * 0 message(s)
 * 0 message(s) with %
 * justif: "left      "
 * justif: "      right"
 * 3: 0003 zero padded
 * 3: 3    left justif.
 * 3:    3 right justif.
 * -3: -003 zero padded
 * -3: -3   left justif.
 * -3:   -3 right justif.
 */
#endif

```

exlbt/input/printf-stdarg-def.c

```

#include "print.h"

// Definition putchar(int c) for printf-stdarg.c
// For memory IO
int putchar(int c)
{
    char *p = (char*)IOADDR;
    *p = c;

    return 0;
}

```

exlbt/input/printf-stdarg.c

```
/*
Copyright 2001, 2002 Georges Menie (www.menie.org)
stdarg version contributed by Christian Ettinger

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
*/

/*
putchar is the only external dependency for this file,
if you have a working putchar, leave it commented out.
If not, uncomment the define below and
replace outbyte(c) by your own function call.

#define putchar(c) outbyte(c)
*/

#include <stdarg.h>

static void printchar(char **str, int c)
{
    extern int putchar(int c);

    if (str) {
        **str = c;
        ++(*str);
    }
    else (void)putchar(c);
}

#define PAD_RIGHT 1
#define PAD_ZERO 2

static int prints(char **out, const char *string, int width, int pad)
{
    int pc = 0, padchar = ' ';

    if (width > 0) {
        int len = 0;
        const char *ptr;
```

(continues on next page)

(continued from previous page)

```

    for (ptr = string; *ptr; ++ptr) ++len;
    if (len >= width) width = 0;
    else width -= len;
    if (pad & PAD_ZERO) padchar = '0';
}
if (!(pad & PAD_RIGHT)) {
    for ( ; width > 0; --width) {
        printchar (out, padchar);
        ++pc;
    }
}
for ( ; *string ; ++string) {
    printchar (out, *string);
    ++pc;
}
for ( ; width > 0; --width) {
    printchar (out, padchar);
    ++pc;
}

return pc;
}

/* the following should be enough for 32 bit int */
#define PRINT_BUF_LEN 12

static int printi(char **out, int i, int b, int sg, int width, int pad, int letbase)
{
    char print_buf[PRINT_BUF_LEN];
    char *s;
    int t, neg = 0, pc = 0;
    unsigned int u = i;

    if (i == 0) {
        print_buf[0] = '0';
        print_buf[1] = '\0';
        return prints (out, print_buf, width, pad);
    }

    if (sg && b == 10 && i < 0) {
        neg = 1;
        u = -i;
    }

    s = print_buf + PRINT_BUF_LEN-1;
    *s = '\0';

    while (u) {
        t = u % b;
        if( t >= 10 )
            t += letbase - '0' - 10;
        *--s = t + '0';
    }

```

(continues on next page)

(continued from previous page)

```

    u /= b;
}

if (neg) {
    if( width && (pad & PAD_ZERO) ) {
        putchar (out, '-');
        ++pc;
        --width;
    }
    else {
        *--s = '-';
    }
}

return pc + prints (out, s, width, pad);
}

static int print(char **out, const char *format, va_list args )
{
    int width, pad;
    int pc = 0;
    char scr[2];

    for (; *format != 0; ++format) {
        if (*format == '%') {
            ++format;
            width = pad = 0;
            if (*format == '\\0') break;
            if (*format == '%') goto out;
            if (*format == '-') {
                ++format;
                pad = PAD_RIGHT;
            }
            //bool have_ll = (format[0] == 'l' && format[1] == 'l');
            //pc += have_ll * 2;
            while (*format == '0') {
                ++format;
                pad |= PAD_ZERO;
            }
            for ( ; *format >= '0' && *format <= '9'; ++format) {
                width *= 10;
                width += *format - '0';
            }
            if( *format == 's' ) {
                char *s = (char *)va_arg( args, int );
                pc += prints (out, s?s:"(null)", width, pad);
                continue;
            }
            if( *format == 'd' ) {
                pc += printi (out, va_arg( args, int ), 10, 1, width, pad, 'a');
                continue;
            }
        }
    }
}

```

(continues on next page)

(continued from previous page)

```

    if( *format == 'x' ) {
        pc += printi (out, va_arg( args, int ), 16, 0, width, pad, 'a');
        continue;
    }
    if( *format == 'X' ) {
        pc += printi (out, va_arg( args, int ), 16, 0, width, pad, 'A');
        continue;
    }
    if( *format == 'u' ) {
        pc += printi (out, va_arg( args, int ), 10, 0, width, pad, 'a');
        continue;
    }
    if( *format == 'c' ) {
        /* char are converted to int then pushed on the stack */
        scr[0] = (char)va_arg( args, int );
        scr[1] = '\0';
        pc += prints (out, scr, width, pad);
        continue;
    }
}
else {
out:
    printchar (out, *format);
    ++pc;
}
}
if (out) **out = '\0';
va_end( args );
return pc;
}

int printf(const char *format, ...)
{
    va_list args;

    va_start( args, format );
    return print( 0, format, args );
}

int sprintf(char *out, const char *format, ...)
{
    va_list args;

    va_start( args, format );
    return print( &out, format, args );
}

#ifdef TEST_PRINTF
int main(void)
{
    char ptr[] = "Hello world!";
    char *np = 0;

```

(continues on next page)

(continued from previous page)

```

int i = 5;
unsigned int bs = sizeof(int)*8;
int mi;
char buf[80];

mi = (1 << (bs-1)) + 1;
printf("%s\n", ptr);
printf("printf test\n");
printf("%s is null pointer\n", np);
printf("%d = 5\n", i);
printf("%d = - max int\n", mi);
printf("char %c = 'a'\n", 'a');
printf("hex %x = ff\n", 0xff);
printf("hex %02x = 00\n", 0);
printf("signed %d = unsigned %u = hex %x\n", -3, -3, -3);
printf("%d %s(s)", 0, "message");
printf("\n");
printf("%d %s(s) with %%\n", 0, "message");
sprintf(buf, "justif: \"%-10s\"\n", "left"); printf("%s", buf);
sprintf(buf, "justif: \"%10s\"\n", "right"); printf("%s", buf);
sprintf(buf, " 3: %04d zero padded\n", 3); printf("%s", buf);
sprintf(buf, " 3: %-4d left justif.\n", 3); printf("%s", buf);
sprintf(buf, " 3: %4d right justif.\n", 3); printf("%s", buf);
sprintf(buf, "-3: %04d zero padded\n", -3); printf("%s", buf);
sprintf(buf, "-3: %-4d left justif.\n", -3); printf("%s", buf);
sprintf(buf, "-3: %4d right justif.\n", -3); printf("%s", buf);

return 0;
}

/*
 * if you compile this file with
 * gcc -Wall $(YOUR_C_OPTIONS) -DTEST_PRINTF -c printf.c
 * you will get a normal warning:
 * printf.c:214: warning: spurious trailing '%' in format
 * this line is testing an invalid % at the end of the format string.
 *
 * this should display (on 32bit int machine) :
 *
 * Hello world!
 * printf test
 * (null) is null pointer
 * 5 = 5
 * -2147483647 = - max int
 * char a = 'a'
 * hex ff = ff
 * hex 00 = 00
 * signed -3 = unsigned 4294967293 = hex ffffffff
 * 0 message(s)
 * 0 message(s) with %
 * justif: "left      "
 * justif: "          right"

```

(continues on next page)

(continued from previous page)

```

* 3: 0003 zero padded
* 3: 3    left justif.
* 3:  3   right justif.
* -3: -003 zero padded
* -3: -3   left justif.
* -3:  -3  right justif.
*/

#endif

```

exlbt/input/start.cpp

```

#include "start.h"

extern int main();

// Real entry (first instruction) is from cpu0BootAtomContent of
// Cpu0RelocationPass.cpp jump to asm("start:") of start.cpp.
void start() {
    asm("start:");

    INIT_SP
    initRegs();
    main();
    asm("addiu $lr, $ZERO, -1");
    asm("ret $lr");
}

```

exlbt/input/lib_cpu0.c

```

// This file provides function symbols used for c++ in lld

void _start() {}
void __stack_chk_fail() {}
void __stack_chk_guard() {}
void _ZdlPv() {}
void __dso_handle() {}
void _ZNSt8ios_base4InitC1Ev() {}
void __cxa_atexit() {}
void _ZTVN10__cxxabiv120__si_class_type_infoE() {}
void _ZTVN10__cxxabiv117__class_type_infoE() {}
void _Znwm() {}
void __cxa_pure_virtual() {}
void _ZNSt8ios_base4InitD1Ev() {}

```

exlbt/input/Common.mk

```

# Thanks https://makefiletutorial.com

TARGET_EXEC := a.out
BUILD_DIR := ./build-$(CPU)-$(ENDIAN)
TARGET := $(BUILD_DIR)/$(TARGET_EXEC)
SRC_DIR := ./

TOOLDIR := ~/llvm/test/build/bin
CC := $(TOOLDIR)/clang
LLC := $(TOOLDIR)/llc
LD := $(TOOLDIR)/ld.lld
AS := $(TOOLDIR)/clang -static -fintegrated-as -c
RANLIB := $(TOOLDIR)/llvm-ranlib
READELF := $(TOOLDIR)/llvm-readelf

# String substitution for every C/C++ file.
# As an example, hello.cpp turns into ./build/hello.cpp.o
OBJS := $(SRCS:%=$(BUILD_DIR)/%.o)

# String substitution (suffix version without %).
# As an example, ./build/hello.cpp.o turns into ./build/hello.cpp.d
DEPS := $(OBJS:.o=.d)

# Add a prefix to INC_DIRS. So moduleA would become -ImoduleA. GCC understands
# this -I flag
INC_FLAGS := $(addprefix -I,$(INC_DIRS))

# The -MMD and -MP flags together generate Makefiles for us!
# These files will have .d instead of .o as the output.
# fintegrated-as: for asm code in C/C++
CPPFLAGS := -MMD -MP -target cpu0$(ENDIAN)-unknown-linux-gnu -static \
            -fintegrated-as $(INC_FLAGS) -march=cpu0$(ENDIAN) -mcpu=$(CPU) \
            -mllvm -has-lld=true -DHAS_COMPLEX

LLFLAGS := -mcpu=$(CPU) -relocation-model=static \
           -filetype=obj -has-lld=true

CFLAGS := -target cpu0$(ENDIAN)-unknown-linux-gnu -static -mcpu=$(CPU) \
          -fintegrated-as -Wno-error=implicit-function-declaration
CONFIGURE := CC="$(CC)" CFLAGS="$(CFLAGS)" AS="$(AS)" RANLIB="$(RANLIB)" \
             READELF="$(READELF)" ../newlib/configure --host=cpu0

#FIND_LIBBUILTINS_DIR := $(shell find . -iname $(LIBBUILTINS_DIR))

$(TARGET): $(OBJS) $(LIBS)
            $(LD) -o $@ $(OBJS) $(LIBS)

$(LIBS):
ifdef LIBBUILTINS_DIR
    $(MAKE) -C $(LIBBUILTINS_DIR)
endif

```

(continues on next page)

(continued from previous page)

```

ifndef NEWLIB_DIR
    $(MAKE) -C $(NEWLIB_DIR)/$(BUILD_DIR)
endif

# Build step for C source
$(BUILD_DIR)/%.c.o: %.c
    mkdir -p $(dir $@)
    $(CC) $(CPPFLAGS) $(CFLAGS) -c $< -o $@

# Build step for C++ source
$(BUILD_DIR)/%.cpp.o: %.cpp
    mkdir -p $(dir $@)
    $(CC) $(CPPFLAGS) $(CXXFLAGS) -c $< -o $@

.PHONY: clean
clean:
    rm -rf $(BUILD_DIR)
ifndef LIBBUILTINS_DIR
    cd $(LIBBUILTINS_DIR) && $(MAKE) -f Makefile clean
endif
ifndef NEWLIB_DIR
    cd $(NEWLIB_DIR) && rm -rf build-$(CPU)-$(ENDIAN)/*
endif

# Include the .d makefiles. The - at the f.cnt suppresses the er.crs.cf missing
# Makefiles. Initially, all the .d files will be missing, and we .cn't want t.cse
# er.crs .c s.cw up.
-include $(DEPS)

```

With the `printf()` of GPL source code, we can program more test code with it to verify the previous `llvm Cpu0` backend generated program. The following code is for this purpose.

exlbt/input/debug.cpp

```

#include "debug.h"

extern "C" int printf(const char *format, ...);

// With read variable form asm, such as sw in this example, the function,
// ISR_Handler() must entry from beginning. The ISR() enter from "ISR:" will
// has incorrect value for reload instruction in offset.
// For example, the correct one is:
//  "addiu $sp, $sp, -12"
//  "mov $fp, $sp"
// ISR:
//  "ld $2, 32($fp)"
// Go to ISR directly, then the $fp is 12+ than original, then it will get
//  "ld $2, 20($fp)" actually.
void ISR_Handler() {
    SAVE_REGISTERS;
    asm("lui $7, 0xffff");
}

```

(continues on next page)

(continued from previous page)

```

asm("ori $7, $7, 0xfdf");
asm("and $sw, $sw, $7"); // clear `IE

volatile int sw;
__asm__ __volatile__("addiu %0, $sw, 0"
                    : "=r"(sw)
                    );
int interrupt = (sw & INT);
int softint = (sw & SOFTWARE_INT);
int overflow = (sw & OVERFLOW);
int int1 = (sw & INT1);
int int2 = (sw & INT2);
if (interrupt) {
    if (softint) {
        if (overflow) {
            printf("Overflow exception\n");
            CLEAR_OVERFLOW;
        }
        else {
            printf("Software interrupt\n");
        }
        CLEAR_SOFTWARE_INT;
    }
    else if (int1) {
        printf("Hardware interrupt 0\n");
        asm("lui $7, 0xffff");
        asm("ori $7, $7, 0x7fff");
        asm("and $sw, $sw, $7");
    }
    else if (int2) {
        printf("Hardware interrupt 1\n");
        asm("lui $7, 0xfffe");
        asm("ori $7, $7, 0xffff");
        asm("and $sw, $sw, $7");
    }
    asm("lui $7, 0xffff");
    asm("ori $7, $7, 0xfdf");
    asm("and $sw, $sw, $7"); // clear `I
}
asm("ori $sw, $sw, 0x200"); // int enable
RESTORE_REGISTERS;
return;
}

void ISR() {
    asm("ISR:");
    asm("lui $at, 7");
    asm("ori $at, $at, 0xff00");
    asm("st $14, 48($at)");
    ISR_Handler();
    asm("lui $at, 7");
    asm("ori $at, $at, 0xff00");

```

(continues on next page)

(continued from previous page)

```

asm("ld $14, 48($at)");
asm("cmov $pc, $epc");
}

void int_sim() {
    asm("ori $sw, $sw, 0x200"); // int enable
    asm("ori $sw, $sw, 0x2000"); // set interrupt
    asm("ori $sw, $sw, 0x4000"); // Software interrupt
    asm("ori $sw, $sw, 0x200"); // int enable
    asm("ori $sw, $sw, 0x2000"); // set interrupt
    asm("ori $sw, $sw, 0x8000"); // hardware interrupt 0
    asm("ori $sw, $sw, 0x200"); // int enable
    asm("ori $sw, $sw, 0x2000"); // set interrupt
    asm("lui $at, 1");
    asm("or $sw, $sw, $at"); // hardware interrupt 1
    return;
}

```

exlbt/input/ch_1ld_staticlink.h

```

#include "debug.h"
#include "print.h"

// #define PRINT_TEST

extern "C" int printf(const char *format, ...);
extern "C" int sprintf(char *out, const char *format, ...);

extern unsigned char sBuffer[4];
extern int test_overflow();
extern int test_add_overflow();
extern int test_sub_overflow();
extern int test_ctrl2();
extern int test_phinode(int a, int b, int c);
extern int test_blockaddress(int x);
extern int test_longbranch();
extern int test_func_arg_struct();
extern int test_tailcall(int a);
extern bool exception0ccur;
extern int test_detect_exception(bool exception);

extern int test_staticlink();

```

exlbt/input/ch_ild_staticlink.cpp

```

#include "ch4_1_addsuboverflow.cpp"
#include "ch8_1_br_jt.cpp"
#include "ch8_2_phinode.cpp"
#include "ch8_1_blockaddr.cpp"
#include "ch8_2_longbranch.cpp"
#include "ch9_2_tailcall.cpp"
#include "ch9_3_detect_exception.cpp"

void test_printf() {
    char ptr[] = "Hello world!";
    char *np = 0;
    int i = 5;
    unsigned int bs = sizeof(int)*8;
    int mi;
    char buf[80];

    mi = (1 << (bs-1)) + 1;
    printf("%s\n", ptr);
    printf("printf test\n");
    printf("%s is null pointer\n", np);
    printf("%d = 5\n", i);
    printf("%d = - max int\n", mi);
    printf("char %c = 'a'\n", 'a');
    printf("hex %x = ff\n", 0xff);
    printf("hex %02x = 00\n", 0);
    printf("signed %d = unsigned %u = hex %x\n", -3, -3, -3);
    printf("%d %s(s)", 0, "message");
    printf("\n");
    printf("%d %s(s) with %%\n", 0, "message");
    sprintf(buf, "justif: \"%-10s\"\n", "left"); printf("%s", buf);
    sprintf(buf, "justif: \"%10s\"\n", "right"); printf("%s", buf);
    sprintf(buf, " 3: %04d zero padded\n", 3); printf("%s", buf);
    sprintf(buf, " 3: %-4d left justif.\n", 3); printf("%s", buf);
    sprintf(buf, " 3: %4d right justif.\n", 3); printf("%s", buf);
    sprintf(buf, "-3: %04d zero padded\n", -3); printf("%s", buf);
    sprintf(buf, "-3: %-4d left justif.\n", -3); printf("%s", buf);
    sprintf(buf, "-3: %4d right justif.\n", -3); printf("%s", buf);
}

void verify_test_ctrl2()
{
    int a = -1;
    int b = -1;
    int c = -1;
    int d = -1;

    sBuffer[0] = (unsigned char)0x35;
    sBuffer[1] = (unsigned char)0x35;
    a = test_ctrl2();
    sBuffer[0] = (unsigned char)0x30;

```

(continues on next page)

(continued from previous page)

```

sBuffer[1] = (unsigned char)0x29;
b = test_ctrl2();
sBuffer[0] = (unsigned char)0x35;
sBuffer[1] = (unsigned char)0x35;
c = test_ctrl2();
sBuffer[0] = (unsigned char)0x34;
d = test_ctrl2();
printf("test_ctrl2(): a = %d, b = %d, c = %d, d = %d", a, b, c, d);
if (a == 1 && b == 0 && c == 1 && d == 0)
    printf(", PASS\n");
else
    printf(", FAIL\n");

return;
}

int test_staticlink()
{
    int a = 0;

    // pre-defined compiler macro (from llc -march=cpu0${ENDIAN} or
    // clang -target cpu0${ENDIAN}-unknown-linux-gnu
    // http://beefchunk.com/documentation/lang/c/pre-defined-c/prearch.html
#ifdef __CPU0EB__
    printf("__CPU0EB__\n");
#endif
#ifdef __CPU0EL__
    printf("__CPU0EL__\n");
#endif
    test_printf();
    a = test_add_overflow();
    a = test_sub_overflow();
    a = test_global(); // gI = 100
    printf("global variable gI = %d", a);
    if (a == 100)
        printf(", PASS\n");
    else
        printf(", FAIL\n");
    verify_test_ctrl2();
    a = test_phinode(3, 1, 0);
    printf("test_phinode(3, 1) = %d", a); // a = 3
    if (a == 3)
        printf(", PASS\n");
    else
        printf(", FAIL\n");
    a = test_blockaddress(1);
    printf("test_blockaddress(1) = %d", a); // a = 1
    if (a == 1)
        printf(", PASS\n");
    else
        printf(", FAIL\n");
    a = test_blockaddress(2);

```

(continues on next page)

(continued from previous page)

```

printf("test_blockaddress(2) = %d", a); // a = 2
if (a == 2)
    printf(", PASS\n");
else
    printf(", FAIL\n");
a = test_longbranch();
printf("test_longbranch() = %d", a); // a = 0
if (a == 0)
    printf(", PASS\n");
else
    printf(", FAIL\n");
a = test_func_arg_struct();
printf("test_func_arg_struct() = %d", a); // a = 0
if (a == 0)
    printf(", PASS\n");
else
    printf(", FAIL\n");
a = test_constructor();
printf("test_constructor() = %d", a); // a = 0
if (a == 0)
    printf(", PASS\n");
else
    printf(", FAIL\n");
a = test_template();
printf("test_template() = %d", a); // a = 15
if (a == 15)
    printf(", PASS\n");
else
    printf(", FAIL\n");
long long res = test_template_ll();
printf("test_template_ll() = 0x%X-%X", (int)(res>>32), (int)res); // res = -1
if (res == -1)
    printf(", PASS\n");
else
    printf(", FAIL\n");
a = test_tailcall(5);
printf("test_tailcall(5) = %d", a); // a = 15
if (a == 120)
    printf(", PASS\n");
else
    printf(", FAIL\n");
test_detect_exception(true);
printf("exceptionOccur= %d", exceptionOccur);
if (exceptionOccur)
    printf(", PASS\n");
else
    printf(", FAIL\n");
test_detect_exception(false);
printf("exceptionOccur= %d", exceptionOccur);
if (!exceptionOccur)
    printf(", PASS\n");
else

```

(continues on next page)

(continued from previous page)

```

    printf(", FAIL\n");
    a = inlineasm_global(); // 4
    printf("inlineasm_global() = %d", a); // a = 4
    if (a == 4)
        printf(", PASS\n");
    else
        printf(", FAIL\n");
    a = test_cpp_polymorphism();
    printf("test_cpp_polymorphism() = %d", a); // a = 0
    if (a == 0)
        printf(", PASS\n");
    else
        printf(", FAIL\n");

    int_sim();

    return 0;
}

// test passing compilation only
#include "builtins-cpu0.c"

```

exlbt/input/ch_slinker.cpp

```

#include "ch_nolld.h"
#include "ch_lld_staticlink.h"

int main()
{
    bool pass = true;
    pass = test_nolld();
    if (pass)
        printf("test_nolld(): PASS\n");
    else
        printf("test_nolld(): FAIL\n");
    pass = true;
    pass = test_staticlink();

    return pass;
}

#include "ch_nolld.cpp"
#include "ch_lld_staticlink.cpp"

```

exlbt/input/Makefile.slinker

```
SRCS := start.cpp debug.cpp printf-stdarg-def.c printf-stdarg.c ch_slinker.cpp \
        lib_cpu0.c
INC_DIRS := $(SRC_DIR) $(LBDEX_DIR)/input
LIBBUILTINS_DIR :=
LIBS :=

include Common.mk
```

exlbt/input/make.sh

```
#!/usr/bin/env bash

# for example:
# bash make.sh cpu032I eb Makefile.newlib
# bash make.sh cpu032I eb Makefile.builtins
# bash make.sh cpu032I el Makefile.slinker
# bash make.sh cpu032II eb Makefile.float
# bash make.sh cpu032II el Makefile.ch13_1

LBDEX_DIR=$HOME/git/lbd/lbdex
NEWLIB_DIR=$HOME/git/newlib-cygwin

ARG_NUM=$#
CPU=$1
ENDIAN=$2
MKFILE=$3

BUILD_DIR=build-$CPU-$ENDIAN

build_newlib() {
    pushd $NEWLIB_DIR
    rm -rf $BUILD_DIR
    mkdir $BUILD_DIR
    cd build
    CC=$TOOLDIR/clang \
    CFLAGS="-target cpu0$ENDIAN-unknown-linux-gnu -mcpu=$CPU -static \
        -fintegrated-as -Wno-error=implicit-function-declaration" \
    AS="$TOOLDIR/clang -static -fintegrated-as -c" \
    AR="$TOOLDIR/llvm-ar" RANLIB="$TOOLDIR/llvm-ranlib" \
    READELF="$TOOLDIR/llvm-readelf" ../newlib/configure --host=cpu0
    make
    popd
}

prologue() {
    if [ $ARG_NUM == 0 ]; then
        echo "usage: bash $sh_name cpu_type ENDIAN"
        echo "  cpu_type: cpu032I or cpu032II"
    fi
}
```

(continues on next page)

(continued from previous page)

```

echo "  ENDIAN: be (big ENDIAN, default) or le (little ENDIAN)"
echo "for example:"
echo "  bash build-slinker.sh cpu032I eb"
exit 1;
fi
if [ $CPU != cpu032I ] && [ $CPU != cpu032II ]; then
    echo "1st argument is cpu032I or cpu032II"
    exit 1
fi

INCDIR=../../lbdex/input
OS=`uname -s`
echo "OS =" ${OS}

TOOLDIR=~/.llvm/test/build/bin
CLANG=~/.llvm/test/build/bin/clang

echo "CPU =" "${CPU}"
echo "ENDIAN =" "${ENDIAN}"

if [ $ENDIAN != eb ] && [ $ENDIAN != el ]; then
    echo "2nd argument is be (big ENDIAN, default) or le (little ENDIAN)"
    exit 1
fi

if [ $MKFILE == "Makefile.newlib" ] || [ $MKFILE == "Makefile.builtins" ]; then
    echo "build_newlib"
#    build_newlib;
fi
rm $BUILD_DIR/a.out
}

isLittleEndian() {
    echo "ENDIAN = " "$ENDIAN"
    if [ "$ENDIAN" == "LittleEndian" ] ; then
        LE="true"
    elif [ "$ENDIAN" == "BigEndian" ] ; then
        LE="false"
    else
        echo "!ENDIAN unknown"
        exit 1
    fi
}

elf2hex() {
    ${TOOLDIR}/elf2hex -le=$LE $BUILD_DIR/a.out > ${LBDEX_DIR}/verilog/cpu0.hex
    if [ $LE == "true" ] ; then
        echo "1 /* 0: big ENDIAN, 1: little ENDIAN */" > ${LBDEX_DIR}/verilog/cpu0.config
    else
        echo "0 /* 0: big ENDIAN, 1: little ENDIAN */" > ${LBDEX_DIR}/verilog/cpu0.config
    fi
    cat ${LBDEX_DIR}/verilog/cpu0.config
}

```

(continues on next page)

(continued from previous page)

```

}

epilogue() {
    ENDIAN=`${TOOLDIR}/llvm-readobj -h $BUILD_DIR/a.out|grep "DataEncoding"|awk '{print $2}'`
    isLittleEndian;
    elf2hex;
}

FILE=$3

if [ ! -f "$FILE" ]; then
    echo "$FILE does not exists."
    exit 0;
fi

prologue;

make -f $FILE CPU=${CPU} ENDIAN=${ENDIAN} LBDEX_DIR=${LBDEX_DIR} NEWLIB_DIR=${NEWLIB_DIR}
clean
make -f $FILE CPU=${CPU} ENDIAN=${ENDIAN} LBDEX_DIR=${LBDEX_DIR} NEWLIB_DIR=${NEWLIB_DIR}

epilogue;

```

```

$ cd ~/git/lbd/lbdex/verilog
$ make
$ cd ~/git/lbt/exlbt/input
$ pwd
$HOME/git/lbt/exlbt/input
$ bash make.sh cpu032I el Makefile.slinker
...
endian = LittleEndian
ISR address:00020780
1 /* 0: big endian, 1: little endian */
$ cd ~/git/lbd/lbdex/verilog
$ pwd
$HOME/git/lbd/lbdex/verilog
$ ./cpu0Is
WARNING: cpu0.v:487: $readmemh(cpu0.hex): Not enough words in the file for the requested
range [0:524287].
taskInterrupt(001)
74
7
0
0
253
3
1
13
3
-126

```

(continues on next page)

(continued from previous page)

```

130
-32766
32770
393307
16777222
-3
-4
51
2
3
1
2147483647
-2147483648
9
12
5
0
31
49
test_nolld(): PASS
__CPU0EL__
Hello world!
printf test
(null) is null pointer
5 = 5
-2147483647 = - max int
char a = 'a'
hex ff = ff
hex 00 = 00
signed -3 = unsigned 4294967293 = hex ffffffff
0 message(s)
0 message(s) with %
justif: "left    "
justif: "      right"
  3: 0003 zero padded
  3: 3    left justif.
  3:   3 right justif.
-3: -003 zero padded
-3: -3   left justif.
-3:  -3 right justif.
global variable gI = 100, PASS
test_ctrl2(): a = 1, b = 0, c = 1, d = 0, PASS
test_phinode(3, 1) = 3, PASS
test_blockaddress(1) = 1, PASS
test_blockaddress(2) = 2, PASS
test_longbranch() = 0, PASS
test_func_arg_struct() = 0, PASS
test_constructor() = 0, PASS
test_template() = 15, PASS
test_template_ll() = 0xffffffff-ffffffff, PASS
test_tailcall(5) = 120, PASS
exceptionOccur= 1, PASS

```

(continues on next page)

(continued from previous page)

```

exceptionOccur= 0, PASS
inlineasm_global() = 4, PASS
20
10
5
test_cpp_polymorphism() = 0, PASS
taskInterrupt(011)
Software interrupt
taskInterrupt(011)
Hardware interrupt 0
taskInterrupt(011)
Hardware interrupt 1
...
RET to PC < 0, finished!

```

Above test includes the printf format verification. Let's check the result with PC program printf-stdarg-1.c output as follows,

```

$ cd ~/git/lbt/exlbt/input
$ pwd
$HOME/git/lbt/exlbt/input
$ clang printf-stdarg-1.c
printf-stdarg-1.c:58:19: warning: incomplete format specifier [-Wformat]
    printf("%d %s(s)%", 0, "message");
                   ^
1 warning generated.
$ ./a.out
Hello world!
printf test
(null) is null pointer
5 = 5
-2147483647 = - max int
char a = 'a'
hex ff = ff
hex 00 = 00
signed -3 = unsigned 4294967293 = hex ffffffff
0 message(s)
0 message(s) with \%
justif: "left      "
justif: "      right"
3: 0003 zero padded
3: 3    left justif.
3:    3 right justif.
-3: -003 zero padded
-3: -3   left justif.
-3:   -3 right justif.

```

As above, by taking the open source code advantage, Cpu0 got the more stable printf() program. Once Cpu0 backend can translate the printf() function of the open source C printf() program into machine instructions, the llvm Cpu0 backend can be verified with printf(). With the quality code of open source printf() program, the Cpu0 toolchain is extended from compiler backend to C std library support. (Notice that some GPL open source code are not quality code, but some are.)

The “Overflow exception is printed twice meaning the ISR() of debug.cpp is called twice from ch4_1_2.cpp. The printed “taskInterrupt(001)” and “taskInterrupt(011)” just are trace message from cpu0.v code.

3.2.3 Dynamic linker

I remove dynamic linker demonstration from 3.9.0 because I don’t know how to do it from lld 3.9 and this demonstration add lots of code in elf2hex, verilog and lld of Cpu0 backend. However it can be run with llvm 3.7 with the following command.

```
1-160-136-173:test Jonathan$ pwd
/Users/Jonathan/test
1-160-136-173:test Jonathan$ git clone https://github.com/Jonathan2251/lbd
1-160-136-173:test Jonathan$ git clone https://github.com/Jonathan2251/lbt
1-160-136-173:test Jonathan$ cd lbd
1-160-136-173:lbd Jonathan$ pwd
/Users/Jonathan/test/lbd
1-160-136-173:lbd Jonathan$ git checkout release_374
1-160-136-173:lbd Jonathan$ cd ../lbt
1-160-136-173:test Jonathan$ git checkout release_374
1-160-136-173:lbt Jonathan$ make html
```

Then reading this section in lld.html for it.

3.3 Summary

3.3.1 Create a new backend base on LLVM

Thanks the llvm open source project. To write a linker and ELF to Hex tools for a new CPU architecture is easy and reliable. Combined with the llvm Cpu0 backend code and Verilog language code programmed in previous chapters, we design a software toolchain to compile C/C++ code, link and run it on Verilog Cpu0 simulator without any real hardware investment. If you buy the FPGA development hardware, we believe these code can run on FPGA CPU even though we didn’t do it. Extend system program toolchain to support a new CPU instruction set can be finished just like we have shown you at this point. School knowledges of system program, compiler, linker, loader, computer architecture and CPU design has been translated into a real work and see how it is running. Now, these school books knowledge is not limited on paper. We design it, program it, and run it on real world.

The total code size of llvm Cpu0 backend compiler, Cpu0 lld linker, elf2hex and Cpu0 Verilog Language is around 10 thousands lines of source code include comments. The total code size of clang, llvm and lld has 1000 thousands lines exclude the test and documents parts. It is only 1 % of the llvm size. More over, the llvm Cpu0 backend and lld Cpu0 backend are 70% of same with llvm Mips and lld X86_64. Based on this truth, we believe llvm is a well defined structure in compiler architecture.

3.3.2 Contribute back to Open Source through working and learning

Finally, 10 thousands lines of source code in Cpu0 backend is very small in UI program. But it's quite complex in system program which based on llvm. We spent 600 pages of pdf to explain these code. Open source code give programmers best opportunity to understand the code and enhance/extend the code function. But it can be better, we believe the documentation is the next most important thing to improve the open source code development. The Open Source Organization recognized this point and set Open Source Document Project years ago⁸⁹¹⁰¹¹¹². Open Source grows up and becomes a giant software infrastructure with the forces of company¹³¹⁴, school research team and countless talent engineers passion. It terminated the situation of everyone trying to re-invent wheels during 10 years ago. Extend your software from the re-usable source code is the right way. Of course you should consider an open source license if you are working with business. Actually anyone can contribute back to open source through the learning process. This book is written through the process of learning llvm backend and contribute back to llvm open source project. We think this book cannot exists in traditional paper book form since only few number of readers interested in study llvm backend even though there are many paper published books in concept of compiler. So, this book is published via electric media form and try to match the Open Document License Expection¹⁵. There are distance between the concept and the realistic program implemenation. Keep note through learning a large complicate software such as this llvm backend is not enough. We all learned the knowledge through books during school and after school. So, if you cannot find a good way to produce documents, you can consider to write documents like this book. This book document uses sphinx tool just like the llvm development team. Sphinx uses restructured text format here¹⁶¹⁷¹⁸. Appendix A of lbd book tell you how to install sphinx tool. Documentation work will help yourself to re-examine your software and make your program better in structure, reliability and more important "Extend your code to somewhere you didn't expect".

⁸ http://en.wikipedia.org/wiki/BSD_Documentation_License

⁹ <http://www.freebsd.org/docproj/>

¹⁰ <http://www.freebsd.org/copyright/freebsd-doc-license.html>

¹¹ http://en.wikipedia.org/wiki/GNU_Free_Documentation_License

¹² <http://www.gnu.org/copyleft/fdl.html>

¹³ <http://www.apple.com/opensource/>

¹⁴ <https://www.ibm.com/developerworks/opensource/>

¹⁵ <http://www.gnu.org/philosophy/free-doc.en.html>

¹⁶ <http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html>

¹⁷ <http://docutils.sourceforge.net/docs/ref/rst/directives.html>

¹⁸ <http://docutils.sourceforge.net/rst.html>

OPTIMIZATION

- *LLVM IR optimization*
- *Project*
 - *LLVM-VPO*

This chapter introduce llvm optimization.

4.1 LLVM IR optimization

The llvm-link provide optimizatoin in IR level which can apply in different programs developed by more than one language. Of course, it can apply in the same language which support seperate compile.

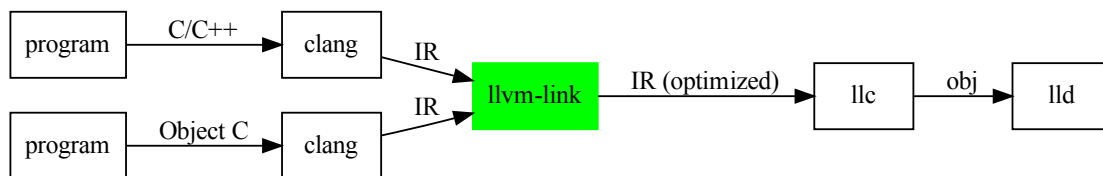


Fig. 4.1: llvm-link flow

Clang provide optimization options to do optimization from high level language to IR. But since many languages like C/C++ support separate compilation, it meaning there is no chance to do inter-procedure optimization if the functions come from different source files. To solve this problem, llvm provide **llvm-link** to link all *.bc into a single IR file, and through **opt** to finish the inter-procedure optimization¹. Beyond the DAG local optimization mentioned in Chapter 2, there are global optimization based on inter-procedure analysis². The following steps and examples show this optimization solution in llvm.

¹ <http://www.cs.cmu.edu/afs/cs/academic/class/15745-s12/public/lectures/L3-LLVM-Part1.pdf>

² Refer chapter 9 of book Compilers: Principles, Techniques, and Tools (2nd Edition)

exlbt/input/optimizen/1.cpp

```
int callee(const int *a) {  
    return *a+1;  
}
```

exlbt/input/optimize/2.cpp

```
extern int callee(const int *X);  
  
int caller() {  
    int T;  
  
    T = 4;  
  
    return callee(&T);  
}
```

```
JonathantekiiMac:input Jonathan$ clang -O3 -target mips-unknown-linux-gnu  
-c 1.cpp -emit-llvm -o 1.bc  
JonathantekiiMac:input Jonathan$ clang -O3 -target mips-unknown-linux-gnu  
-c 2.cpp -emit-llvm -o 2.bc  
JonathantekiiMac:input Jonathan$ llvm-link -o=a.bc 1.bc 2.bc  
JonathantekiiMac:input Jonathan$ opt -O3 -o=a1.bc a.bc  
JonathantekiiMac:input Jonathan$ llvm-dis a.bc -o -
```

```
...  
; Function Attrs: nounwind readonly  
define i32 @_Z6calleePKi(i32* nocapture readonly %a) #0 {  
    %1 = load i32* %a, align 4, !tbaa !1  
    %2 = add nsw i32 %1, 1  
    ret i32 %2  
}
```

```
define i32 @_Z6callerv() #1 {  
    %T = alloca i32, align 4  
    store i32 4, i32* %T, align 4, !tbaa !1  
    %1 = call i32 @_Z6calleePKi(i32* %T)  
    ret i32 %1  
}  
...
```

```
JonathantekiiMac:input Jonathan$ llvm-dis a1.bc -o -
```

```
...  
; Function Attrs: nounwind readonly  
define i32 @_Z6calleePKi(i32* nocapture readonly %a) #0 {  
    %1 = load i32* %a, align 4, !tbaa !1  
    %2 = add nsw i32 %1, 1  
    ret i32 %2  
}
```

(continues on next page)

(continued from previous page)

```
; Function Attrs: nounwind readnone
define i32 @_Z6callerv() #1 {
    ret i32 5
}
...
```

From the result as above, the **opt** output has lesser number of IR instructions. Of course, the backend code will be more effective as follows,

```
JonathantekiiMac:input Jonathan$ ~/llvm/test/build/
bin/llc -mach=cpu0 -relocation-model=pic -filetype=asm a.bc -o -
.section .mdebug.abi32
.previous
.file "a.bc"
.text
.globl      _Z6calleepKi
.align      2
.type _Z6calleepKi,@function
.ent _Z6calleepKi          # @_Z6calleepKi
_Z6calleepKi:
.frame      $sp,0,$lr
.mask       0x00000000,0
.set noreorder
.set nomacro
# BB#0:
ld    $2, 0($sp)
ld    $2, 0($2)
addiu $2, $2, 1
ret   $lr
.set macro
.set reorder
.end _Z6calleepKi
$tmp0:
.size _Z6calleepKi, ($tmp0)-_Z6calleepKi

.globl      _Z6callerv
.align      2
.type _Z6callerv,@function
.ent _Z6callerv          # @_Z6callerv
_Z6callerv:
.cfi_startproc
.frame      $sp,32,$lr
.mask       0x00004000,-4
.set noreorder
.cpload     $t9
.set nomacro
# BB#0:
addiu $sp, $sp, -32
$tmp3:
.cfi_def_cfa_offset 32
st    $lr, 28($sp)          # 4-byte Folded Spill
```

(continues on next page)

(continued from previous page)

```

$tmp4:
    .cfi_offset 14, -4
    .cpstore     8
    addiu $2, $zero, 4
    st     $2, 24($sp)
    addiu $2, $sp, 24
    st     $2, 0($sp)
    ld     $t9, %call16(_Z6calleePKi)($gp)
    jalr   $t9
    ld     $gp, 8($sp)
    ld     $lr, 28($sp)          # 4-byte Folded Reload
    addiu $sp, $sp, 32
    ret    $lr
    .set    macro
    .set    reorder
    .end    _Z6callerv

$tmp5:
    .size _Z6callerv, ($tmp5)-_Z6callerv
    .cfi_endproc

JonathantekiiMac:input Jonathan$ ~/llvm/test/build/
bin/llc -march=cpu0 -relocation-model=pic -filetype=asm a1.bc -o -
    .section .mdebug.abi32
    .previous
    .file "a1.bc"
    .text
    .globl     _Z6calleePKi
    .align     2
    .type _Z6calleePKi,@function
    .ent _Z6calleePKi          # @_Z6calleePKi
_Z6calleePKi:
    .frame      $sp,0,$lr
    .mask       0x00000000,0
    .set    noreorder
    .set    nomacro
# BB#0:
    ld     $2, 0($sp)
    ld     $2, 0($2)
    addiu $2, $2, 1
    ret    $lr
    .set    macro
    .set    reorder
    .end    _Z6calleePKi

$tmp0:
    .size _Z6calleePKi, ($tmp0)-_Z6calleePKi

    .globl     _Z6callerv
    .align     2
    .type _Z6callerv,@function
    .ent _Z6callerv          # @_Z6callerv
_Z6callerv:
    .frame      $sp,0,$lr

```

(continues on next page)

(continued from previous page)

```
.mask      0x00000000,0
.set  noreorder
.set  nomacro
# BB#0:
    addiu $2, $zero, 5
    ret   $lr
    .set  macro
    .set  reorder
    .end  _Z6callerv
$tmp1:
    .size _Z6callerv, ($tmp1)-_Z6callerv
```

Though llvm-link provide optimization in IR level to support separate compile, it come with the cost in compile time. As you can imagine, any one statement change will change the output IR of llvm-link. And the obj binary code have to re-compile. Compare to the separate compile for each *.c file, it only need to re-compile the corresponding *.o file only.

4.2 Project

4.2.1 LLVM-VPO

Friend Gang-Ryung Uh replace LLC compiler by llvm on Very Portable Optimizer (VPO) compiler toolchain. VPO performs optimizations on a single intermediate representation called Register Transfer Lists (RTLs). In other word, the system generate RTLs from llvm IR and it do further optimization on RTLs.

The LLVM-VPO is illustrated at his home page. Click “**6. LLVM-VPO Compiler Development - 2012 Google Faculty Research Award**” at this home page³ will get the information.

³ <http://cs.boisestate.edu/~uh/>

- *The theory of Floating point implementation*
- *The dependence for Cpu0 based on Compiler-rt's builtin*
- *C Library (Newlib)*
- *Compiler-rt's builtins*
 - *Verification*

5.1 The theory of Floating point implementation

Fixed point for representation of floating point as Fig. 5.1 and calculate as below after.



Fig. 5.1: Fixed point representation

Assume Sign part: 1-bit (0:+, 1:-), Integer part: 2-bit, Fraction part: 2-bit.

- $3.0 * 0.5 = \{0\ 11\ 00\} * \{0\ 00\ 10\} = \{(0\ \text{xor}\ 0)\ (11\ 00 * 00\ 10) \gg 2\} = \{0\ 01\ 10\} = 1.5$

The layout for half precision of Floating point as Fig. 5.2.

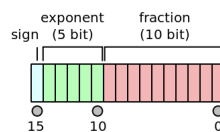


Fig. 5.2: IEEE 754 half precision of Floating point representation^{Page 67, 2}

Floating point can be implemented both on software and hardware.

The 16-bit $a*b$ can be calculated by transferring to Fixed point above for both a and b on more bits of memory/registers, calculate as Fixed point and transfer back to Fixed point as this website¹.

The example for multiplication based on exponent 2 as follows,

² https://en.wikipedia.org/wiki/Half-precision_floating-point_format

¹ <https://witscad.com/course/computer-architecture/chapter/floating-point-arithmetic>

- Precondition: a and b are normalized of IEEE half precision (16-bit)². Exponent bias: zero offset being 15: 15->0, 1-> -14, 30 -> 15. And 31 -> NaN.
- Transformation for a*b:
 - 1. {sign-bit(a) xor sign-bit(b)} {exponent-bits(a)+exponent-bits(b)-15} {significand-bits(a)*significand-bits(b) >> 10}
 - 2. Normalize:
- ex.
 - a = 0.01 (binary) = {0 01110 1000000000}; b = 1.1 (binary) = {010000 1100000000}
 - 1. a*b = {0 xor 0} {01110+10000-01111=01111} {1000000000*1100000000 >> 10 = 0110000000}
 - 2. Normalize: {0 01111 0110000000} -> {0 01110 1100000000} = 0.011

The following is for division operation.

- Transformation for a/b:
 - 1. {sign-bit(a) xor sign-bit(b)} {exponent-bits(a)-exponent-bits(b)+15} {significand-bits(a)/significand-bits(b) >> 10}
 - 2. Normalize:
- ex.
 - a = 0.01 (binary) = {0 01110 1000000000}; b = 1 (binary) = {0 10000 1000000000}
 - 1. a*b = {0 xor 0} {01110-10000+01111=01101} {1000000000/1000000000 << 9 = 1000000000}
 - 2. Normalize: {0 01101 0000000001} -> {0 01101 1000000000} = 0.01

IEEE-754 floating standard also consider NaN (Not a Number) such as 0/0 and ∞ as Fig. 5.3.

Exponent	Significand = zero	Significand \neq zero	Equation
00000 ₂	zero, -0	subnormal numbers	$(-1)^{\text{signbit}} \times 2^{-14} \times 0.\text{significandbits}_2$
00001 ₂ , ..., 11110 ₂	normalized value		$(-1)^{\text{signbit}} \times 2^{\text{exponent}-15} \times 1.\text{significandbits}_2$
11111 ₂	$\pm\text{infinity}$	NaN (quiet, signalling)	

Fig. 5.3: Encoding of exponent for IEEE 754 half precision [Page 67, 2](#)

Since Normalization applied in Floating precision is the critical code, Cpu0 HW provides clz and clo to speedup Normalization Operation.

Compiler-rt implement multiplication of Floating point considering NaN and ∞ as the same way for implementation above and clz/clo to speedup Normalization as follows,

~/llvm/debug/compiler-rt/lib/builtins/fp_lib.h

```
#if defined SINGLE_PRECISION
static __inline int rep_clz(rep_t a) { return __builtin_clz(a); }
...
#endif
...
static __inline rep_t toRep(fp_t x) {
  const union {
```

(continues on next page)

(continued from previous page)

```

    fp_t f;
    rep_t i;
} rep = {.f = x};
return rep.i;
}
...
static __inline int normalize(rep_t *significand) {
    const int shift = rep_clz(*significand) - rep_clz(implicitBit);
    *significand <<= shift;
    return 1 - shift;
}

```

~/llvm/debug/compiler-rt/lib/builtins/fp_mul_impl.inc

```

#include "fp_lib.h"
...
static __inline fp_t __mulXf3__(fp_t a, fp_t b) {
    const unsigned int aExponent = toRep(a) >> significandBits & maxExponent;
    const unsigned int bExponent = toRep(b) >> significandBits & maxExponent;
    ...
    int productExponent = aExponent + bExponent - exponentBias + scale;
    ...
    productHi |= (rep_t)productExponent << significandBits;
    ...
    return fromRep(productHi);
}

```

5.2 The dependence for Cpu0 based on Compiler-rt's builtin

Since Cpu0 has not hardware float point instructions, it needs software float point library to finish the floating point operation. LLVM compiler-rt project has software floating point library implementation [Fig. 5.4](#), so I choose it as the implementation.

Since compiler-rt uses unix/linux rootfs structure, we fill the gap by add few empty include-files in exlbt/include.

The dependences for compiler-rt on libm as [Fig. 5.6](#).

Table 5.1: lldb dependences

functions	depend on
scanf	newlib/libc
printf	sanitizer_printf.c of compiler-rt

Table 5.2: sanitizer_printf.c of compiler-rt dependences

functions	depend on
sanitizer_printf.c	builtins of compiler-rt

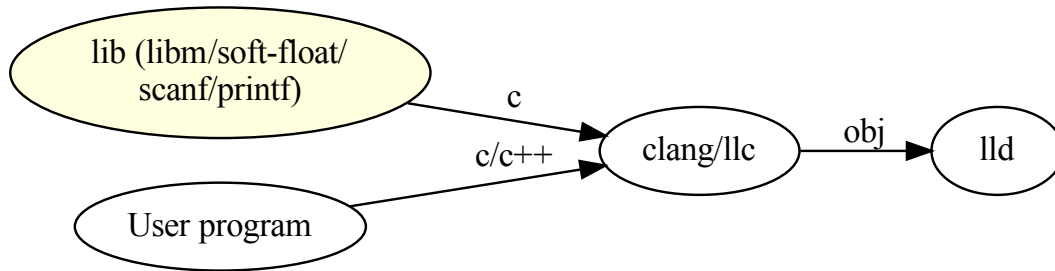


Fig. 5.4: compiler-rt/lib/builtins' software float library

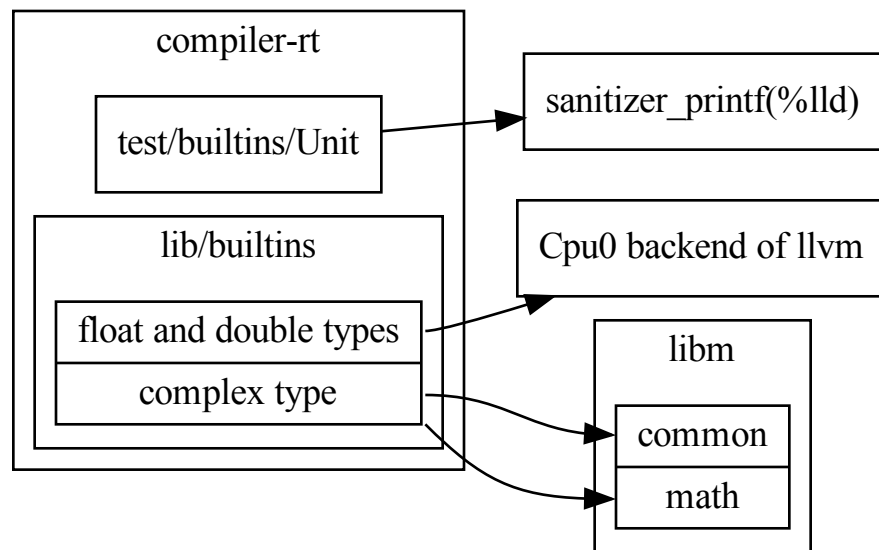


Fig. 5.5: Dependences for compiler-rt on libm

5.3 C Library (Newlib)

Since complex type of compiler-rt depends on libm, I porting NewLib in this section.

Newlib is a C library for bare metal platform. Two libraries in newlib are libc and libm. Libc is for functions of IO, file and string supported while libm is for mathematical functions. Web of newlib is here³ and newlib/libm here⁴. Since the next section compiler-rt/builtins depends on libm, please run the following bash script to install and build newlib for Cpu0.

lbt/exlbt/newlib-cpu0.sh

```
#!/usr/bin/env bash

# change this dir for newlib-cygwin
NEWLIB_PARENT_DIR=$HOME/git

NEWLIB_DIR=$NEWLIB_PARENT_DIR/newlib-cygwin
CURR_DIR=`pwd`
CC=$HOME/llvm/test/build/bin/clang
CFLAGS="-target cpu0el-unknown-linux-gnu -static -fintegrated-as -Wno-error=implicit-
↪function-declaration"
AS="$HOME/llvm/test/build/bin/clang -static -fintegrated-as -c"
AR="$HOME/llvm/test/build/bin/llvm-ar"
RANLIB="$HOME/llvm/test/build/bin/llvm-ranlib"
READELF="$HOME/llvm/test/build/bin/llvm-readelf"

install_newlib() {
  pushd $NEWLIB_PARENT_DIR
  git clone git://sourceware.org/git/newlib-cygwin.git
  cd newlib-cygwin
  git checkout dcb25665be227fb5a05497b7178a3d5df88050ec
  cp $CURR_DIR/newlib.patch .
  git apply newlib.patch
  cp -rf $CURR_DIR/newlib-cygwin/newlib/libc/machine/cpu0 newlib/libc/machine/.
  cp -rf $CURR_DIR/newlib-cygwin/libgloss/cpu0 libgloss/.
  popd
}

build_cpu0() {
  rm -rf build-$CPU-$ENDIAN
  mkdir build-$CPU-$ENDIAN
  cd build-$CPU-$ENDIAN
  CFLAGS="-target cpu0$ENDIAN-unknown-linux-gnu -mcpu=$CPU -static -fintegrated-as -Wno-
↪error=implicit-function-declaration"
  CC=$CC CFLAGS=$CFLAGS AS=$AS AR=$AR RANLIB=$RANLIB READELF=$READELF ../newlib/
↪configure --host=cpu0
  make
  cd ..
}
```

(continues on next page)

³ <https://sourceware.org/newlib/>

⁴ <https://sourceware.org/newlib/libm.html>

(continued from previous page)

```

build_newlib() {
    pushd $NEWLIB_DIR
    CPU=cpu032I
    ENDIAN=eb
    build_cpu0;
    CPU=cpu032I
    ENDIAN=el
    build_cpu0;
    CPU=cpu032II
    ENDIAN=eb
    build_cpu0;
    CPU=cpu032II
    ENDIAN=el
    build_cpu0;
    popd
}

install_newlib;
build_newlib;

```

Note: In order to add Cpu0 backend to NewLib, the following changes in `lbt/exlbt/newlib.patch`

- `lbt/exlbt/newlib-cygwin/newlib/libc/machine/cpu0/setjmp.S` is added;
- `newlib-cygwin/config.sub`, `newlib-cygwin/newlib/configure.host`, `newlib-cygwin/newlib/libc/include/machine/ieeefp.h`, `newlib-cygwin/newlib/libc/include/sys/unistd.h` and `newlib-cygwin/newlib/libc/machine/configure` are modified for adding `cpu0`.

`lbt/exlbt/newlib.patch`

```

diff --git a/config.sub b/config.sub
index 63c1f1c8b..575e8d9d7 100755
--- a/config.sub
+++ b/config.sub
@@ -1177,6 +1177,7 @@ case $cpu-$vendor in
        | d10v | d30v | dlx | dsp16xx \
        | e2k | elxsi | epiphany \
        | f30[01] | f700 | fido | fr30 | frv | ft32 | fx80 \
+       | cpu0 \
        | h8300 | h8500 \
        | hppa | hppa1.[01] | hppa2.0 | hppa2.0[nw] | hppa64 \
        | hexagon \
diff --git a/newlib/configure.host b/newlib/configure.host
index ca6b46f03..7bbf46f25 100644
--- a/newlib/configure.host
+++ b/newlib/configure.host
@@ -176,6 +176,10 @@ case "${host_cpu}" in
    fr30)
        machine_dir=fr30
        ;;

```

(continues on next page)

(continued from previous page)

```

-void      encrypt (char *__block, int __edflag);
+void      encrypt (char *__libc_block, int __edflag);
#endif
#if __BSD_VISIBLE || (__XSI_VISIBLE && __XSI_VISIBLE < 500)
void      endusershell (void);
diff --git a/newlib/libc/machine/configure b/newlib/libc/machine/configure
index 62064cdfd..5ef5eec08 100755
--- a/newlib/libc/machine/configure
+++ b/newlib/libc/machine/configure
@@ -823,6 +823,7 @@ csky
d10v
d30v
epiphany
+cpu0
fr30
frv
ft32
@@ -12007,6 +12008,8 @@ subdirs="$subdirs a29k"
d30v) subdirs="$subdirs d30v"

;;
epiphany) subdirs="$subdirs epiphany"
+ ;;
+ cpu0) subdirs="$subdirs cpu0"
;;
fr30) subdirs="$subdirs fr30"
;;

```

lbt/exlbt/newlib-cygwin/newlib/libc/machine/cpu0/setjmp.S

```

# setjmp/longjmp for cpu0. The jmpbuf looks like this:
#
# Register      jmpbuf offset
# $9            0x00
# $10           0x04
# $11           0x08
# $12           0x0c
# $13           0x10
# $14           0x14
# $15           0x18

.macro save reg
    st        \reg,@r4
    add       #4,r4
.endm

.macro restore reg
    ld        @r4,\reg
    add       #4,r4
.endm

```

(continues on next page)

(continued from previous page)

```

        .text
        .global      setjmp
setjmp:
    st $9,  0($a0)
    st $10, 4($a0)
    st $11, 8($a0)
    st $12, 12($a0)
    st $13, 16($a0)
    st $14, 20($a0)
    st $15, 24($a0)
# Return 0 to caller.
    addiu $lr, $zero, 0x0
    ret $lr

        .global      longjmp
longjmp:
    ld $9,  0($a0)
    ld $10, 4($a0)
    ld $11, 8($a0)
    ld $12, 12($a0)
    ld $13, 16($a0)
    ld $14, 20($a0)
    ld $15, 24($a0)

# If caller attempted to return 0, return 1 instead.
    cmp     $sw, $5,$0
    jne     $sw, $BB1
    addiu   $5,$0,1
$BB1:
    addu    $2,$0,$5
    ret     $lr

```

```
cschen@cschendeiMac exlbt % bash newlib-cpu0.sh
```

The libm.a depends on variable errno of libc defined in sys/errno.h.

- libgloss is BSP⁵

5.4 Compiler-rt's builtins

Compiler-rt is a project for runtime libraries implementation⁶. Compiler-rt/lib/builtins provides functions for basic operations such as +, -, *, /, ... on type of float or double and for conversion between float and integer, or other type of more than 32-bit, such as *long long*. The compiler-rt/lib/builtins/README.txt⁷ includes the dependent functions that the whole builtins called. The dependent functions is a small part of libm listed in compiler-rt/lib/builtins/int_math.h⁸.

⁵ <https://www.embecosm.com/appnotes/ean9/html/ch03s01.html>

⁶ <http://compiler-rt.llvm.org/>

⁷ <https://github.com/llvm-mirror/compiler-rt/blob/master/lib/builtins/README.txt>

⁸ https://github.com/microsoft/compiler-rt/blob/master/lib/builtins/int_math.h

~git/newlib-cygwin/build-cpu032l-eb/Makefile

```
MATHDIR = math
```

```
# The newlib hardware floating-point routines have been disabled due to
# inaccuracy. If you wish to work on them, you will need to edit the
# configure.in file to re-enable the configuration option. By default,
# the NEWLIB_HW_FP variable will always be false.
#MATHDIR = mathfp
```

As above Makefile, newlib uses libm/math. The dependences for builtin functions of compiler-rt on libm as Fig. 5.6.

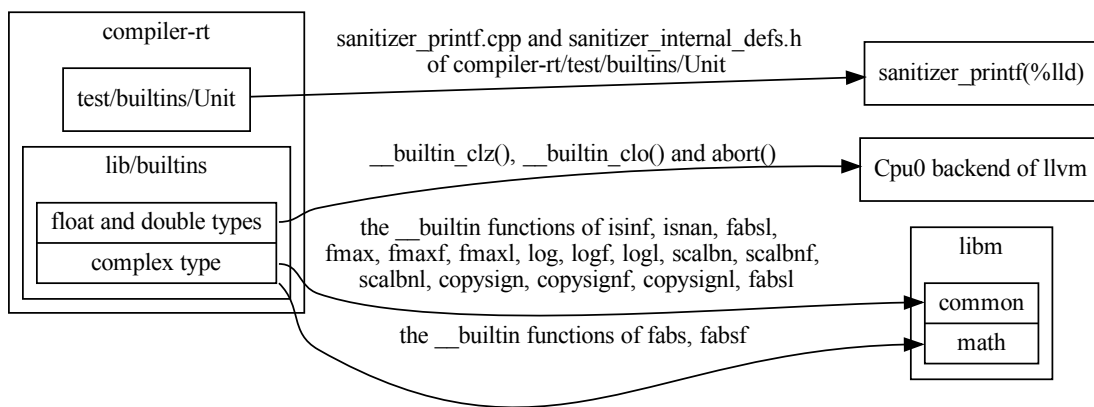


Fig. 5.6: Dependences for builtin functions of compiler-rt on libm

In this section, I get test cases for verification of SW Float Point from compiler-rt/test/builtins/Unit to compiler-rt-test/builtins/Unit/.

Since lbt/exlbt/input/printf-stdarg.c does not support %lld (long long integer, 64-bit) and test cases in compiler-rt/test/builtins/Unit needs it to verify the result of test cases for SW Float Point, I port sanitizer_printf.cpp and sanitizer_internal_defs.h of lbt/exlbt/input from sanitizer_printf.cpp and sanitizer_internal_defs.h of compiler-rt/lib/sanitizer_common.

Table 5.3: compiler-rt builtins dependences on newlib/libm (open source libc for bare metal)

function	file	directory of libm
abort	lbt/exlbt/compiler-rt/ cpu0/abort.c	
isinf	s_isinf.c	newlib-cygwin/newlib/libm/common
isnan	s_isnan.c	“
fabsl	fabsl.c	“
fmax	s_fmax.c	“
fmaxf	sf_fmax.c	“
fmaxl	fmaxl.c	“
log	log.c	“
logf	sf_log.c	“
logl	logl.c	“
scalbn	s_scalbn.c	“
scalbnf	sf_scalbn.c	“
scalbnl	scalbnl.c	“
copysign	s_copysign.c	“
copysignf	sf_copysign.c	“
copysignl	copysignl.c	“
fabsl	fabsl.c	“
fabs	s_fabs.c	newlib-cygwin/newlib/libm/math
fabsf	sf_fabs.c	“

- Libm has no dependence to any other library.
- Only type of complex in compiler-rt/lib/builtin need above, others (float and double) depend on `__builtin_clz()`, `__builtin_clo()` and `abort()` only. I has ported in `lbt/exlbt/compiler-rt/cpu0/abort.c`.
- All test cases in `compiler-rt/test/builtins/Unit` depend on `printf(%lld or %lX, ...)`, I ported from `compiler-rt/lib/sanitizer_common/sanitizer_printf.cpp` to `lbt/exlbt/input/sanitizer_printf.cpp`.
- These dependent functions of complex type has bee ported from newlib/libm.
- Except builtins, the other three, sanitizer runtimes, profile and BlocksRuntime, in compiler-rt are not needed for my embedded Cpu0.

The libgcc's Integer plus Soft float library⁹¹⁰¹¹ are equal to functions of compiler-rt's builtins.

In compiler-rt/lib/builtins, the dependence between files as table.

Table 5.4: dependence between files for compiler-rt/lib/builtins

functions	depend on
*.c	*.inc
*.inc	*.h

Though the 'rt' means RunTime libaraies, in builtins library, most of these functions written in target-independent C form and can be compiled and static-linked into target. When you compile the following c code, llc will generate **jsub __addsf3** to call compiler-rt float function for Cpu0. This is because Cpu0 hasn't hardware float-instructions, so Cpu0 backend doesn't handle the DAG of `__addsf3`. The end result, llvm treats the DAG of `__addsf3` as a function call for float-add instruction.

⁹ <https://gcc.gnu.org/onlinedocs/gccint/Libgcc.html>

¹⁰ <https://gcc.gnu.org/onlinedocs/gccint/Integer-library-routines.html#Integer-library-routines>

¹¹ <https://gcc.gnu.org/onlinedocs/gccint/Soft-float-library-routines.html#Soft-float-library-routines>

lbt/exlbt/input/ch_call_compilerrt_func.c

```
// clang -target mips-unknown-linux-gnu -S ch_call_compilerrt_func.c -emit-llvm -o ch_
↳ call_compilerrt_func.ll
// ~/llvm/test/build/bin/llc -march=cpu0 -mcpu=cpu032II -relocation-model=static -
↳ filetype=asm ch_call_compilerrt_func.ll -o -

/// start
float ch_call_compilerrt_func()
{
    float a = 3.1;
    float b = 2.2;
    float c = a + b;

    return c;
}
```

```
chungshu@ChungShudeMacBook-Air input % clang -target mips-unknown-linux-gnu -S
ch_call_compilerrt_func.c -emit-llvm
chungshu@ChungShudeMacBook-Air input % cat ch_call_compilerrt_func.ll

...
%4 = load float, float* %1, align 4
%5 = load float, float* %2, align 4
%6 = fadd float %4, %5

chungshu@ChungShudeMacBook-Air input % ~/llvm/test/build/bin/llc -march=cpu0
-mcpu=cpu032II -relocation-model=static -filetype=asm ch_call_compilerrt_func.ll -o -

...
ld      $4, 20($fp)
ld      $5, 16($fp)
jsub    __addsf3
```

For some bare-metal or embedded application, the C code doesn't need the file and high-level IO in libc. Libm provides a lots of functions to support software floating point beyond basic operations¹². Libc provides file, high-level IO functions and basic float functions¹³.

Cpu0 hires Compiler-rt/lib/builtins and compiler-rt/lib/sanitizer_common/sanitizer_printf.cpp at this point. The compiler-rt/lib/builtins is a target-independent C form of software float library implementation. Cpu0 implements compiler-rt-12.x/cpu0/abort.c only at this point for supporting this feature.

Note: Why these libm functions called builtins in compiler-rt/lib/builtins?

Though these compiler-rt builtins functions are written in C. The CPU can provide float type instructions or high level instructions to compile these libm function calls into specific HW instructions to speed up.

In order to speed up these libm functions, many CPU provide float instructions for them. Of course, for the implementation, clang compiles these float type's operation in C into llvm ir, then Mips backends compiles them into their HW instructions. For example:

- float a, b, c; a=b*c; -> (clang) -> %add = fmul float %0, %1¹⁵

¹² <https://www.programiz.com/c-programming/library-function/math.h>

¹³ <https://www.cplusplus.com/reference/clibrary>

¹⁵ Reference https://github.com/Jonathan2251/lbd/tree/master/lbdex/input/ch7_1_fmulo.c

Mips backend compiles *fmul* into HW instructions as follows,

- `%cadd = fmul float %0, %1 -> (llvm-mips) -> mul.s`^{Page 78, 1514}

Cpu0 backend compiles *fmul* into libm function call *fmul* as follows,

- `%cadd = fmul float %0, %1 -> (llvm-mips) -> jsb fmul`^{Page 78, 15}

For high level of math functions, clang compiles these float type's operation in C into llvm intrinsic functions, then the llvm backends of these CPU compile them into their HW instructions. For example, clang compiles *pow()* into `@llvm.pow.f32` as follows,

- `%cpow = call float @llvm.pow.f32(float %cx, float %cy)`¹⁶

AMDGPU compiles `@llvm.pow.f32` into a few instructions as follows:

- `%cpow = call float @llvm.pow.f32(float %cx, float %cy) (llvm-AMDGPU) -> ... + v_exp_f32_e32 v0, v0 + ...`¹⁶

Mips compiles `@llvm.pow.f32` into a few instructions as follows:

- `%cpow = call float @llvm.pow.f32(float %cx, float %cy) (llvm-AMDGPU) -> jal powf`¹⁶

Clang treats these libm functions as builtin and compiles them into llvm ir or intrinsic, then different backends can choose to compile them into specific instructions or call builtin functions in libm. The following is Clang's comment¹⁷.

```
RValue CodeGenFunction::EmitBuiltinExpr(...)
...
// There are LLVM math intrinsics/instructions corresponding to math library
// functions except the LLVM op will never set errno while the math library
// might. Also, math builtins have the same semantics as their math library
// twins. Thus, we can transform math library and builtin calls to their
// LLVM counterparts if the call is marked 'const' (known to never set errno).
```

5.4.1 Verification

The following sanitizer_printf.cpp extended from compiler-rt can support printf("%lld"). It's implementation calling some floating lib functions in compiler-rt/lib/builtins.

exlbt/include/math.h

```
#ifndef _MATH_H_
#define _MATH_H_

// #ifdef HAS_COMPLEX
// #ifndef HUGE_VALF
// #define HUGE_VALF (1.0e999999999F)
// #endif

// #if !defined(INFINITY)
// #define INFINITY (HUGE_VALF)
// #endif

// #if !defined(NAN)
```

(continues on next page)

¹⁴ <https://github.com/llvm/llvm-project/blob/main/llvm/test/CodeGen/Mips/fmadd1.ll>

¹⁶ Reference https://github.com/Jonathan2251/lbt/tree/master/exlbt/input/test_pow.c

¹⁷ <https://github.com/llvm/llvm-project/blob/main/clang/lib/CodeGen/CGBuiltin.cpp>

(continued from previous page)

```
#define NAN (0.0F/0.0F)
#endif

float cabsf(float complex) ;
//#endif
#endif
```

exlbt/include/stdio.h

```
#ifndef _STDIO_H_
#define _STDIO_H_

#define stdin 0
#define stdout 1
#define stderr 2

#define size_t unsigned int

#endif
```

exlbt/include/stdlib.h

```
#ifndef _STDLIB_H_
#define _STDLIB_H_

#ifdef __cplusplus
extern "C" {
#endif

void abort();

#ifdef __cplusplus
}
#endif

#endif
```

exlbt/include/string.h

```
#ifndef _STRING_H_
#define _STRING_H_

#endif
```

exlbt/compiler-rt/cpu0/abort.c

```
void abort() {
    // cpu0.v: ABORT at mem 0x04
    asm("addiu $lr, $ZERO, 4");
    asm("ret $lr");
}
```

exlbt/input/sanitizer_internal_defs.h

```
//====-- sanitizer_internal_defs.h -----*- C++ -*====//
//
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
// See https://llvm.org/LICENSE.txt for license information.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
//
//====-
//
// This file is shared between AddressSanitizer and ThreadSanitizer.
// It contains macro used in run-time libraries code.
//====-
#ifdef SANITIZER_DEFS_H
#define SANITIZER_DEFS_H

// For portability reasons we do not include stddef.h, stdint.h or any other
// system header, but we do need some basic types that are not defined
// in a portable way by the language itself.
namespace __sanitizer {

#ifdef _WIN64
// 64-bit Windows uses LLP64 data model.
typedef unsigned long long uptr;
typedef signed long long sptr;
#else
typedef unsigned long uptr;
typedef signed long sptr;
#endif // defined(_WIN64)
#ifdef __x86_64__
// Since x32 uses ILP32 data model in 64-bit hardware mode, we must use
// 64-bit pointer to unwind stack frame.
typedef unsigned long long uhwptr;
#else
typedef uptr uhwptr;
#endif

typedef unsigned char u8;
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned long long u64;
typedef signed char s8;
typedef signed short s16;
```

(continues on next page)

(continued from previous page)

```

typedef signed int s32;
typedef signed long long s64;

// Check macro
#define RAW_CHECK_MSG(expr, msg)

#define RAW_CHECK(expr) RAW_CHECK_MSG(expr, #expr)

#define CHECK_IMPL(c1, op, c2)

#define CHECK(a) CHECK_IMPL((a), !=, 0)
#define CHECK_EQ(a, b) CHECK_IMPL((a), ==, (b))
#define CHECK_NE(a, b) CHECK_IMPL((a), !=, (b))
#define CHECK_LT(a, b) CHECK_IMPL((a), <, (b))
#define CHECK_LE(a, b) CHECK_IMPL((a), <=, (b))
#define CHECK_GT(a, b) CHECK_IMPL((a), >, (b))
#define CHECK_GE(a, b) CHECK_IMPL((a), >=, (b))

} // namespace __sanitizer

#endif

```

exlbt/input/sanitizer_printf.cpp

```

//====-- sanitizer_printf.cpp -----===//
//
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
// See https://llvm.org/LICENSE.txt for license information.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
//
//====-----===//
//
// This file is shared between AddressSanitizer and ThreadSanitizer.
//
// Internal printf function, used inside run-time libraries.
// We can't use libc printf because we intercept some of the functions used
// inside it.
//====-----===//

#include "sanitizer_internal_defs.h"

#include <stdio.h>
#include <stdarg.h>

#include "debug.h"

extern "C" int putchar(int c);

extern void* internal_memset(void* b, int c, size_t len);

```

(continues on next page)

(continued from previous page)

```

#if SANITIZER_WINDOWS && defined(_MSC_VER) && _MSC_VER < 1800 && \
    !defined(va_copy)
# define va_copy(dst, src) ((dst) = (src))
#endif

namespace __sanitizer {

static int strlen(const char* s) {
    int len = 0;
    for (const char* p = s; *p != '\0'; p++) {
        len++;
    }
    return len;
}

static int AppendChar(char **buff, const char *buff_end, char c) {
    if (*buff < buff_end) {
        **buff = c;
        (*buff)++;
    }
    return 1;
}

// Appends number in a given base to buffer. If its length is less than
// |minimal_num_length|, it is padded with leading zeroes or spaces, depending
// on the value of |pad_with_zero|.
static int AppendNumber(char **buff, const char *buff_end, u64 absolute_value,
                        u8 base, u8 minimal_num_length, bool pad_with_zero,
                        bool negative, bool uppercase, bool left_justified) {
    uptr const kMaxLen = 30;
    RAW_CHECK(base == 10 || base == 16);
    RAW_CHECK(base == 10 || !negative);
    RAW_CHECK(absolute_value || !negative);
    RAW_CHECK(minimal_num_length < kMaxLen);
    int result = 0;
    if (negative && minimal_num_length)
        --minimal_num_length;
    if (negative && pad_with_zero)
        result += AppendChar(buff, buff_end, '-');
    uptr num_buffer[kMaxLen];
    int num_pads = 0;
    int pos = 0;
    do {
        RAW_CHECK_MSG((uptr)pos < kMaxLen, "AppendNumber buffer overflow");
        num_buffer[pos++] = absolute_value % base;
        absolute_value /= base;
    } while (absolute_value > 0);
    if (pos < minimal_num_length) {
        // Make sure compiler doesn't insert call to memset here.
        internal_memset(&num_buffer[pos], 0,
                       sizeof(num_buffer[0]) * (minimal_num_length - pos));
        num_pads = minimal_num_length - pos;
    }
}

```

(continues on next page)

(continued from previous page)

```

    pos = minimal_num_length;
}
RAW_CHECK(pos > 0);
pos--;
for (; pos >= 0 && num_buffer[pos] == 0; pos--) {
    char c = (pad_with_zero || pos == 0) ? '0' : ' ';
    if (!left_justified)
        result += AppendChar(buff, buff_end, c);
}
if (negative && !pad_with_zero) result += AppendChar(buff, buff_end, '-');
for (; pos >= 0; pos--) {
    char digit = static_cast<char>(num_buffer[pos]);
    digit = (digit < 10) ? '0' + digit : (uppercase ? 'A' : 'a') + digit - 10;
    result += AppendChar(buff, buff_end, digit);
}
if (left_justified) {
    for (int i = 0; i < num_pads; i++)
        result += AppendChar(buff, buff_end, ' ');
}
return result;
}

static int AppendUnsigned(char **buff, const char *buff_end, u64 num, u8 base,
                          u8 minimal_num_length, bool pad_with_zero,
                          bool uppercase, bool left_justified) {
    return AppendNumber(buff, buff_end, num, base, minimal_num_length,
                        pad_with_zero, false /* negative */, uppercase,
                        left_justified);
}

static int AppendSignedDecimal(char **buff, const char *buff_end, s64 num,
                               u8 minimal_num_length, bool pad_with_zero,
                               bool left_justified) {
    bool negative = (num < 0);
    return AppendNumber(buff, buff_end, (u64)(negative ? -num : num), 10,
                        minimal_num_length, pad_with_zero, negative,
                        false /* uppercase */, left_justified);
}

// Use the fact that explicitly requesting 0 width (%0s) results in UB and
// interpret width == 0 as "no width requested":
// width == 0 - no width requested
// width < 0 - left-justify s within and pad it to -width chars, if necessary
// width > 0 - right-justify s, implement for cpu0
static int AppendString(char **buff, const char *buff_end, int width,
                       int max_chars, const char *s, bool left_justified) {
    if (!s)
        s = "<null>";
    int result = 0;
    if (!left_justified) {
        int s_len = strlen(s);

```

(continues on next page)

(continued from previous page)

```

    while (result < width - s_len)
        result += AppendChar(buff, buff_end, ' ');
}
for (; *s; s++) {
    if (max_chars >= 0 && result >= max_chars)
        break;
    result += AppendChar(buff, buff_end, *s);
}
if (left_justified) {
    while (width < -result)
        result += AppendChar(buff, buff_end, ' ');
}
return result;
}

static int AppendPointer(char **buff, const char *buff_end, u64 ptr_value,
                        bool left_justified) {
    int result = 0;
    result += AppendString(buff, buff_end, 0, -1, "0x", left_justified);
    result += AppendUnsigned(buff, buff_end, ptr_value, 16,
// By running clang -E, can get the macro value for SANITIZER_POINTER_FORMAT_LENGTH is_
↪(12)
//
                        SANITIZER_POINTER_FORMAT_LENGTH,
                        (12),
                        true /* pad_with_zero */, false /* uppercase */,
                        left_justified);
    return result;
}

int VSNPrintf(char *buff, int buff_length,
              const char *format, va_list args) {
    static const char *kPrintfFormatsHelp =
        "Supported Printf formats: %([0-9]*)?(z|ll)?{d,u,x,X}; %p; "
        "%[-]([0-9]*)?(\\.\\|\\|*)?s; %c\\n";
    RAW_CHECK(format);
    RAW_CHECK(buff_length > 0);
    const char *buff_end = &buff[buff_length - 1];
    const char *cur = format;
    int result = 0;
    for (; *cur; cur++) {
        if (*cur != '%') {
            result += AppendChar(&buff, buff_end, *cur);
            continue;
        }
        cur++;
        bool left_justified = *cur == '-';
        if (left_justified)
            cur++;
        bool have_width = (*cur >= '0' && *cur <= '9');
        bool pad_with_zero = (*cur == '0');
        int width = 0;
        if (have_width) {

```

(continues on next page)

(continued from previous page)

```

    while (*cur >= '0' && *cur <= '9') {
        width = width * 10 + *cur++ - '0';
    }
}
bool have_precision = (cur[0] == '.' && cur[1] == '*');
int precision = -1;
if (have_precision) {
    cur += 2;
    precision = va_arg(args, int);
}
bool have_z = (*cur == 'z');
cur += have_z;
bool have_ll = !have_z && (cur[0] == 'l' && cur[1] == 'l');
cur += have_ll * 2;
s64 dval;
u64 uval;
const bool have_length = have_z || have_ll;
const bool have_flags = have_width || have_length;
// At the moment only %s supports precision and left-justification.
CHECK(!((precision >= 0 || left_justified) && *cur != 's'));
switch (*cur) {
    case 'd': {
        dval = have_ll ? va_arg(args, s64)
            : have_z ? va_arg(args, sptr)
            : va_arg(args, int);
        result += AppendSignedDecimal(&buff, buff_end, dval, width,
            pad_with_zero, left_justified);

        break;
    }
    case 'u':
    case 'x':
    case 'X': {
        uval = have_ll ? va_arg(args, u64)
            : have_z ? va_arg(args, uptr)
            : va_arg(args, unsigned);
        bool uppercase = (*cur == 'X');
        result += AppendUnsigned(&buff, buff_end, uval, (*cur == 'u') ? 10 : 16,
            width, pad_with_zero, uppercase, left_justified);

        break;
    }
    case 'p': {
        RAW_CHECK_MSG(!have_flags, kPrintfFormatsHelp);
        result += AppendPointer(&buff, buff_end, va_arg(args, uptr),
            left_justified);

        break;
    }
    case 's': {
        RAW_CHECK_MSG(!have_length, kPrintfFormatsHelp);
        CHECK(!have_width || left_justified);
        result += AppendString(&buff, buff_end, left_justified ? -width : width,
            precision, va_arg(args, char*), left_justified);

        break;
    }
}

```

(continues on next page)

(continued from previous page)

```

    }
    case 'c': {
        RAW_CHECK_MSG(!have_flags, kPrintfFormatsHelp);
        result += AppendChar(&buff, buff_end, va_arg(args, int));
        break;
    }
    case '%' : {
        RAW_CHECK_MSG(!have_flags, kPrintfFormatsHelp);
        result += AppendChar(&buff, buff_end, '%');
        break;
    }
    default: {
        RAW_CHECK_MSG(false, kPrintfFormatsHelp);
    }
}
}
RAW_CHECK(buff <= buff_end);
AppendChar(&buff, buff_end + 1, '\\0');
return result;
}

} // namespace __sanitizer

int prints(const char *string)
{
    int pc = 0, padchar = ' ';

    for ( ; *string ; ++string) {
        putchar (*string);
        ++pc;
    }

    return pc;
}

extern "C" int sprintf(char *buffer, const char *format, ...) {
    int length = 1000;
    va_list args;
    va_start(args, format);
    int needed_length = __sanitizer::VSNPrintf(buffer, length, format, args);
    va_end(args);
    return 0;
}

extern "C" int printf(const char *format, ...) {
    int length = 1000;
    char buffer[1000];
    va_list args;
    va_start(args, format);
    int needed_length = __sanitizer::VSNPrintf(buffer, length, format, args);
    va_end(args);
    prints(buffer);
}

```

(continues on next page)

(continued from previous page)

```

    return 0;
}

extern "C" int san_printf(const char *format, ...) {
    int length = 1000;
    char buffer[1000];
    va_list args;
    va_start(args, format);
    int needed_length = __sanitizer::VSNPrintf(buffer, length, format, args);
    va_end(args);
    prints(buffer);
    return 0;
}

```

Above two sanitizer_*.c files are ported from compiler-rt and I add code to support left-justify for number-printf and right-justify for string-printf. The following ch_float.cpp test the float lib.

lbt/exlbt/compiler-rt-12.x/builtins/Makefile

```

# Thanks .c .cb Vranish (https://spin.a.cmi.cjobject..cm/2016/08/26/makefile-c-p.cjects/)

# CPU and endian passed from command line, such as "make CPU=cpu032II ENDIAN=el"

TARGET_LIB := libbuiltins.a
BUILD_DIR := ./build-$(CPU)-$(ENDIAN)
TARGET := $(BUILD_DIR)/$(TARGET_LIB)

SRC_DIR := $(HOME)/llvm/llvm-project/compiler-rt/lib/builtins

PWD := $(shell pwd)

TOOLDIR := ~/llvm/test/build/bin
CC := $(TOOLDIR)/clang
AR := $(TOOLDIR)/llvm-ar

# copy GENERIC_SOURCES from compiler-rt/lib/builtin/CMakeLists.txt
GENERIC_SOURCES := \
    absvdi2.c \
    absvsi2.c \
    absvti2.c \
    adddf3.c \
    addsf3.c \
    addvdi3.c \
    addvsi3.c \
    addvti3.c \
    apple_versioning.c \
    ashldi3.c \
    ashlti3.c \
    ashrdi3.c \
    ashrti3.c \
    bswapdi2.c \

```

(continues on next page)

(continued from previous page)

```
bswapsi2.c \  
clzdi2.c \  
clzsi2.c \  
clzti2.c \  
cmpdi2.c \  
cmpti2.c \  
comparedf2.c \  
comparesf2.c \  
ctzdi2.c \  
ctzsi2.c \  
ctzti2.c \  
divdc3.c \  
divdf3.c \  
divdi3.c \  
divsc3.c \  
divtc3.c \  
divmoddi4.c \  
divmodsi4.c \  
divmodti4.c \  
divsc3.c \  
divsf3.c \  
divsi3.c \  
divti3.c \  
divxc3.c \  
extendsfdf2.c \  
extendhfsf2.c \  
ffsdi2.c \  
ffssi2.c \  
ffsti2.c \  
fixdfdi.c \  
fixdfsi.c \  
fixdfti.c \  
fixsfdi.c \  
fixsfsi.c \  
fixsfti.c \  
fixunsdfdi.c \  
fixunsdfsi.c \  
fixunsdfti.c \  
fixunssfdi.c \  
fixunssfsi.c \  
fixunssfti.c \  
floatdidf.c \  
floatdisf.c \  
floatsidf.c \  
floatsisf.c \  
floattidf.c \  
floattisf.c \  
floatundidf.c \  
floatundisf.c \  
floatunsidf.c \  
floatunsisf.c \  
floatuntidf.c \  

```

(continues on next page)

(continued from previous page)

```
floatuntisf.c \  
fp_mode.c \  
int_util.c \  
lshrdi3.c \  
lshrti3.c \  
moddi3.c \  
modsi3.c \  
modti3.c \  
muldc3.c \  
muldf3.c \  
muldi3.c \  
mulodi4.c \  
mulosi4.c \  
muloti4.c \  
mulsc3.c \  
mulsf3.c \  
multi3.c \  
mulvdi3.c \  
mulvsi3.c \  
mulvti3.c \  
mulxc3.c \  
negdf2.c \  
negdi2.c \  
negsf2.c \  
negti2.c \  
negvdi2.c \  
negvsi2.c \  
negvti2.c \  
os_version_check.c \  
paritydi2.c \  
paritysi2.c \  
parityti2.c \  
popcountdi2.c \  
popcountsi2.c \  
popcountti2.c \  
powidf2.c \  
powisf2.c \  
subdf3.c \  
subsf3.c \  
subvdi3.c \  
subvsi3.c \  
subvti3.c \  
trampoline_setup.c \  
truncdfhf2.c \  
truncdfs2.c \  
truncsfhf2.c \  
ucmpdi2.c \  
ucmpti2.c \  
udivdi3.c \  
udivmoddi4.c \  
udivmodsi4.c \  
udivmodti4.c \  

```

(continues on next page)

(continued from previous page)

```

    udivsi3.c \
    udivti3.c \
    umoddi3.c \
    umodsi3.c \
    umodti3.c

SRCS := $(GENERIC_SOURCES)

# String substitution for every C file.
# As an example, absvdi2.c turns into ./builtins/absvdi2.c
SRCS := $(SRCS:%=$(SRC_DIR)/%) $(PWD)/../cpu0/abort.c

# String substitution for every C/C++ file.
# As an example, absvdi2.c turns into ./build-$(CPU)-$(ENDIAN)/absvdi2.c.o
OBS := $(SRCS:%=$(BUILD_DIR)/%.o)

# String substitution (suffix version without %).
# As an example, ./build/absvdi2.c.o turns into ./build-$(CPU)-$(ENDIAN)/absvdi2.c.d
DEPS := $(OBS:.o=.d)

# Every folder in ./src will need to be passed to GCC so that it can find header files
# stdlib.h, ..., etc existed in ../../include
INC_DIRS := $(shell find $(SRC_DIR) -type d) ../../include
# Add a prefix to INC_DIRS. So moduleA would become -I moduleA. GCC understands this -I_
→flag
INC_FLAGS := $(addprefix -I,$(INC_DIRS))

# The -MMD and -MP flags together generate Makefiles for us!
# These files will have .d instead of .o as the output.
CPPFLAGS := -MMD -MP -target cpu0$(ENDIAN)-unknown-linux-gnu -static \
    -fintegrated-as $(INC_FLAGS) -mcpu=$(CPU) -mllvm -has-llc=true

# The final build step.
$(TARGET): $(OBS)
    $(AR) -rcs $@ $(OBS)

# Build step for C source
$(BUILD_DIR)/%.c.o: %.c
    mkdir -p $(dir $@)
    $(CC) $(CPPFLAGS) $(CFLAGS) -c $< -o $@

.PHONY: clean
clean:
    rm -rf $(BUILD_DIR)

# Include the .d makefiles. The - at the f.cnt suppresses the er.crs.cf missing
# Makefiles. Initially, all the .d files will be missing, and we .cn't want t.cse
# er.crs .c s.cw up.
-include $(DEPS)

```

exlbt/input/ch_float.cpp

```

//#include "debug.h"

extern "C" int printf(const char *format, ...);
extern "C" int sprintf(char *out, const char *format, ...);

#include "ch9_3_longlongshift.cpp"

void test_printf()
{
    char buf[80];
    long long a = 0x1000000007ffffff;
    printf("a: %llX, %llx, %lld\n", a, a, a);
    int b = 0x100000000;
    printf("b: %x, %d\n", b, b);
    sprintf(buf, "b: %x, %d\n", b, b); printf("%s", buf);

    // sanitizer_printf.cpp support right-justify for num only and left-justify
    // for string only. However, I change and support right-justify for cpu0.
    char ptr[] = "Hello world!";
    char *np = 0;
    int i = 5;
    unsigned int bs = sizeof(int)*8;
    int mi;

    mi = (1 << (bs-1)) + 1;
    printf("%s\n", ptr);
    printf("printf test\n");
    printf("%s is null pointer\n", np);
    printf("%d = 5\n", i);
    printf("%d = - max int\n", mi);
    printf("char %c = 'a'\n", 'a');
    printf("hex %x = ff\n", 0xff);
    printf("hex %02x = 00\n", 0);
    printf("signed %d = unsigned %u = hex %x\n", -3, -3, -3);
    printf("%d %s(s)", 0, "message");
    printf("\n");
    printf("%d %s(s) with %%\n", 0, "message");
    sprintf(buf, "justif: \"%-10s\"\n", "left"); printf("%s", buf);
    sprintf(buf, "justif: \"%10s\"\n", "right"); printf("%s", buf);
    sprintf(buf, " 3: %04d zero padded\n", 3); printf("%s", buf);
    sprintf(buf, " 3: %-4d left justif.\n", 3); printf("%s", buf);
    sprintf(buf, " 3: %4d right justif.\n", 3); printf("%s", buf);
    sprintf(buf, "-3: %04d zero padded\n", -3); printf("%s", buf);
    sprintf(buf, "-3: %-4d left justif.\n", -3); printf("%s", buf);
    sprintf(buf, "-3: %4d right justif.\n", -3); printf("%s", buf);
}

template <class T>
T test_shift_left(T a, T b) {
    return (a << b);
}

```

(continues on next page)

(continued from previous page)

```

template <class T>
T test_shift_right(T a, T b) {
    return (a >> b);
}

template <class T1, class T2, class T3>
T1 test_add(T2 a, T3 b) {
    T1 c = a + b;
    return c;
}

template <class T1, class T2, class T3>
T1 test_mul(T2 a, T3 b) {
    T1 c = a * b;
    return c;
}

template <class T1, class T2, class T3>
T1 test_div(T2 a, T3 b) {
    T1 c = a / b;
    return c;
}

bool check_result(const char* fn, long long res, long long expected) {
    printf("%s = %lld\n", fn, res);
    if (res != expected) {
        printf("\terror: result %lld, expected %lld\n", res, expected);
    }
    return (res == expected);
}

bool check_result(const char* fn, unsigned long long res, unsigned long long expected) {
    printf("%s = %llu\n", fn, res);
    if (res != expected) {
        printf("\terror: result %llu, expected %llu\n", res, expected);
    }
    return (res == expected);
}

bool check_result(const char* fn, int res, int expected) {
    printf("%s = %d\n", fn, res);
    if (res != expected) {
        printf("\terror: result %d, expected %d\n", res, expected);
    }
    return (res == expected);
}

int main() {
    long long a;
    unsigned long long b;
    int c;

```

(continues on next page)

(continued from previous page)

```

test_printf();

a = test_longlong_shift1();
check_result("test_longlong_shift1()", a, 289LL);

a = test_longlong_shift2();
check_result("test_longlong_shift2()", a, 22LL);

// call __ashldi3
a = test_shift_left<long long>(0x12LL, 4LL); // 0x120 = 288
check_result("test_shift_left<long long>(0x12LL, 4LL)", a, 288LL);

// call __ashrdi3
a = test_shift_right<long long>(0x0016666600000000a, 48LL); // 0x16 = 22
check_result("test_shift_right<long long>(0x0016666600000000a, 48LL)", a, 22LL);

// call __lshrdi3
b = test_shift_right<unsigned long long>(0x0016666600000000a, 48LLu); // 0x16 = 22
check_result("test_shift_right<unsigned long long>(0x0016666600000000a, 48LLu)", b,
↳ 22LLu);

// call __addsf3, __fixsfsi
c = (int)test_add<float, float, float>(-2.2, 3.3); // (int)1.1 = 1
check_result("(int)test_add<float, float, float>(-2.2, 3.3)", c, 1);

// call __mulsf3, __fixsfsi
c = (int)test_mul<float, float, float>(-2.2, 3.3); // (int)-7.26 = -7
check_result("(int)test_mul<float, float, float>(-2.2, 3.3)", c, -7);

// call __divsf3, __fixsfsi
c = (int)test_div<float, float, float>(-1.8, 0.5); // (int)-3.6 = -3
check_result("(int)test_div<float, float, float>(-1.8, 0.5)", c, -3);

// call __extendsfdf2, __adddf3, __fixdfsi
c = (int)test_add<double, double, float>(-2.2, 3.3); // (int)1.1 = 1
check_result("(int)test_add<double, double, float>(-2.2, 3.3)", c, 1);

// call __extendsfdf2, __adddf3, __fixdfsi
c = (int)test_add<double, float, double>(-2.2, 3.3); // (int)1.1 = 1
check_result("(int)test_add<double, float, double>(-2.2, 3.3)", c, 1);

// call __extendsfdf2, __adddf3, __fixdfsi
c = (int)test_add<float, float, double>(-2.2, 3.3); // (int)1.1 = 1
check_result("(int)test_add<float, float, double>(-2.2, 3.3)", c, 1);

// call __extendsfdf2, __muldf3, __fixdfsi
c = (int)test_mul<double, float, double>(-2.2, 3.3); // (int)-7.26 = -7
check_result("(int)test_mul<double, float, double>(-2.2, 3.3)", c, -7);

// call __extendsfdf2, __muldf3, __truncdfsf2, __fixdfsi
// ! __truncdfsf2 in truncdfsf2.c is not work for Cpu0

```

(continues on next page)

(continued from previous page)

```

c = (int)test_mul<float, float, double>(-2.2, 3.3); // (int)-7.26 = -7
check_result("(int)test_mul<float, float, double>(-2.2, 3.3)", c, -7);

// call __divdf3, __fixdfsi
c = (int)test_div<double, double, double>(-1.8, 0.5); // (int)-3.6 = -3
check_result("(int)test_div<double, double, double>(-1.8, 0.5)", c, -3);

#if 0 // these three do call builtins
c = (int)test_mul<int, int, int>(-2, 3); // -6
check_result("(int)test_mul<int, int, int>(-2, 3)", c, -6);

c = (int)test_div<int, int, int>(-10, 4); // -2 <- -2*4+2, quotient:-2, remainder:2
↪(remainder < 4:divident)
check_result("(int)test_div<int, int, int>(-10, 4)", c, -3);

a = test_mul<long long, long long, long long>(-2LL, 3LL); // -6LL
check_result("test_mul<long long, long long, long long>(-2LL, 3LL)", a, -6LL);
#endif

// call __divdi3,
a = test_div<long long, long long, long long>(-10LL, 4LL); // -3
check_result("test_div<long long, long long, long long>(-10LL, 4LL)", a, -2LL);

return 0;
}

```

exlbt/input/Makefile.float

```

SRCS := start.cpp debug.cpp sanitizer_printf.cpp printf-stdarg-def.c \
        cpu0-builtins.cpp ch_float.cpp lib_cpu0.c
LIBBUILTINS_DIR := ../compiler-rt/builtins
INC_DIRS := ../ $(NEWLIB_DIR)/newlib/libc/include $(LBDEX_DIR)/input
LIBS := $(LIBBUILTINS_DIR)/build-$(CPU)-$(ENDIAN)/libbuiltins.a

include Common.mk

```

```

chungshu@ChungShudeMacBook-Air input % bash make.sh cpu032II eb Makefile.float
...
endian =  BigEndian
ISR address:00020614
0  /* 0: big endian, 1: little endian */

chungshu@ChungShudeMacBook-Air verilog % iverilog -o cpu0IIs cpu0IIs.v
chungshu@ChungShudeMacBook-Air verilog % ./cpu0IIs
...

a: 1000000007FFFFFFF, 1000000007fffffff, 1152921506754330623
b: 100000000, 268435456

```

(continues on next page)

(continued from previous page)

```

b: 10000000, 268435456
Hello world!
printf test
<null> is null pointer
5 = 5
-2147483647 = - max int
char a = 'a'
hex ff = ff
hex 00 = 00
signed -3 = unsigned 4294967293 = hex ffffffff
0 message(s)
0 message(s) with %
justif: "left      "
justif: "      right"
 3: 0003 zero padded
 3: 3    left justif.
 3:   3 right justif.
-3: -003 zero padded
-3: -3   left justif.
-3:  -3 right justif.
test_longlong_shift1() = 289
test_longlong_shift2() = 22
test_shift_left<long long>(0x12, 4LL) = 288
test_shift_right<long long>(0x0016666600000000a, 48LL) = 22
test_shift_right<unsigned long long>(0x0016666600000000a, 48LLu) = 22
(int)test_add<float, float, float>(-2.2, 3.3) = 1
(int)test_mul<float, float, float>(-2.2, 3.3) = -7
(int)test_div<float, float, float>(-1.8, 0.5) = -3
(int)test_add<double, double, float>(-2.2, 3.3) = 1
(int)test_add<double, float, double>(-2.2, 3.3) = 1
(int)test_add<float, float, double>(-2.2, 3.3) = 1
(int)test_mul<double, float, double>(-2.2, 3.3) = -7
(int)test_mul<float, float, double>(-2.2, 3.3) = -7
(int)test_div<double, double, double>(-1.8, 0.5) = -3
test_div<long long, long long, long long>(-10LL, 4LL) = -2
...
RET to PC < 0, finished!

```

The `exlbt/input/compiler-rt-test/builtins/Unit` copied from `compiler-rt/test/builtins/Unit` as follows,

`exlbt/input/ch_builtins.cpp`

```

#include "debug.h"
#include <stdlib.h>

extern "C" int printf(const char *format, ...);
extern "C" int sprintf(char *out, const char *format, ...);

extern "C" int absvdi2_test();
extern "C" int absvsi2_test();
extern "C" int absvti2_test();

```

(continues on next page)

(continued from previous page)

```

extern "C" int adddf3vfp_test();
extern "C" int addsf3vfp_test();
extern "C" int addvdi3_test();
extern "C" int addvsi3_test();
extern "C" int addvti3_test();
extern "C" int ashldi3_test();
extern "C" int ashlti3_test();
extern "C" int ashrdi3_test();
extern "C" int ashrti3_test();

// atomic.c need memcmp(...)
//extern "C" int atomic_test();
extern "C" int bswapdi2_test();
extern "C" int bswapsi2_test();

extern "C" int clzdi2_test();
extern "C" int clzsi2_test();
extern "C" int clzti2_test();
extern "C" int cmpdi2_test();
extern "C" int cmpti2_test();
extern "C" int comparedf2_test();
extern "C" int comparesf2_test();

// Needless to compare compiler_rt_logb() with logb() of libm
//extern "C" int compiler_rt_logb_test();
//extern "C" int compiler_rt_logbf_test();
//extern "C" int compiler_rt_logbl_test();

extern "C" int cpu_model_test();
extern "C" int ctzdi2_test();
extern "C" int ctzsi2_test();
extern "C" int ctzti2_test();

// div for complex type need libm: fabs, isinf, ..., skip it at this point
#ifdef HAS_COMPLEX
extern "C" int divdc3_test();
#endif
extern "C" int divdf3_test();
extern "C" int divdf3vfp_test();
extern "C" int divdi3_test();
extern "C" int divmodsi4_test();
extern "C" int divmodti4_test();
#ifdef HAS_COMPLEX
extern "C" int divsc3_test();
#endif
extern "C" int divsf3_test();
extern "C" int divsf3vfp_test();
extern "C" int divsi3_test();
#ifdef HAS_COMPLEX
extern "C" int divtc3_test();
#endif
extern "C" int divtf3_test();

```

(continues on next page)

(continued from previous page)

```

extern "C" int divti3_test();
#ifdef HAS_COMPLEX
extern "C" int divxc3_test();
#endif
extern "C" int enable_execute_stack_test();
extern "C" int eqdf2vfp_test();
extern "C" int eqsf2vfp_test();
extern "C" int eqtf2_test();
extern "C" int extenddftf2_test();
extern "C" int extendhfsf2_test();
extern "C" int extendhftf2_test();
extern "C" int extendsfdf2vfp_test();
extern "C" int extendsftf2_test();
#if 0
extern "C" int gcc_personality_test();
#endif
extern "C" int gedf2vfp_test();
extern "C" int gesf2vfp_test();
extern "C" int getf2_test();
extern "C" int gtdf2vfp_test();
extern "C" int gtsf2vfp_test();
extern "C" int gttf2_test();
extern "C" int ledf2vfp_test();
extern "C" int lesf2vfp_test();
extern "C" int letf2_test();
extern "C" int lshrdi3_test();
extern "C" int lshrui3_test();
extern "C" int ltdf2vfp_test();
extern "C" int ltsf2vfp_test();
extern "C" int lttf2_test();
extern "C" int moddi3_test();
extern "C" int modsi3_test();
extern "C" int modst3_test();
extern "C" int modti3_test();
#ifdef HAS_COMPLEX
extern "C" int muldc3_test();
#endif
extern "C" int muldf3vfp_test();
extern "C" int muldi3_test();
extern "C" int mulodi4_test();
extern "C" int mulosi4_test();
extern "C" int muloti4_test();
#ifdef HAS_COMPLEX
extern "C" int mulsc3_test();
#endif
extern "C" int mulsf3vfp_test();
//extern "C" int mulsi3_test(); no this mulsi3.c
#ifdef HAS_COMPLEX
extern "C" int multc3_test();
#endif
extern "C" int multf3_test();
extern "C" int multi3_test();

```

(continues on next page)

(continued from previous page)

```

extern "C" int mulvdi3_test();
extern "C" int mulvsi3_test();
extern "C" int mulvti3_test();
#ifdef HAS_COMPLEX
extern "C" int mulxc3_test();
#endif
extern "C" int nedf2vfp_test();
extern "C" int negdf2vfp_test();
extern "C" int negdi2_test();
extern "C" int negsf2vfp_test();
extern "C" int negti2_test();
extern "C" int negvdi2_test();
extern "C" int negvsi2_test();
extern "C" int negvti2_test();
extern "C" int nesf2vfp_test();
extern "C" int netf2_test();
/* need rand, signbit, ...
extern "C" int paritydi2_test();
extern "C" int paritysi2_test();
extern "C" int parityti2_test();
extern "C" int popcountdi2_test();
extern "C" int popcountsi2_test();
extern "C" int popcountti2_test();
extern "C" int powidf2_test();
extern "C" int powisf2_test();
extern "C" int powitf2_test();
extern "C" int powixf2_test();
*/
extern "C" int subdf3vfp_test();
extern "C" int subsf3vfp_test();
extern "C" int subtf3_test();
extern "C" int subvdi3_test();
extern "C" int subvsi3_test();
extern "C" int subvti3_test();
extern "C" int trampoline_setup_test();
extern "C" int truncdfhf2_test();
extern "C" int truncdfsf2_test();
extern "C" int truncdfsf2vfp_test();
extern "C" int truncsfhf2_test();
extern "C" int truncfdf2_test();
extern "C" int truncfhf2_test();
extern "C" int truncfsf2_test();
extern "C" int ucmpdi2_test();
extern "C" int ucmti2_test();
extern "C" int udivdi3_test();
extern "C" int udivmoddi4_test();
extern "C" int udivmodsi4_test();
extern "C" int udivmodti4_test();
extern "C" int udivsi3_test();
extern "C" int udivti3_test();
extern "C" int umoddi3_test();
extern "C" int umodsi3_test();

```

(continues on next page)

(continued from previous page)

```

extern "C" int umodti3_test();
extern "C" int unorddf2vfp_test();
extern "C" int unordsf2vfp_test();
extern "C" int unordtf2_test();

void show_result(const char *fn, int res) {
    if (res == 1)
        printf("%s: FAIL!\n", fn);
    else if (res == 0)
        printf("%s: PASS!\n", fn);
    else if (res == -1)
        printf("%s: SKIPPED!\n", fn);
    else {
        printf("FIXME!");
        abort();
    }
}

int main() {
    int res = 0;

    // pre-defined compiler macro (from llc -march=cpu0${ENDIAN} or
    // clang -target cpu0${ENDIAN}-unknown-linux-gnu
#ifdef __CPU0EB__
    printf("__CPU0EB__\n");
#endif
#ifdef __CPU0EL__
    printf("__CPU0EL__\n");
#endif

    res = absvdi2_test();
    show_result("absvdi2_test()", res);

    res = absvsi2_test();
    show_result("absvsi2_test()", res);

    res = absvti2_test();
    show_result("absvti2_test()", res);

    res = adddf3vfp_test();
    show_result("adddf3vfp_test()", res);

    res = addsf3vfp_test();
    show_result("addsf3vfp_test()", res);

    res = addvdi3_test();
    show_result("addvdi3_test()", res);

    res = addvsi3_test();
    show_result("addvsi3_test()", res);

    res = addvti3_test();

```

(continues on next page)

(continued from previous page)

```

show_result("addvti3_test()", res);

res = ashldi3_test();
show_result("ashldi3_test()", res);

res = ashlti3_test();
show_result("ashlti3_test()", res);

res = ashrdi3_test();
show_result("ashrdi3_test()", res);

res = ashrti3_test();
show_result("ashrti3_test()", res);

#if 0 // atomic.c need memcmp(...)
    res = atomic_test();
    show_result("atomic_test()", res);
#endif

res = bswapdi2_test();
show_result("bswapdi2_test()", res);

res = bswapsi2_test();
show_result("bswapsi2_test()", res);

res = clzdi2_test();
show_result("clzdi2_test()", res);

res = clzsi2_test();
show_result("clzsi2_test()", res);

res = clzti2_test();
show_result("clzti2_test()", res);

res = cmpdi2_test();
show_result("cmpdi2_test()", res);

res = cmpti2_test();
show_result("cmpti2_test()", res);

res = comparedf2_test();
show_result("comparedf2_test()", res);

res = comparesf2_test();
show_result("comparesf2_test()", res);

// res = compiler_rt_logb_test();
// show_result("compiler_rt_logb_test()", res);

// res = compiler_rt_logbf_test();
// show_result("compiler_rt_logbf_test()", res);

```

(continues on next page)

(continued from previous page)

```

// res = compiler_rt_logbl_test();
// show_result("compiler_rt_logbl_test()", res);

res = cpu_model_test();
show_result("cpu_model_test()", res);

res = ctzdi2_test();
show_result("ctzdi2_test()", res);

res = ctzsi2_test();
show_result("ctzsi2_test()", res);

res = ctzti2_test();
show_result("ctzti2_test()", res);

#ifdef HAS_COMPLEX
res = divdc3_test();
show_result("divdc3_test()", res);
#endif

res = divdf3_test();
show_result("divdf3_test()", res);

res = divdf3vfp_test();
show_result("divdf3vfp_test()", res);

res = divdi3_test();
show_result("divdi3_test()", res);

res = divmodsi4_test();
show_result("divmodsi4_test()", res);

res = divmodti4_test();
show_result("divmodti4_test()", res);

#ifdef HAS_COMPLEX
res = divsc3_test();
show_result("divsc3_test()", res);
#endif

res = divsf3_test();
show_result("divsf3_test()", res);

res = divsf3vfp_test();
show_result("divsf3vfp_test()", res);

res = divsi3_test();
show_result("divsi3_test()", res);

#ifdef HAS_COMPLEX
res = divtc3_test();
show_result("divtc3_test()", res);

```

(continues on next page)

(continued from previous page)

```

#endif

    res = divtf3_test();
    show_result("divtf3_test()", res);

    res = divti3_test();
    show_result("divti3_test()", res);

#ifdef HAS_COMPLEX
    res = divxc3_test();
    show_result("divxc3_test()", res);
#endif

#if 0
    res = enable_execute_stack_test();
    show_result("enable_execute_stack_test()", res);
#endif

    res = eqdf2vfp_test();
    show_result("eqdf2vfp_test()", res);

    res = eqsf2vfp_test();
    show_result("eqsf2vfp_test()", res);

    res = eqtf2_test();
    show_result("eqtf2_test()", res);

    res = extenddftf2_test();
    show_result("extenddftf2_test()", res);

    res = extendhfsf2_test();
    show_result("extendhfsf2_test()", res);

    res = extendhftf2_test();
    show_result("extendhftf2_test()", res);

    res = extendsfdf2vfp_test();
    show_result("extendsfdf2vfp_test()", res);

    res = extendsftf2_test();
    show_result("extendsftf2_test()", res);

#if 0
    res = gcc_personality_test();
    show_result("gcc_personality_test()", res);
#endif

    res = gedf2vfp_test();
    show_result("gedf2vfp_test()", res);

    res = gesf2vfp_test();
    show_result("gesf2vfp_test()", res);

```

(continues on next page)

(continued from previous page)

```
res = getf2_test();
show_result("getf2_test()", res);

res = gtdf2vfp_test();
show_result("gtdf2vfp_test()", res);

res = gtsf2vfp_test();
show_result("gtsf2vfp_test()", res);

res = gttf2_test();
show_result("gttf2_test()", res);

res = ledf2vfp_test();
show_result("ledf2vfp_test()", res);

res = lesf2vfp_test();
show_result("lesf2vfp_test()", res);

res = letf2_test();
show_result("letf2_test()", res);

res = lshrdi3_test();
show_result("lshrdi3_test()", res);

res = lshrti3_test();
show_result("lshrti3_test()", res);

res = ltdf2vfp_test();
show_result("ltdf2vfp_test()", res);

res = ltsf2vfp_test();
show_result("ltsf2vfp_test()", res);

res = lttf2_test();
show_result("lttf2_test()", res);

res = moddi3_test();
show_result("moddi3_test()", res);

res = modsi3_test();
show_result("modsi3_test()", res);

res = modti3_test();
show_result("modti3_test()", res);

#ifdef HAS_COMPLEX
res = muldc3_test();
show_result("muldc3_test()", res);
#endif

res = muldf3vfp_test();
```

(continues on next page)

(continued from previous page)

```

show_result("muldf3vfp_test()", res);

res = muldi3_test();
show_result("muldi3_test()", res);

res = mulodi4_test();
show_result("mulodi4_test()", res);

res = mulosi4_test();
show_result("mulosi4_test()", res);

res = muloti4_test();
show_result("muloti4_test()", res);

#ifdef HAS_COMPLEX
res = mulsc3_test();
show_result("mulsc3_test()", res);
#endif

res = mulsf3vfp_test();
show_result("mulsf3vfp_test()", res);

// no mulsi3.c
// res = mulsi3_test();
// show_result("mulsi3_test()", res);

#ifdef HAS_COMPLEX
res = multc3_test();
show_result("multc3_test()", res);
#endif

res = multf3_test();
show_result("multf3_test()", res);

res = multi3_test();
show_result("multi3_test()", res);

res = mulvdi3_test();
show_result("mulvdi3_test()", res);

res = mulvsi3_test();
show_result("mulvsi3_test()", res);

res = mulvti3_test();
show_result("mulvti3_test()", res);

#ifdef HAS_COMPLEX
res = mulxc3_test();
show_result("mulxc3_test()", res);
#endif

res = nedf2vfp_test();

```

(continues on next page)

(continued from previous page)

```
show_result("nedf2vfp_test()", res);

res = negdf2vfp_test();
show_result("negdf2vfp_test()", res);

res = negdi2_test();
show_result("negdi2_test()", res);

res = negsf2vfp_test();
show_result("negsf2vfp_test()", res);

res = negti2_test();
show_result("negti2_test()", res);

res = negvdi2_test();
show_result("negvdi2_test()", res);

res = negvsi2_test();
show_result("negvsi2_test()", res);

res = negvti2_test();
show_result("negvti2_test()", res);

res = nesf2vfp_test();
show_result("nesf2vfp_test()", res);

res = netf2_test();
show_result("netf2_test()", res);

/* need rand, signbit, ...
res = paritydi2_test();
show_result("paritydi2_test()", res);

res = paritysi2_test();
show_result("paritysi2_test()", res);

res = parityti2_test();
show_result("parityti2_test()", res);

res = popcountdi2_test();
show_result("popcountdi2_test()", res);

res = popcountsi2_test();
show_result("popcountsi2_test()", res);

res = popcountti2_test();
show_result("popcountti2_test()", res);

res = powidf2_test();
show_result("powidf2_test()", res);

res = powisf2_test();
```

(continues on next page)

(continued from previous page)

```

show_result("powisf2_test()", res);

res = powitf2_test();
show_result("powitf2_test()", res);

res = powixf2_test();
show_result("powixf2_test()", res);
*/

res = subdf3vfp_test();
show_result("subdf3vfp_test()", res);

res = subsf3vfp_test();
show_result("subsf3vfp_test()", res);

res = subtf3_test();
show_result("subtf3_test()", res);

res = subvdi3_test();
show_result("subvdi3_test()", res);

res = subvsi3_test();
show_result("subvsi3_test()", res);

res = subvti3_test();
show_result("subvti3_test()", res);

res = trampoline_setup_test();
show_result("trampoline_setup_test()", res);

res = truncdfhf2_test();
show_result("truncdfhf2_test()", res);

res = truncdfs2_test();
show_result("truncdfs2_test()", res);

res = truncdfs2vfp_test();
show_result("truncdfs2vfp_test()", res);

res = truncsfhf2_test();
show_result("truncsfhf2_test()", res);

res = truncfdf2_test();
show_result("truncfdf2_test()", res);

res = truncfhf2_test();
show_result("truncfhf2_test()", res);

res = truncfsf2_test();
show_result("truncfsf2_test()", res);

res = ucmpdi2_test();

```

(continues on next page)

(continued from previous page)

```
show_result("ucmpdi2_test()", res);

res = ucmpdi2_test();
show_result("ucmpdi2_test()", res);

res = udivdi3_test();
show_result("udivdi3_test()", res);

res = udivmoddi4_test();
show_result("udivmoddi4_test()", res);

res = udivmodsi4_test();
show_result("udivmodsi4_test()", res);

res = udivmodti4_test();
show_result("udivmodti4_test()", res);

res = udivsi3_test();
show_result("udivsi3_test()", res);

res = udivti3_test();
show_result("udivti3_test()", res);

res = umoddi3_test();
show_result("umoddi3_test()", res);

res = umodsi3_test();
show_result("umodsi3_test()", res);

res = umodti3_test();
show_result("umodti3_test()", res);

res = unorddf2vfp_test();
show_result("unorddf2vfp_test()", res);

res = unordsf2vfp_test();
show_result("unordsf2vfp_test()", res);

res = unordtf2_test();
show_result("unordtf2_test()", res);

return 0;
}
```


exlbt/input/Makefile.builtins

```

# CPU and endian passed from command line, such as
#   "make -f Makefile.builtins CPU=cpu032II ENDIAN=eb or
#   "make -f Makefile.builtins CPU=cpu032I  ENDIAN=el

# start.cpp must be put at beginning
SRCS := start.cpp debug.cpp syscalls.c sanitizer_printf.cpp printf-stdarg-def.c \
  compiler-rt-test/builtins/Unit/absvdi2_test.c \
  compiler-rt-test/builtins/Unit/absvsi2_test.c \
  compiler-rt-test/builtins/Unit/absvti2_test.c \
  compiler-rt-test/builtins/Unit/adddf3vfp_test.c \
  compiler-rt-test/builtins/Unit/addsf3vfp_test.c \
  compiler-rt-test/builtins/Unit/addvdi3_test.c \
  compiler-rt-test/builtins/Unit/addvsi3_test.c \
  compiler-rt-test/builtins/Unit/addvti3_test.c \
  compiler-rt-test/builtins/Unit/ashldi3_test.c \
  compiler-rt-test/builtins/Unit/ashlti3_test.c \
  compiler-rt-test/builtins/Unit/ashrdi3_test.c \
  compiler-rt-test/builtins/Unit/ashrti3_test.c \
  compiler-rt-test/builtins/Unit/bswapdi2_test.c \
  compiler-rt-test/builtins/Unit/bswapsi2_test.c \
  compiler-rt-test/builtins/Unit/clzdi2_test.c \
  compiler-rt-test/builtins/Unit/clzsi2_test.c \
  compiler-rt-test/builtins/Unit/clzti2_test.c \
  compiler-rt-test/builtins/Unit/cmpdi2_test.c \
  compiler-rt-test/builtins/Unit/cmpti2_test.c \
  compiler-rt-test/builtins/Unit/comparedf2_test.c \
  compiler-rt-test/builtins/Unit/comparesf2_test.c \
  compiler-rt-test/builtins/Unit/cpu_model_test.c \
  compiler-rt-test/builtins/Unit/ctzdi2_test.c \
  compiler-rt-test/builtins/Unit/ctzsi2_test.c \
  compiler-rt-test/builtins/Unit/ctzti2_test.c \
  compiler-rt-test/builtins/Unit/divdc3_test.c \
  compiler-rt-test/builtins/Unit/divdf3_test.c \
  compiler-rt-test/builtins/Unit/divdf3vfp_test.c \
  compiler-rt-test/builtins/Unit/divdi3_test.c \
  compiler-rt-test/builtins/Unit/divmodsi4_test.c \
  compiler-rt-test/builtins/Unit/divmodti4_test.c \
  compiler-rt-test/builtins/Unit/divsc3_test.c \
  compiler-rt-test/builtins/Unit/divsf3_test.c \
  compiler-rt-test/builtins/Unit/divsf3vfp_test.c \
  compiler-rt-test/builtins/Unit/divsi3_test.c \
  compiler-rt-test/builtins/Unit/divtc3_test.c \
  compiler-rt-test/builtins/Unit/divtf3_test.c \
  compiler-rt-test/builtins/Unit/divti3_test.c \
  compiler-rt-test/builtins/Unit/divxc3_test.c \
  compiler-rt-test/builtins/Unit/enable_execute_stack_test.c \
  compiler-rt-test/builtins/Unit/eqdf2vfp_test.c \
  compiler-rt-test/builtins/Unit/eqsf2vfp_test.c \
  compiler-rt-test/builtins/Unit/eqtf2_test.c \
  compiler-rt-test/builtins/Unit/extenddftf2_test.c \
  compiler-rt-test/builtins/Unit/extendhfsf2_test.c \

```

(continues on next page)

(continued from previous page)

```

compiler-rt-test/builtins/Unit/extendhftf2_test.c \
compiler-rt-test/builtins/Unit/extendsfdf2vfp_test.c \
compiler-rt-test/builtins/Unit/extendsftf2_test.c \
compiler-rt-test/builtins/Unit/gedf2vfp_test.c \
compiler-rt-test/builtins/Unit/gesf2vfp_test.c \
compiler-rt-test/builtins/Unit/getf2_test.c \
compiler-rt-test/builtins/Unit/gtdf2vfp_test.c \
compiler-rt-test/builtins/Unit/gtsf2vfp_test.c \
compiler-rt-test/builtins/Unit/gttf2_test.c \
compiler-rt-test/builtins/Unit/ledf2vfp_test.c \
compiler-rt-test/builtins/Unit/lesf2vfp_test.c \
compiler-rt-test/builtins/Unit/letf2_test.c \
compiler-rt-test/builtins/Unit/lshr3_test.c \
compiler-rt-test/builtins/Unit/lshrti3_test.c \
compiler-rt-test/builtins/Unit/ltdf2vfp_test.c \
compiler-rt-test/builtins/Unit/ltsf2vfp_test.c \
compiler-rt-test/builtins/Unit/lttf2_test.c \
compiler-rt-test/builtins/Unit/moddi3_test.c \
compiler-rt-test/builtins/Unit/modsi3_test.c \
compiler-rt-test/builtins/Unit/modti3_test.c \
compiler-rt-test/builtins/Unit/muldc3_test.c \
compiler-rt-test/builtins/Unit/muldf3vfp_test.c \
compiler-rt-test/builtins/Unit/muldi3_test.c \
compiler-rt-test/builtins/Unit/mulodi4_test.c \
compiler-rt-test/builtins/Unit/mulosi4_test.c \
compiler-rt-test/builtins/Unit/muloti4_test.c \
compiler-rt-test/builtins/Unit/mulsc3_test.c \
compiler-rt-test/builtins/Unit/mulsf3vfp_test.c \
compiler-rt-test/builtins/Unit/mulsi3_test.c \
compiler-rt-test/builtins/Unit/multc3_test.c \
compiler-rt-test/builtins/Unit/multf3_test.c \
compiler-rt-test/builtins/Unit/multi3_test.c \
compiler-rt-test/builtins/Unit/mulvdi3_test.c \
compiler-rt-test/builtins/Unit/mulvsi3_test.c \
compiler-rt-test/builtins/Unit/mulvti3_test.c \
compiler-rt-test/builtins/Unit/mulxc3_test.c \
compiler-rt-test/builtins/Unit/nedf2vfp_test.c \
compiler-rt-test/builtins/Unit/negdf2vfp_test.c \
compiler-rt-test/builtins/Unit/negdi2_test.c \
compiler-rt-test/builtins/Unit/negsf2vfp_test.c \
compiler-rt-test/builtins/Unit/negti2_test.c \
compiler-rt-test/builtins/Unit/negvdi2_test.c \
compiler-rt-test/builtins/Unit/negvsi2_test.c \
compiler-rt-test/builtins/Unit/negvti2_test.c \
compiler-rt-test/builtins/Unit/nesf2vfp_test.c \
compiler-rt-test/builtins/Unit/netf2_test.c \
compiler-rt-test/builtins/Unit/paritydi2_test.c \
compiler-rt-test/builtins/Unit/paritysi2_test.c \
compiler-rt-test/builtins/Unit/parityti2_test.c \
compiler-rt-test/builtins/Unit/popcountdi2_test.c \
compiler-rt-test/builtins/Unit/popcountsi2_test.c \
compiler-rt-test/builtins/Unit/popcountti2_test.c \

```

(continues on next page)

(continued from previous page)

```

compiler-rt-test/builtins/Unit/powidf2_test.c \
compiler-rt-test/builtins/Unit/powisf2_test.c \
compiler-rt-test/builtins/Unit/powitf2_test.c \
compiler-rt-test/builtins/Unit/powixf2_test.c \
compiler-rt-test/builtins/Unit/subdf3vfp_test.c \
compiler-rt-test/builtins/Unit/subsf3vfp_test.c \
compiler-rt-test/builtins/Unit/subtf3_test.c \
compiler-rt-test/builtins/Unit/subvdi3_test.c \
compiler-rt-test/builtins/Unit/subvsi3_test.c \
compiler-rt-test/builtins/Unit/subvti3_test.c \
compiler-rt-test/builtins/Unit/trampoline_setup_test.c \
compiler-rt-test/builtins/Unit/truncdfhf2_test.c \
compiler-rt-test/builtins/Unit/truncdfsf2_test.c \
compiler-rt-test/builtins/Unit/truncdfsf2vfp_test.c \
compiler-rt-test/builtins/Unit/truncsfhf2_test.c \
compiler-rt-test/builtins/Unit/trunctfdf2_test.c \
compiler-rt-test/builtins/Unit/trunctfhf2_test.c \
compiler-rt-test/builtins/Unit/trunctfsf2_test.c \
compiler-rt-test/builtins/Unit/ucmpdi2_test.c \
compiler-rt-test/builtins/Unit/ucmpti2_test.c \
compiler-rt-test/builtins/Unit/udivdi3_test.c \
compiler-rt-test/builtins/Unit/udivmoddi4_test.c \
compiler-rt-test/builtins/Unit/udivmodsi4_test.c \
compiler-rt-test/builtins/Unit/udivmodti4_test.c \
compiler-rt-test/builtins/Unit/udivsi3_test.c \
compiler-rt-test/builtins/Unit/udivti3_test.c \
compiler-rt-test/builtins/Unit/umoddi3_test.c \
compiler-rt-test/builtins/Unit/umodsi3_test.c \
compiler-rt-test/builtins/Unit/umodti3_test.c \
compiler-rt-test/builtins/Unit/unorddf2vfp_test.c \
compiler-rt-test/builtins/Unit/unordsf2vfp_test.c \
compiler-rt-test/builtins/Unit/unordtf2_test.c \
cpu0-builtins.cpp ch_builtins.cpp lib_cpu0.c

INC_DIRS := ./ $(LBDEX_DIR)/input \
             $(HOME)/llvm/llvm-project/compiler-rt/lib/builtins \
             $(NEWLIB_DIR)/newlib/libc/include \
             $(NEWLIB_DIR)/libgloss
LIBBUILTINS_DIR := ../compiler-rt/builtins
LIBS := $(LIBBUILTINS_DIR)/build-$(CPU)-$(ENDIAN)/libbuiltins.a \
        $(NEWLIB_DIR)/build-$(CPU)-$(ENDIAN)/libm.a \
        $(NEWLIB_DIR)/build-$(CPU)-$(ENDIAN)/libc.a

include Common.mk

```

Run as follows,

```

chungshu@ChungShudeMacBook-Air input % bash make.sh cpu032II eb Makefile.builtins
...
chungshu@ChungShudeMacBook-Air verilog % ./cpu0IIs
...
absvdi2_test(): PASS!

```

(continues on next page)

(continued from previous page)

```
absvsi2_test(): PASS!
absvti2_test(): SKIPPED!
adddf3vfp_test(): SKIPPED!
addsf3vfp_test(): SKIPPED!
addvdi3_test(): PASS!
addvsi3_test(): PASS!
addvti3_test(): SKIPPED!
ashldi3_test(): PASS!
ashlti3_test(): SKIPPED!
ashrdi3_test(): PASS!
ashrti3_test(): SKIPPED!
bswapdi2_test(): PASS!
bswapsi2_test(): PASS!
clzdi2_test(): PASS!
clzsi2_test(): PASS!
clzti2_test(): SKIPPED!
cmpdi2_test(): PASS!
cmpti2_test(): SKIPPED!
comparedf2_test(): PASS!
comparesf2_test(): PASS!
cpu_model_test(): SKIPPED!
ctzdi2_test(): PASS!
ctzsi2_test(): PASS!
ctzti2_test(): SKIPPED!
divdc3_test(): PASS!
divdf3_test(): PASS!
divdf3vfp_test(): SKIPPED!
divdi3_test(): PASS!
divmodsi4_test(): PASS!
divmodti4_test(): SKIPPED!
divsf3_test(): PASS!
divsf3vfp_test(): SKIPPED!
divsi3_test(): PASS!
divtc3_test(): PASS!
divtf3_test(): SKIPPED!
divti3_test(): SKIPPED!
divxc3_test(): PASS!
eqdf2vfp_test(): SKIPPED!
eqsf2vfp_test(): SKIPPED!
eqtf2_test(): SKIPPED!
extenddftf2_test(): SKIPPED!
extendhfsf2_test(): PASS!
extendhftf2_test(): SKIPPED!
extendsfdf2vfp_test(): SKIPPED!
extendsftf2_test(): SKIPPED!
gedf2vfp_test(): SKIPPED!
gesf2vfp_test(): SKIPPED!
getf2_test(): SKIPPED!
gtdf2vfp_test(): SKIPPED!
gtsf2vfp_test(): SKIPPED!
gttf2_test(): SKIPPED!
ledf2vfp_test(): SKIPPED!
```

(continues on next page)

(continued from previous page)

```
lesf2vfp_test(): SKIPPED!
letf2_test(): SKIPPED!
lshr3_test(): PASS!
lshrt3_test(): SKIPPED!
ltdf2vfp_test(): SKIPPED!
ltsf2vfp_test(): SKIPPED!
lttf2_test(): SKIPPED!
moddi3_test(): PASS!
modsi3_test(): PASS!
modti3_test(): SKIPPED!
muldc3_test(): PASS!
muldf3vfp_test(): SKIPPED!
muldi3_test(): PASS!
mulodi4_test(): PASS!
mulosi4_test(): PASS!
muloti4_test(): SKIPPED!
mulsc3_test(): PASS!
mulsf3vfp_test(): SKIPPED!
multc3_test(): SKIPPED!
multf3_test(): SKIPPED!
multi3_test(): SKIPPED!
mulvdi3_test(): PASS!
mulvsi3_test(): PASS!
mulvti3_test(): SKIPPED!
mulxc3_test(): PASS!
nedf2vfp_test(): SKIPPED!
negdf2vfp_test(): SKIPPED!
negdi2_test(): PASS!
negsf2vfp_test(): SKIPPED!
negti2_test(): SKIPPED!
negvdi2_test(): PASS!
negvsi2_test(): PASS!
negvti2_test(): SKIPPED!
nesf2vfp_test(): SKIPPED!
netf2_test(): SKIPPED!
subdf3vfp_test(): SKIPPED!
subsf3vfp_test(): SKIPPED!
subtf3_test(): SKIPPED!
subvdi3_test(): PASS!
subvsi3_test(): PASS!
subvti3_test(): SKIPPED!
trampoline_setup_test(): SKIPPED!
truncdfhf2_test(): PASS!
truncdfsf2_test(): PASS!
truncdfsf2vfp_test(): SKIPPED!
truncsfhf2_test(): PASS!
trunctfdf2_test(): SKIPPED!
trunctfhf2_test(): SKIPPED!
trunctfsf2_test(): SKIPPED!
ucmpdi2_test(): PASS!
ucmpti2_test(): SKIPPED!
udivdi3_test(): PASS!
```

(continues on next page)

(continued from previous page)

```
udivmoddi4_test(): PASS!  
udivmodsi4_test(): PASS!  
udivmodti4_test(): SKIPPED!  
udivsi3_test(): PASS!  
udivti3_test(): SKIPPED!  
umoddi3_test(): PASS!  
umodsi3_test(): PASS!  
umodti3_test(): SKIPPED!  
unorddf2vfp_test(): SKIPPED!  
unordsf2vfp_test(): SKIPPED!  
unordtf2_test(): SKIPPED!  
...  
RET to PC < 0, finished!
```

LLDB

With scanf from newlib/libc and printf from compiler-rt, now ready to porting lldb. Beginning...

APPENDIX A: RISC-V

- *ISA*
- *Mem*
- *RISC compiler toolchain installation*
- *Linker Command*
- *Qemu simulator*
- *Gem5 simulator*
- *GDB*
- *RISC-V Calling Convention*
- *RVV*
- *Atomic instructions*
- *RISC-V+NPU for Deep Learning*
- *Resources*
 - *FreeBSD*
 - *FreeRTOS*
 - *Zephyr*
 - *Andes*
- *Websites*
- *To do:*

This chapter shows the RISC toolchain installation including gnu, llvm and simulators on Linux as figure and table below.

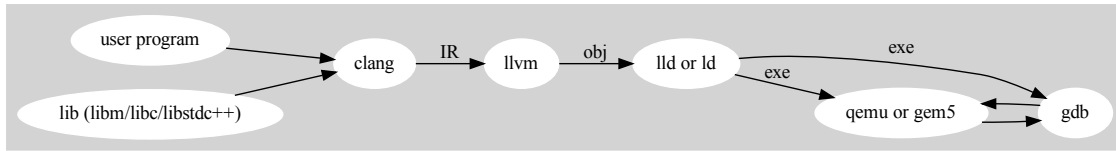


Fig. 7.1: RISC-V toolchain flow

Table 7.1: RISC-V toolchain^{Page 118, 1}

Component	name	github
C/C++ Compiler	clang/llvm	llvm-project
LLVM Assembler	llvm integrated assembler	“
LLVM Linker	ld.lld	“
debug tool	lldb	“
Utils	llvm-ar, llvm-objdump etc.	“
gcc Assembler	as	riscv-gnu-toolchain
gcc Linker	ld.bfd ld.gold	“
Runtime	libgcc	“
Unwinder	libgcc_s	“
C library	libc	“
C++ library	libsupc++ libstdc++	“
debug tool	gdb	“
Utils	ar, objdump etc.	“
Functional sim	qemu	qemu
Cycle sim	gem5	gem5

7.1 ISA

As above Fig. 7.2 and Fig. 7.3, RISC has 32/64/128 bit and I (integer) is the Base part and the others are optional. G=IMAxFD, general extensions (i.e., IMAFD)²³⁴.

Since RISC-V has vector instruction for variable length of data and allowing vendor to encode variable length of instruction set, the little endian is the dominate format in market⁵.

¹ Reference “Table 1 Toolchain components” of <http://jonathan2251.github.io/lbt/about.html#outline-of-chapters>

² <https://riscv.org/technical/specifications/>

³ <https://en.wikipedia.org/wiki/RISC-V>

⁴ <https://wiki.riscv.org/display/HOME/Recently+Ratified+Extensions>

⁵ <https://www.technicalsourcery.net/posts/on-endianness>

Base	Version	Draft Frozen?
RV32I	2.0	Y
RV32E	1.9	N
RV64I	2.0	Y
RV128I	1.7	N
Extension	Version	Frozen?
M	2.0	Y
A	2.0	Y
F	2.0	Y
D	2.0	Y
Q	2.0	Y
L	0.0	N
C	2.0	Y
B	0.0	N
J	0.0	N
T	0.0	N
P	0.1	N
V	0.7	N
N	1.1	N

Fig. 7.2: RISC-V ISA

Bit	Character	Description
0	A	Atomic extension
1	B	<i>Tentatively reserved for Bit-Manipulation extension</i>
2	C	Compressed extension
3	D	Double-precision floating-point extension
4	E	RV32E base ISA
5	F	Single-precision floating-point extension
6	G	<i>Reserved</i>
7	H	Hypervisor extension
8	I	RV32I/64I/128I base ISA
9	J	<i>Tentatively reserved for Dynamically Translated Languages extension</i>
10	K	<i>Reserved</i>
11	L	<i>Reserved</i>
12	M	Integer Multiply/Divide extension
13	N	<i>Tentatively reserved for User-Level Interrupts extension</i>
14	O	<i>Reserved</i>
15	P	<i>Tentatively reserved for Packed-SIMD extension</i>
16	Q	Quad-precision floating-point extension
17	R	<i>Reserved</i>
18	S	Supervisor mode implemented
19	T	<i>Reserved</i>
20	U	User mode implemented
21	V	<i>Tentatively reserved for Vector extension</i>
22	W	<i>Reserved</i>
23	X	Non-standard extensions present
24	Y	<i>Reserved</i>
25	Z	<i>Reserved</i>

Table 3.2: **Encoding of Extensions** field in `misr`. All bits that are reserved for future use must return zero when read.

Fig. 7.3: RISC-V ISA Description

7.2 Mem

- I-cache, D-cache: Size: 4KB to 64KB in Andes N25f
- ILM, DLM: Size: 4KB to 16MB⁶
 - For deterministic and efficient program execution
 - Flexible size selection to fit diversified needs
- DRAM

7.3 RISC compiler toolchain installation

First, install the depended packages according <https://github.com/riscv-collab/riscv-gnu-toolchain#readme>. Next, create your \$HOME/riscv and \$HOME/riscv/git. Then run the following bash script.

exlbt/riscv/riscv-toolchain-setup.sh

```
#!/usr/bin/env bash

# Verified on ubuntu 18.04
# mkdir riscv/git, riscv/riscv_newlib, riscv_linux before running this bash script
export LLVM_VER=14.x
#export LLVM_VER=13.0.0
export GNU_SRC_DIR=$HOME/riscv/git
export LLVM_SRC_DIR=$HOME/riscv/git/$LLVM_VER

export GNU_NEWLIB_INSTALL_DIR=$HOME/riscv/$LLVM_VER/riscv_newlib
export LLVM_NEWLIB_BUILD_DIR=$LLVM_SRC_DIR/llvm-project/build_riscv_newlib

export GNU_LINUX_INSTALL_DIR=$HOME/riscv/$LLVM_VER/riscv_linux
export LLVM_LINUX_BUILD_DIR=$LLVM_SRC_DIR/llvm-project/build_riscv_linux

riscv_gnu_toolchain_prerequisites() {
    sudo apt-get install autoconf automake autotools-dev curl python3 libmpc-dev \
    libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo gperf libtool \
    patchutils bc zlib1g-dev libexpat-dev
    if [ ! -f "/usr/bin/python" ]; then
        sudo ln -s /usr/bin/python3 /usr/bin/python
    fi
}

riscv_llvm_prerequisites() {
    sudo apt-get install ninja-build
}

get_llvm() {
    if [ ! -d "$GNU_SRC_DIR" ]; then
```

(continues on next page)

⁶ <http://www.andestech.com/en/products-solutions/andestech-processors/riscv-n25f/> , <http://www.andestech.com/en/products-solutions/andestech-architecture/>

(continued from previous page)

```

    echo "GNU_SRC_DIR: $GNU_SRC_DIR not exist"
    exit 1
fi
rm -rf $LLVM_SRC_DIR
mkdir $LLVM_SRC_DIR
pushd $LLVM_SRC_DIR
git clone https://github.com/llvm/llvm-project.git
cd llvm-project
git checkout -b $LLVM_VER origin/release/$LLVM_VER
popd
}

check() {
    if [ ! -d "$GNU_SRC_DIR" ]; then
        echo "GNU_SRC_DIR: $GNU_SRC_DIR not exist"
        exit 1
    fi
    if [ -d "$GNU_NEWLIB_INSTALL_DIR" ]; then
        echo "GNU_NEWLIB_INSTALL_DIR: $GNU_NEWLIB_INSTALL_DIR exist. Remove it before
↳running."
        exit 1
    fi
    if [ -d "$GNU_LINUX_INSTALL_DIR" ]; then
        echo "GNU_LINUX_INSTALL_DIR: $GNU_LINUX_INSTALL_DIR exist. Remove it before running."
        exit 1
    fi
}

build_gnu_toolchain() {
    pushd $GNU_SRC_DIR
    git clone https://github.com/riscv/riscv-gnu-toolchain
    cd riscv-gnu-toolchain
    # Looks branch change from original/rvv-intrinsic to origin/__archive__
    # git checkout -b rvv-intrinsic origin/rvv-intrinsic
    # commit 409b951ba6621f2f115aebddfb15ce2dd78ec24f of master branch is work for vadd.vv
↳of vadd1.c
    mkdir build_newlib
    cd build_newlib
    # NX27V is 32-64 bits configurable and has HW float point
    ../configure --prefix=$GNU_NEWLIB_INSTALL_DIR \
        --with-arch=rv64gc --with-abi=lp64d
    # --with-multilib-generator="rv32i-ilp32--;rv32imafd-ilp32--;rv64ima-lp64--"
    make -j4

    cd ..
    mkdir build_linux
    cd build_linux
    ../configure --prefix=$GNU_LINUX_INSTALL_DIR
    make linux -j4
    popd
}

```

(continues on next page)

(continued from previous page)

```

# LLVM_OPTIMIZED_TABLEGEN: Builds a release tablegen that gets used during the LLVM_
↳ build. This can dramatically speed up debug builds.
# LLVM_INSTALL_TOOLCHAIN_ONLY default is OFF already. Check CmakeCache.txt.
#   https://llvm.org/docs/BuildingADistribution.html?highlight=llvm_install_toolchain_
↳ only#difference-between-install-and-install-distribution
# LLVM_BINUTILS_INCDIR:
#   https://stackoverflow.com/questions/45715423/how-to-enable-cfi-in-llvm
#   For lld. https://llvm.org/docs/GoldPlugin.html
#   For llvm version 13.0.0 -DLLVM_BINUTILS_INCDIR will fail on ninja as follows,
#   /home/jonathanchen/riscv/git/13.0.0/llvm-project/llvm/tools/gold/gold-plugin.
↳ cpp:38:10: fatal error: plugin-api.h: No such file or directory
#   #include <plugin-api.h>
#
#   ^~~~~~
# -DLLVM_BINUTILS_INCDIR=$GNU_SRC_DIR/riscv-gnu-toolchain/riscv-binutils/include will_
↳ incur above fail on 13.x
# DEFAULT_SYSROOT:
#   https://stackoverflow.com/questions/66357013/compile-clang-with-alternative-sysroot
# LLVM_DEFAULT_TARGET_TRIPLE:
#   https://clang.llvm.org/docs/CrossCompilation.html#general-cross-compilation-options-
↳ in-clang
# LLVM_INSTALL_UTILS:BOOL
#   If enabled, utility binaries like FileCheck and not will be installed to CMAKE_
↳ INSTALL_PREFIX.
#   https://llvm.org/docs/CMake.html
# Use "clang --sysroot" if did not "cmake -DDEFAULT_SYSROOT"
# $LLVM_NEWLIB_BUILD_DIR/bin/clang++ --gcc-toolchain=$GNU_NEWLIB_INSTALL_DIR test.cpp -
↳ march=rv64g -O0 -mabi=lp64d -v
# $LLVM_LINUX_BUILD_DIR/bin/clang++ --gcc-toolchain=$GNU_LINUX_INSTALL_DIR --sysroot=
↳ $GNU_LINUX_INSTALL_DIR/sysroot/ --static test.cpp -march=rv64g -O0 -mabi=lp64d -v
build_llvm_toolchain() {
    rm -rf $LLVM_NEWLIB_BUILD_DIR
    mkdir $LLVM_NEWLIB_BUILD_DIR
    pushd $LLVM_NEWLIB_BUILD_DIR
    cmake -G "Ninja" -DCMAKE_BUILD_TYPE=Debug -DLLVM_TARGETS_TO_BUILD="RISCV" \
        -DLLVM_ENABLE_PROJECTS="clang;lld" \
        -DLLVM_OPTIMIZED_TABLEGEN=On \
        -DCMAKE_INSTALL_PREFIX=$GNU_NEWLIB_INSTALL_DIR -DLLVM_PARALLEL_COMPILE_JOBS=4 \
        -DLLVM_PARALLEL_LINK_JOBS=1 -DLLVM_DEFAULT_TARGET_TRIPLE=riscv64-unknown-elf \
        -DDEFAULT_SYSROOT=$GNU_NEWLIB_INSTALL_DIR/riscv64-unknown-elf \
        -DLLVM_INSTALL_UTILS=ON ../llvm
    ninja
    ninja install
    popd
    rm -rf $LLVM_LINUX_BUILD_DIR
    mkdir $LLVM_LINUX_BUILD_DIR
    pushd $LLVM_LINUX_BUILD_DIR
    cmake -G "Ninja" -DCMAKE_BUILD_TYPE=Debug -DLLVM_TARGETS_TO_BUILD="RISCV" \
        -DLLVM_ENABLE_PROJECTS="clang;lld" \
        -DLLVM_OPTIMIZED_TABLEGEN=On \
        -DCMAKE_INSTALL_PREFIX=$GNU_LINUX_INSTALL_DIR -DLLVM_PARALLEL_COMPILE_JOBS=4 \
        -DLLVM_PARALLEL_LINK_JOBS=1 -DLLVM_DEFAULT_TARGET_TRIPLE=riscv64-unknown-linux-gnu \
        -DDEFAULT_SYSROOT=$GNU_LINUX_INSTALL_DIR/sysroot -DLLVM_INSTALL_UTILS=ON ../llvm

```

(continues on next page)

(continued from previous page)

```

ninja
ninja install
popd
}

riscv_gnu_toolchain_prerequisites;
riscv_llvm_prerequisites;
get_llvm;
check;
build_gnu_toolchain;
build_llvm_toolchain;

```

```

$ pwd
$ $HOME/git/lbt/exlbt/riscv
$ bash riscv-toolchain-setup.sh

```

RISCV toolchain includes both baremetal(Newlib) and Linux platform support.

```

$ pwd
$ $HOME/git/lbt/exlbt/riscv
$ ls $HOME/riscv/riscv_newlib
bin include lib libexec riscv64-unknown-elf share

```

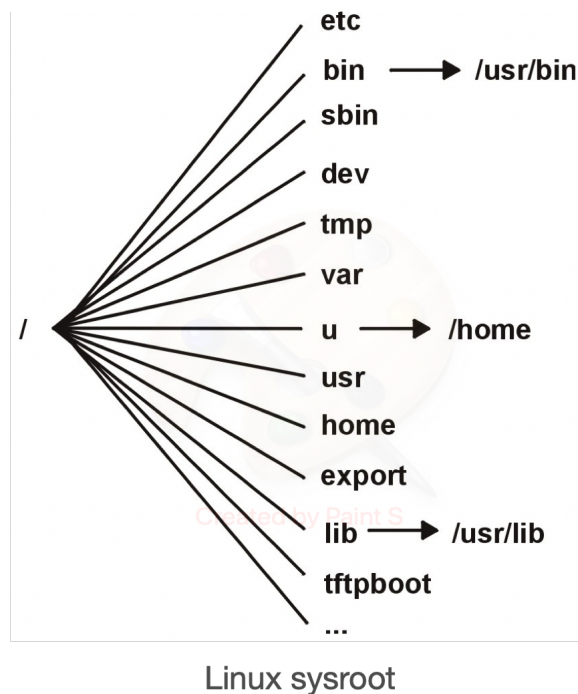


Fig. 7.4: RISCV ISA Description

The linux sysroot as Fig. 7.4 above. You can compare it with the following installed directory.

```

$ ls $HOME/riscv/riscv_linux/sysroot/

```

(continues on next page)

(continued from previous page)

```
etc lib lib64 sbin usr var
$ ls $HOME/riscv/riscv_linux/sysroot/usr
bin include lib libexec sbin share
```

7.4 Linker Command

Different HW platforms have their memory map for their RISC-V architecture. As result, their HW may need to initialize \$sp, \$pc and \$gp GNU linker command language¹⁵ provides user's memory map to their HW.

The crt0.S and riscv64-virt.ld in lbt/exlbt/riscv modified from origin as follows,

```
riscv$ pwd
~/git/lbt/exlbt/riscv
riscv$ diff exlbt/riscv/crt0.S ~/riscv/git/riscv-gnu-toolchain/newlib/libgloss/riscv/
↪ crt0.S
22d21
< la sp, __stack_top
riscv$ ~/riscv/14.x/riscv_newlib/bin/riscv64-unknown-elf-ld --verbose &> riscv64-virt-
↪ origin.ld
riscv$ diff riscv64-virt-origin.ld riscv64-virt.ld
1,8d0
< GNU ld (GNU Binutils) 2.39
< Supported emulations:
< elf64lrvscv
< elf32lrvscv
< elf64briscv
< elf32briscv
< using internal linker script:
< =====
16a9,12
> MEMORY
> {
>   RAM (rx) : ORIGIN = 0x10000, LENGTH = 128M
> }
22a19
> PROVIDE(__stack_top = ORIGIN(RAM) + LENGTH(RAM));
257,259d253
<
<
< =====
riscv$ ~/riscv/14.x/riscv_newlib/bin/clang hello.c -menable-experimental-extensions \
-march=rv64gcvlp0 -O0 -mabi=lp64d -T riscv64-virt.ld -nostartfiles crt0.S -v
```

If uses RAM (rwx), it has warning and can be removed by *-Wl,-no-warn-rwx-segment* in clang option.

Qemu on next section can run program without initializing \$sp in crt0.S. Maybe it is that qemu initialize \$sp as in the following code.

```
qemu$ pwd
~/riscv/git/qemu
```

(continues on next page)

¹⁵ https://ftp.gnu.org/old-gnu/Manuals/ld-2.9.1/html_chapter/ld_3.html

(continued from previous page)

```
qemu$ vi linux-user/riscv/cpu_loop.c
void target_cpu_copy_regs(CPUArchState *env, struct target_pt_regs *regs)
{
    ...
    env->gpr[xSP] = regs->sp;
}
```

The ELF object file format uses program headers, which are read by the system loader and describe how the program should be loaded into memory. The program headers of an ELF file may be displayed using *llvm-objdump -p*¹⁶.

The “la sp, __stack_top” is Psuedo instructions in RISCv¹⁷.

7.5 Qemu simulator

Install according <https://gitlab.com/qemu-project/qemu> as follows,

```
$ pwd
$ $HOME/riscv/git
$ git clone https://gitlab.com/qemu-project/qemu.git
$ cd qemu
$ git log
commit a28498b1f9591e12dcbfdf06dc8f54e15926760e
...
$ mkdir build
$ cd build
$ ../configure
$ make
```

Then it can compile and run qemu for baremetal as follows,

```
$ pwd
$ $HOME/git/lbt/exlbt/riscv
$ $HOME/riscv/riscv_newlib/bin/clang -march=rv64g hello.c -fuse-ld=lld \
-mno-relax -g -mabi=lp64d -o hello_newlib
$ $HOME/riscv/git/qemu/build/qemu-riscv64 hello_newlib
hello world!
```

Linux¹⁸ as follows,

```
$ $HOME/riscv/riscv_linux/bin/clang -march=rv64g hello.c -o hello_linux -static
$ $HOME/riscv/git/qemu/build/qemu-riscv64 hello_linux
hello world!
```

RISCv does need -lm for math.h as follows,

¹⁶ https://ftp.gnu.org/old-gnu/Manuals/ld-2.9.1/html_chapter/ld_3.html#SEC23

¹⁷ <https://github.com/riscv-non-isa/riscv-asm-manual/blob/master/riscv-asm.md#a-listing-of-standard-risc-v-pseudoinstructions>

¹⁸ <https://github.com/riscv-collab/riscv-gnu-toolchain/issues/644>

exlbt/riscv/pow.cpp

```
// RISCv does need -lm while X86-64 does not.
// $ ~/riscv/riscv_newlib/bin/clang++ -menable-experimental-extensions pow.cpp -
↳march=rv64gcv0p10 -O0 -fuse-ld=lld -mno-relax -g -mabi=lp64d -lm -v
// $ ~/riscv/git/qemu/build/qemu-riscv64 -cpu rv64,v=true a.out

#include <stdio.h>
#include <math.h>

double base = 100;
double power = 2;
double test_math()
{
    double res = 0;

    res = pow(base, power);

    return res;
}

int main() {
    double a = test_math();
    printf("a = %lf\n", a);
    return 0;
}
```

The asm for hello world in baremetal as follows,

```
$ $HOME/riscv/riscv_newlib/bin/riscv64-unknown-elf-gcc -c hello_world.s
$ $HOME/riscv/riscv_newlib/bin/riscv64-unknown-elf-ld hello_world.o -o hello_world
$ $HOME/riscv/git/qemu/build/qemu-riscv64 hello_world
Hello World
```

The link for asm and C in baremetal as follows,

exlbt/riscv/caller_hello.c

```
/* ~/git/lbt/exlbt/riscv$ ~/riscv/riscv_newlib/bin/riscv64-unknown-elf-gcc -c func_hello_
↳start.s
~/git/lbt/exlbt/riscv$ ~/riscv/riscv_newlib/bin/riscv64-unknown-elf-gcc -c caller_hello.c
~/git/lbt/exlbt/riscv$ ~/riscv/riscv_newlib/bin/riscv64-unknown-elf-ld caller_hello.o_
↳func_hello_start.o
~/git/lbt/exlbt/riscv$ ~/riscv/riscv_newlib/bin/riscv64-unknown-elf-ld caller_hello.o_
↳func_hello_start.o -o a.out
~/git/lbt/exlbt/riscv$ ~/riscv/git/qemu/build/qemu-riscv64 a.out
Hello World
*/

extern void hello();

int main() {
```

(continues on next page)

(continued from previous page)

```
hello();
}
```

exlbt/riscv/func_hello_start.s

```
.section .text
.globl hello
hello:

    li a0, 0                # stdout
1:  auipc a1, %pcrel_hi(msg) # load msg(hi)
    addi a1, a1, %pcrel_lo(1b) # load msg(lo)
    li a2, 12               # length
    li a3, 0
    li a7, 64               # _NR_sys_write
    ecall                   # system call

    li a0, 0
    li a1, 0
    li a2, 0
    li a3, 0
    li a7, 93               # _NR_sys_exit
    ecall                   # system call

loop:
    j loop

.section .rodata
msg:
    .string "Hello World\n"

.globl _start
_start:
    call main
```

7.6 Gem5 simulator

Build Gem5 according the following,

https://www.gem5.org/documentation/general_docs/building or <http://learning.gem5.org/book/part1/building.html#requirements-for-gem5>

If you don't have python3.x-config on Ubuntu 18.04 as follows,

```
$ ls /usr/bin/python*
... /usr/bin/python2.7-config
```

Then do install by pip3 as follows¹⁹,

¹⁹ <https://www.anycodings.com/questions/gem5-build-fails-with-embedded-python-library-36-or-newer-required-found-2717>

```
$ sudo apt install python3-pip
$ pip3 install scon
$ ls /usr/bin/python*
... /usr/bin/python3-config
```

After install all dependencies, get gem5 and build RISCv as follows,

```
$ pwd
$ $HOME/riscv/git
$ sudo apt install -y libglib2.0-dev
$ sudo apt install -y libpixman-1-dev
$ git clone https://gem5.googlesource.com/public/gem5
$ cd gem5
$ git log
commit 846dcf0ba4eff824c295f06550b8673ff3f31314
...
$HOME/riscv/git/gem5$ /usr/bin/env python3 $(which scon) ./build/RISCV/gem5.debug -j4
...
$ pwd
$ $HOME/git/lbt/exlbt/riscv
$ $HOME/riscv/git/gem5/build/RISCV/gem5.debug \
$HOME/riscv/git/gem5/configs/example/se.py --cmd=./hello_newlib
**** REAL SIMULATION ****
build/RISCV/sim/simulate.cc:107: info: Entering event queue @ 0. Starting simulation...
hello world!
```

Check cycles as follows,

```
$HOME/git/lbt/exlbt/riscv$ vi m5out/stats.txt
simSeconds      0.000001      # Number of seconds simulated (Second)
simTicks        1229000      # Number of ticks simulated (Tick)
...

$HOME/git/lbt/exlbt/riscv$ $HOME/riscv/git/gem5/build/RISCV/gem5.debug \
/local/git/gem5/configs/example/se.py --cmd=./hello_linux
...
hello world!
...
```

The config of gem5 reference here. http://learning.gem5.org/book/part1/example_configs.html

7.7 GDB

LLVM 13.x fails on “clang -g” for rvv C/C++ file while LLVM 14.x is work. Run qemu on terminal A with gdb on terminal B⁷ as follows,

```
// terminal A:
$ pwd
$ $HOME/git/lbt/exlbt/riscv
$ $HOME/riscv/14.x/riscv_newlib/bin/clang vadd1.c -menable-experimental-extensions \
```

(continues on next page)

⁷ <https://danielmangum.com/posts/risc-v-bytes-qemu-gdb/>

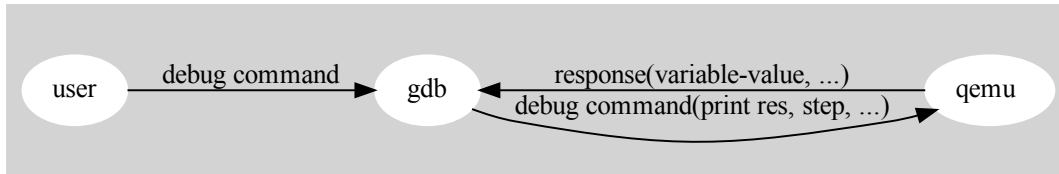


Fig. 7.5: GDB flow

(continued from previous page)

```

-march=rv64gcv1p0 -O0 -g -mabi=lp64d -o a.out -v
$ $HOME/riscv/git/qemu/build/qemu-riscv64 -cpu rv64,v=true -g 1234 a.out
vector version is not specified, use the default value v1.0

// terminal B:
$ pwd
$ $HOME/git/lbt/exlbt/riscv
$ $HOME/riscv/14.x/riscv_newlib/bin/riscv64-unknown-elf-gdb a.out
...
Reading symbols from a.out...
(gdb) target remote :1234
Remote debugging using :1234
0x00000000000010150 in _start ()
(gdb) b vadd1.c:95
Breakpoint 1 at 0x10536: file vadd1.c, line 95.
(gdb) c
Continuing.
Breakpoint 1, main () at vadd1.c:95
95      vOp(a, a, a, array_size(a), VMUL);
(gdb) p a[1]
$1 = 2

// terminal A:
...
v1: 32
array_size(a):4096

a[]: 0 2 4 6 8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40 42 44 46 48 50
52 54 56 58 60 62 64 66 68 70 72 74 76 78 80 82 84 86 88 90 92 94 96 98 100
102 104 106 108 110 112 114 116 118 120 122 124 126 ...
The results of VADD: PASS

// terminal B:
$ ...
(gdb) c

// terminal A:

```

(continues on next page)

(continued from previous page)

```

...
a[]: 0 4 16 36 64 100 144 196 256 324 400 484 576 676 784 900 1024 1156 1296
1444 1600 1764 1936 2116 2304 2500 2704 2916 3136 3364 3600 3844 4096 4356
4624 4900 5184 5476 5776 6084 6400 6724 7056 7396 7744 8100 8464 8836 9216
9604 10000 10404 10816 11236 11664 12100 12544 12996 13456 13924 14400 14884
15376 15876 ...
The results of VMUL: PASS
$

```

Since rvv v1.0 is accepted and v0.10 is draft for release version v1.0, above `-march` change from `rv64gv0p10` of `llvm 13.x` to `rv64gcv1p0` `llvm 14.x`⁸.

Beside GDB runs in the same host environment as your program. When you need more flexibility—for example, running GDB on a physically separate host, or controlling a standalone system over a serial port or a realtime system over a TCP/IP connection⁹. To use a TCP connection, use an argument of the form `host:port`. Above “target remote :1234”, means host and target run on the same host listening port 1234¹⁰.

7.8 RISC-V Calling Convention^{Page 130, 11}

The rv32: register size 32-bit. The rv64 register size 64-bit.

In RV64, 32-bit types, such as `int`, are stored in integer registers as proper sign extensions of their 32-bit values; that is, bits 63..31 are all equal.

Two kinds of ABI, `ilp32` and `lp64`, `-mabi=ilp32`, `ilp32f`, `ilp32d`, `lp64`, `lp64f`, `lp64d`. The ways of pass float arguments on integer/single-float/double-float.

Table 7.2: ABI, caller passing integer/float/double arguments¹²¹³

name	float	double
<code>ilp32/lp64</code>	a registers	a registers
<code>ilp32f/lp64f</code>	fa registers	a registers
<code>ilp32d/lp64d</code>	fa registers	fa registers

`ilp32` and `lp64` are Soft-Float Calling Convention. Soft-Float Calling Convention: Floating-point arguments are passed and returned **in integer registers**, using the rules for integer arguments of the same size.

- `-mabi=ABI-string`

Specify integer and floating-point calling convention. ABI-string contains two parts: the size of integer types and the registers used for floating-point types. For example `‘-march=rv64ifd -mabi=lp64d’` means that ‘long’ and pointers are 64-bit (implicitly defining ‘int’ to be 32-bit), and that floating-point values up to 64 bits wide are passed in F registers. Contrast this with `‘-march=rv64ifd -mabi=lp64f’`, which still allows the compiler to generate code that uses the F and D extensions but only allows floating-point values up to 32 bits long to be passed in registers; or `‘-march=rv64ifd -mabi=lp64’`, in which no floating-point arguments will be passed in registers.

The default for this argument is system dependent, users who want a specific calling convention should specify one explicitly. The valid calling conventions are: ‘`ilp32`’, ‘`ilp32f`’, ‘`ilp32d`’, ‘`lp64`’, ‘`lp64f`’, and ‘`lp64d`’. Some calling conventions are impossible to implement on some ISAs: for example, `‘-march=rv32if -mabi=ilp32d’` is

⁸ <https://github.com/riscv/riscv-v-spec/releases>

⁹ https://ftp.gnu.org/old-gnu/Manuals/gdb/html_chapter/gdb_15.html#SEC125

¹⁰ https://ftp.gnu.org/old-gnu/Manuals/gdb/html_chapter/gdb_15.html#SEC133

¹¹ <https://riscv.org/wp-content/uploads/2015/01/riscv-calling.pdf>

¹² <https://blog.csdn.net/zoomdy/article/details/79353313>

¹³ https://wiki.gentoo.org/wiki/RISC-V_ABIs

invalid because the ABI requires 64-bit values be passed in F registers, but F registers are only 32 bits wide. There is also the ‘ilp32e’ ABI that can only be used with the ‘rv32e’ architecture. This ABI is not well specified at present, and is subject to change¹⁴.

7.9 RVV

Clang/llvm provide RVV (RISC-V Vectors) written in C rather than inline-asm. Though it notices as clang option: `-target-feature +experimental-v`, this way in C is shorter, more user-friendly and easy to remember for users than in inline-asm form. Builtin is C function and friendly either. RVV can be written and run as follows,

exlbt/riscv/vadd2.c

```
// pass:
/* ~/riscv/riscv_newlib/bin/clang++ vadd2.c -menable-experimental-extensions \
-march=rv64gcv0p10 -O0 -mllvm --riscv-v-vector-bits-min=256 -v */
// ~/riscv/git/qemu/build/qemu-riscv64 -cpu rv64,v=true a.out
// ref. https://jia.je/software/2022/01/25/rvv-1.0-toolchain/
//      https://pages.dogdog.run/toolchain/riscv_vector_extension.html

#include <assert.h>
#include <stddef.h>
#include <stdint.h>
#include <stdio.h>
#include <riscv_vector.h>

#define array_size(a) (sizeof(a) / sizeof((a)[0]))

// Vector-vector add
void vadd(uint32_t *a, const uint32_t *b, const uint32_t *c, size_t n) {
    while (n > 0) {
        size_t vl = vsetvl_e32m8(n);
        vuint32m8_t vb = vle32_v_u32m8(b, vl);
// generate:
//      vsetvli zero, a0, e32, m8, ta, mu
//      vadd.vv v8, v8, v16
        vuint32m8_t vc = vle32_v_u32m8(c, vl);
        vuint32m8_t va = vadd(vb, vc, vl);
        vse32(a, va, vl);
        a += vl;
        b += vl;
        c += vl;
        n -= vl;
    }
}

// Vector-vector add (inline assembly)
void vadd_asm(uint32_t *a, const uint32_t *b, const uint32_t *c, size_t n) {
    while (n > 0) {
        size_t vl;
```

(continues on next page)

¹⁴ <https://gcc.gnu.org/onlinedocs/gcc/RISC-V-Options.html>

(continued from previous page)

```

    uint32m8_t va, vb, vc;
    //Fail: __asm__ __volatile__ ( "vsetvli %[vl], %[512], e32, m8" : [vl] "=r"(vl) :
    ↪ [512] "r"(512) );
    vl = vsetvl_e32m8(n);
    #if (__clang_major__ > 10)
        __asm__ __volatile__ ( "vle32.v %[vb], %[b]" : [vb] "=vr"(vb) : [b] "r"(b) :
    ↪ "memory" );
        __asm__ __volatile__ ( "vle32.v %[vc], %[c]" : [vc] "=vr"(vc) : [c] "r"(c) :
    ↪ "memory" );
        __asm__ __volatile__ ( "vadd.vv %[va], %[vb], %[vc]" : [va] "=vr"(va) : [vb] "vr"
    ↪ "(vb), [vc] "vr"(vc) );
        __asm__ __volatile__ ( "vse32.v %[va], %[a]" : [va] "=vr"(va) : [a] "r"(a) :
    ↪ "memory" );
    #else
        __asm__ __volatile__ ( "vle32.v %[vb], %[b]" : [vb] "=v8"(vb) : [b] "r"(b) :
    ↪ "memory" );
        __asm__ __volatile__ ( "vle32.v %[vc], %[c]" : [vc] "=v8"(vc) : [c] "r"(c) :
    ↪ "memory" );
        __asm__ __volatile__ ( "vadd.vv %[va], %[vb], %[vc]" : [va] "=v8"(va) : [vb] "v8"
    ↪ "(vb), [vc] "v8"(vc) );
        __asm__ __volatile__ ( "vse32.v %[va], %[a]" : [va] "=v8"(va) : [a] "r"(a) :
    ↪ "memory" );
    #endif

    a += vl;
    b += vl;
    c += vl;
    n -= vl;
}
}

uint32_t a[4096];
uint8_t m[512];

int main(void) {
    printf("array_size(a):%lu\n", array_size(a));
    // init source
    for (size_t i = 0; i < array_size(a); ++i)
        a[i] = i;

    vadd(a, a, a, array_size(a));

    printf("\na[]: ");
    for (size_t i = 0; i < array_size(a); ++i) {
        if (i < 10)
            printf("%d ", a[i]);
        else if (i == 11)
            printf("...");
        assert(a[i] == i * 2);
    }
    printf("\nThe results of vadd:\tPASS\n");
}

```

(continues on next page)

(continued from previous page)

```

vadd_asm(a, a, a, array_size(a));

printf("\na[]: ");
for (size_t i = 0; i < array_size(a); ++i) {
    if (i < 10)
        printf("%d ", a[i]);
    else if (i == 11)
        printf("...");
    assert(a[i] == i * 4);
}
printf("\nThe results of vadd_asm:\tPASS\n");

return 0;
}

```

```

$ pwd
$ $HOME/git/lbt/exlbt/riscv
$ $HOME/riscv/riscv_newlib/bin/clang vadd2.c -menable-experimental-extensions \
  -march=rv64gcv0p10 -O0 -mllvm --riscv-v-vector-bits-min=256
$ $HOME/riscv/git/qemu/build/qemu-riscv64 -cpu rv64,v=true a.out
vector version is not specified, use the default value v1.0
array_size(a):4096

a[]: 0 2 4 6 8 10 12 14 16 18 ...
The results of vadd: PASS

a[]: 0 4 8 12 16 20 24 28 32 36 ...
The results of vadd_asm: PASS

$ $HOME/riscv/riscv_newlib/bin/clang vadd1.c -menable-experimental-extensions \
  -march=rv64gcv0p10 -O0 -mllvm --riscv-v-vector-bits-min=256 -static
$ $HOME/riscv/git/qemu/build/qemu-riscv64 -cpu rv64,v=true a.out
vector version is not specified, use the default value v1.0
1 11 11 11 11 11 11 11 11 1
$ $HOME/riscv/riscv_newlib/bin/riscv64-unknown-elf-objdump -d a.out|grep vadd.vv
106fc: 03ae0d57 vadd.vv v26,v26,v28

```

For -march=rv64imfv0p10zfh0p1,

- v0p10: vector version 0.10.
- zfh0p1: “Zfh” 0.1 version^{Page 118, 4}.

For -mabi, as the section above.

Clang/llvm provide builtin and intrinsic functions to implement RVV (RISC-V Vectors)

`$HOME/riscv/git/llvm-project/clang/include/clang/Basic/riscv_vector.td`

```
#define vsetvl_e8mf8(avl) __builtin_rvv_vsetvli((size_t)(avl), 0, 5)
...
defm vmadd : RVVIntTerBuiltinSet;
...
defm vfddiv : RVVFloatingBinBuiltinSet;
```

`$HOME/riscv/git/llvm-project/llvm/include/llvm/IR/IntrinsicsRISCV.td`

```
def int_riscv_vsetvli : Intrinsic<...
...
defm vmadd : RISCVternaryAAXA;
...
defm vfddiv : RISCVBinaryAAX;
```

Refer to clang/llvm test cases in the following folders.

```
$HOME/riscv/git/llvm-project/clang/test/CodeGen/RISCV/rvv-intrinsics$ ls
... vfddiv.c ... vfmadd.c

$HOME/riscv/git/llvm-project/llvm/test/CodeGen/RISCV/rvv$ ls
... vfddiv-rv64.ll ... vfmadd-rv64.ll ...
```

7.10 Atomic instructions

RISCV atomic instructions²⁰.

7.11 RISCV+NPU for Deep Learning

<https://github.com/Jonathan2251/lbt/blob/master/present/AnLLVMBasedNPUCompiler.pdf>

7.12 Resources

7.12.1 FreeBSD

FreeBSD RISCV²¹. FreeBSD²².

²⁰ Chapter 8 of Volume 1, Unprivileged Spec v. 20191213 <https://five-embeddev.com/riscv-isa-manual/latest/a.html>

²¹ <https://www.freebsd.org/platforms/>

²² <https://github.com/freebsd>

7.12.2 FreeRTOS

Code²³²⁴. Documents²⁵²⁶.

7.12.3 Zephyr

RISC-V Boards²⁷.

7.12.4 Andes

Amazon-FreeRTOS²⁸, Xilinx has RISC-V inside in MicroZed²⁹ and vivado³⁰.

Zephyr³¹.

7.13 Websites

<https://twilco.github.io/riscv-from-scratch/2019/04/27/riscv-from-scratch-2.html>

<https://twilco.github.io>

7.14 To do:

As `exlbt/riscv/riscv-toolchain-setup-macos-intel.sh`. Currently, this script complete build on both `riscv_newlib` and `riscv_linux`. However, `qemu` does not create `qemu-riscv64` for baremetal on macos. So cannot verify it yet.

- How to run `qemu-system-riscv64` as baremetal `qemu-riscv64`? Following <https://twilco.github.io/riscv-from-scratch/2019/04/27/riscv-from-scratch-2.html#link-it-up> to create `crt0.S` and `riscv64-virt.ld` but not work. It fails without changing `crt0.S` either.

```
// terminal A:
riscv % ~/riscv/14.x/riscv_newlib/bin/clang hello.c -menable-experimental-extensions \
-march=rv64gcvlp0 -O0 -mabi=lp64d -T riscv64-virt.ld -Wl,--no-warn-rwx-segment -
↪nostartfiles crt0.S -v
riscv % ~/riscv/git/qemu/build/qemu-system-riscv64 -machine virt -m 128M -cpu \
rv64,v=true -gdb tcp::1234 -kernel a.out

// terminal B:
riscv % ~/riscv/14.x/riscv_newlib/bin/riscv64-unknown-elf-gdb a.out
...
```

(continues on next page)

²³ In <https://github.com/FreeRTOS/FreeRTOS/tree/main/FreeRTOS/Demo>, directories RISC-V*

²⁴ <https://github.com/FreeRTOS/FreeRTOS-Kernel.git>

²⁵ <https://www.freertos.org/Using-FreeRTOS-on-RISC-V.html>

²⁶ https://www.freertos.org/fr-content-src/uploads/2018/07/161204_Mastering_the_FreeRTOS_Real_Time_Kernel-A_Hands-On_Tutorial_Guide.pdf

²⁷ <https://docs.zephyrproject.org/3.1.0/boards/riscv/index.html>

²⁸ <https://github.com/andestech/amazon-freertos>

²⁹ <https://www.xilinx.com/about/blogs/adaptable-advantage-blog/2021/MicroZed-Chronicles-Bluespec-RISC-V.html>

³⁰ <https://github.com/eugene-tarassov/vivado-risc-v>

³¹ Andes Zephyr port supports SMP (Symmetric Multi-Processing) and has been verified on Andes RISC-V multicore. <https://www.globenewswire.com/news-release/2021/04/23/2216131/0/en/Andes-Announces-the-New-Upgrade-of-AndeSight-IDE-v5-0-a-comprehensive-software-solution-to-acc.html>

(continued from previous page)

```
Reading symbols from a.out...
(gdb) target remote :1234
Remote debugging using :1234
0x00000000080000408 in ?? ()
(gdb) c
Continuing.
```

qemu-system-riscv64 with -nographic fail too.

APPENDIX B: CLANG LIBRARY AND TOOLS

- *Install llvm with clang tools*
- *Using Clang as a Library*
 - *Clang Plugins*
 - *Using LibTooling and LibASTMatchers*
- *Using Clang as Tools*
 - *clang-query*

This chapter shows that Using Clang as a Library and Tools [[clang-doc](#)].

8.1 Install llvm with clang tools

exlbt/clang-tools/install-llvm.sh

```
#!/usr/bin/env bash

#export LLVM_VER=14.x
#export LLVM_VER=15.x
export LLVM_VER=16.x

export LLVM_SRC_DIR=$HOME/llvm/$LLVM_VER

get_llvm() {
    if [ ! -d "$LLVM_SRC_DIR" ]; then
        echo "LLVM_SRC_DIR: $LLVM_SRC_DIR not exist"
        exit 1
    fi
    pushd $LLVM_SRC_DIR
    git clone https://github.com/llvm/llvm-project.git
    cd llvm-project
    git checkout -b $LLVM_VER origin/release/$LLVM_VER
    popd
}

release_build_llvm_toolchain() {
```

(continues on next page)

(continued from previous page)

```
pushd $LLVM_SRC_DIR/llvm-project
if [ -d "$LLVM_SRC_DIR/llvm-project/release-build" ]; then
    echo "$LLVM_SRC_DIR/build exist already. Please remove it before run this bash"
    exit 1
fi
mkdir release-build
cd release-build
cmake -DCMAKE_BUILD_TYPE=Release -DLLVM_ENABLE_PROJECTS="clang;clang-tools-extra" \
-DLLVM_PARALLEL_COMPILE_JOBS=4 -DLLVM_PARALLEL_LINK_JOBS=1 -G "Ninja" ../llvm \
-DCLANG_BUILD_EXAMPLES=ON
ninja
popd
}

debug_build_llvm_toolchain() {
    pushd $LLVM_SRC_DIR/llvm-project
    if [ -d "$LLVM_SRC_DIR/llvm-project/build" ]; then
        echo "$LLVM_SRC_DIR/build exist already. Please remove it before run this bash"
        exit 1
    fi
    mkdir build
    cd build
    cmake -DCMAKE_BUILD_TYPE=Debug -DLLVM_ENABLE_PROJECTS="clang;clang-tools-extra" \
-DLLVM_PARALLEL_COMPILE_JOBS=4 -DLLVM_PARALLEL_LINK_JOBS=1 -G "Ninja" ../llvm \
-DCLANG_BUILD_EXAMPLES=ON
    ninja
    popd
}

get_llvm;
release_build_llvm_toolchain;
debug_build_llvm_toolchain;
```

```
$ pwd
$ $HOME/lbt/exlbt/clang-tools
$ bash install-llvm.sh
```

8.2 Using Clang as a Library

8.2.1 Clang Plugins

References `clang/examples/PrintFunctionNames` [[clang-plugins](#)] [[clang-ast-fa](#)].

exlbt/clang-tools/Rewriter/plugin/README.txt

Clang example:

- Converter from cpp to cpp.
- Modified from clang/examples/PrintFunctionNames.
- Work on llvm 088f33605d8a61ff519c580a71b1dd57d16a03f8 of 15.x,
f28c006a5895fc0e329fe15fead81e37457cb1d1 of 14.x and
e8a397203c67adbeae04763ce25c6a5ae76af52c of 12.x.

Install:

```
~/git/lbt/exlbt/clang-tools/Rewriter/plugin$ cp -rf Rewriter ~/llvm/15.x/llvm-project/
↪ clang/examples/.
echo 'add_subdirectory(Rewriter)' >> ~/llvm/15.x/llvm-project/clang/examples/
↪ CMakeLists.txt
```

Run command:

Check Rewriter/README.txt

exlbt/clang-tools/Rewriter/plugin/Rewriter/README.txt

This is a simple example demonstrating how to use clang's facility for providing AST consumers using a plugin.

Once the plugin is built, you can run it using:

--

Linux:

```
$ <llvm-build-path>/bin/clang++ -Xclang -load -Xclang <llvm-build-path>/lib/Rewriter.
↪ so -Xclang -add-plugin -Xclang ex1-act -Xclang -emit-llvm -S input.cpp -o -
for example:
~/llvm/15.x/llvm-project/build/bin/clang++ -Xclang -load -Xclang ~/llvm/15.x/llvm-
↪ project/build/lib/Rewriter.so -Xclang -add-plugin -Xclang ex1-act -Xclang -ast-dump -
↪ fsyntax-only input.cpp -o -
```

MacOS:

Use Rewriter.dylib instead of .so.

Rewriter/exe/Linux and Rewriter/exe/macOS applies on Linux and macOS as follows,

exlbt/clang-tools/Rewriter/exe/Linux/README.txt

Clang example:

- Converter from cpp to cpp
- Work on llvm 088f33605d8a61ff519c580a71b1dd57d16a03f8 of 15.x.

build:

```
exe/Linux$ make clean; make
```

run:

```
./build/Rewriter ../input.cpp
```

check:

```
cat output.cpp
```

(continues on next page)

(continued from previous page)

```
gdb:
exe/Linux$ gdb --args ./build/Rewriter ../input.cpp
(gdb) b MyASTConsumer.cpp:61
(gdb) r
61          TheRewriter.ReplaceText(CallerSR, StringRef(sBuiltinFunc));
```

exlbt/clang-tools/Rewriter/exe/macOS/README.txt

```
Clang example:
- Converter from cpp to cpp
- Work on llvm 088f33605d8a61ff519c580a71b1dd57d16a03f8 of 15.x.

install:
% brew install llvm
build:
exe/macOS% make clean; make
run:
./build/Rewriter ../input.cpp
check:
cat output.cpp
gdb:
exe/macOS% gdb --args ./build/Rewriter ../input.cpp
(gdb) b MyASTConsumer.cpp:61
(gdb) r
61          TheRewriter.ReplaceText(CallerSR, StringRef(sBuiltinFunc));
```

8.2.2 Using LibTooling and LibASTMatchers

References clang-tools-extra/loop-convert [clang-lamt] [clang-lam].

exlbt/clang-tools/loop-convert2/README.txt

```
Work on llvm 088f33605d8a61ff519c580a71b1dd57d16a03f8 of 15.x
and f28c006a5895fc0e329fe15fead81e37457cb1d1 of 14.x.

install:
~/git/lbt/exlbt/clang-tools$ cp -rf loop-convert2 ~/llvm/15.x/llvm-project/clang-tools-
↳ extra/.
echo 'add_subdirectory(loop-convert2)' >> ~/llvm/15.x/llvm-project/clang-tools-extra/
↳ CMakeLists.txt

build:
~/llvm/14.x/llvm-project/build$ ninja
run:
~/git/lbt/exlbt/clang-tools$ ~/llvm/15.x/llvm-project/build/bin/loop-convert2 loop-
↳ test.cpp --
Potential array-based loop discovered.
ForStmt begin at 8:3
```

(continues on next page)

(continued from previous page)

```

ForStmt end at 10:3
~/git/lbt/exlbt/clang-tools$ ~/llvm/15.x/llvm-project/build/bin/loop-convert2 -debug_
↪ loop-test.cpp --
Potential array-based loop discovered.
ForStmt 0x1378c7018
|-DeclStmt 0x1378c6dc0
|  |-VarDecl 0x1378c6d38  used i 'int' cinit
|  |-IntegerLiteral 0x1378c6da0 'int' 0
|  |-<<<NULL>>>
|-BinaryOperator 0x1378c6ed8 '_Bool' '<'
|  |-ImplicitCastExpr 0x1378c6ec0 'int' <LValueToRValue>
|  |  |-DeclRefExpr 0x1378c6dd8 'int' lvalue Var 0x1378c6d38 'i' 'int'
|  |  |-CallExpr 0x1378c6ea0 'int'
|  |  |-ImplicitCastExpr 0x1378c6e88 'int (*)(void)' <FunctionToPointerDecay>
|  |  |-DeclRefExpr 0x1378c6e40 'int (void)' lvalue Function 0x1378c6a40 'expr' 'int_
↪ (void)'
|-UnaryOperator 0x1378c6f18 'int' lvalue prefix '++'
|  |-DeclRefExpr 0x1378c6ef8 'int' lvalue Var 0x1378c6d38 'i' 'int'
|-CompoundStmt 0x1378c7000
  |-BinaryOperator 0x1378c6fe0 'int' lvalue '='
  |  |-DeclRefExpr 0x1378c6f30 'int' lvalue Var 0x1378c6c80 'res' 'int'
  |  |-BinaryOperator 0x1378c6fc0 'int' '+'
  |  |  |-ImplicitCastExpr 0x1378c6f90 'int' <LValueToRValue>
  |  |  |  |-DeclRefExpr 0x1378c6f50 'int' lvalue Var 0x1378c6c80 'res' 'int'
  |  |  |  |-ImplicitCastExpr 0x1378c6fa8 'int' <LValueToRValue>
  |  |  |  |-DeclRefExpr 0x1378c6f70 'int' lvalue Var 0x1378c6d38 'i' 'int'
ForStmt begin at 8:3
ForStmt end at 10:3

```

8.3 Using Clang as Tools

8.3.1 clang-query

References¹²³. RVV reference `~/riscv/riscv_newlib/lib/clang/13.0.1/include/riscv_vector.h`.

exlbt/clang-tools/vecotor-dsl.cpp

```

/*
~/riscv/riscv_newlib/bin/clang++ vector-dsl.cpp -menable-experimental-extensions \
-march=rv64gcv0p10 -O0 -mllvm --riscv-v-vector-bits-min=256 -DUSE_RVV -v
~/riscv/git/qemu/build/qemu-riscv64 -cpu rv64,v=true a.out
~/riscv/riscv_newlib/bin/riscv64-unknown-elf-objdump -d a.out|grep vmul

The following work for #undef USE_RVV
~/llvm/15.x/llvm-project/build/bin/clang++ vector-dsl.cpp -emit-llvm -S -Xclang -ast-dump

```

(continues on next page)

¹ <https://clang.llvm.org/docs/LibASTMatchersTutorial.html>

² <https://clang.llvm.org/docs/LibASTMatchersReference.html>

³ <https://clang.llvm.org/docs/LibASTMatchersReference.html#narrowing-matchers>

(continued from previous page)

```
~/llvm/15.x/llvm-project/build/bin/clang-query vector-dsl.cpp --
clang-query> set traversal IgnoreUnlessSpelledInSource
clang-query> m cxxOperatorCallExpr(binaryOperation(hasOperatorName("="),
↳ hasLHS(expr(hasType(cxxRecordDecl(hasName("UVec32")))).bind("lhs")),
↳ hasRHS(expr(cxxOperatorCallExpr(binaryOperation(hasOperatorName("+"),
↳ hasLHS(expr(cxxOperatorCallExpr(binaryOperation(hasOperatorName("*"),hasLHS(expr().
↳ bind("lhs2")),hasRHS(expr().bind("rhs2")))).bind("lhs1")),hasRHS(expr().bind("rhs1
↳ ))))).bind("rhs"))))
```

Match #1:

```
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:3: note: "lhs" binds here
  A = alpha*B + C;
  ^
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:7: note: "lhs1" binds here
  A = alpha*B + C;
    ^~~~~~
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:7: note: "lhs2" binds here
  A = alpha*B + C;
    ^~~~~
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:7: note: "rhs" binds here
  A = alpha*B + C;
    ^~~~~~
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:17: note: "rhs1" binds here
  A = alpha*B + C;
                ^
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:13: note: "rhs2" binds here
  A = alpha*B + C;
                ^
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:3: note: "root" binds here
  A = alpha*B + C;
  ^~~~~~
1 match.
```

More:

```
clang-query> m cxxOperatorCallExpr(binaryOperation(hasOperatorName("="),
↳ hasLHS(expr(hasType(cxxRecordDecl(hasName("UVec32")))).bind("lhs")),
↳ hasRHS(expr(cxxOperatorCallExpr(binaryOperation(hasOperatorName("+"),hasLHS(expr().
↳ bind("lhs1")),hasRHS(expr().bind("rhs1")))).bind("rhs"))))
...

```

Match #2:

```
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:3: note: "lhs" binds here
  A = alpha*B + C;
  ^
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:7: note: "lhs1" binds here
  A = alpha*B + C;
    ^~~~~~
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:7: note: "rhs" binds here
  A = alpha*B + C;
```

(continues on next page)

(continued from previous page)

```

^~~~~~
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:17: note: "rhs1" binds here
  A = alpha*B + C;
      ^
/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:3: note: "root" binds here
  A = alpha*B + C;
  ^~~~~~

clang-query> m cxxOperatorCallExpr(binaryOperation(hasOperatorName("*"),hasLHS(expr()).
↳bind("lhs2")),hasRHS(expr().bind("rhs2"))))
...
Match #4:

/home/cschen/git/lbt/exlbt/clang-tools/vector-dsl.cpp:154:7: note: "lhs2" binds here
  A = alpha*B + C;
      ^~~~~~
...

Ref. https://clang.llvm.org/docs/LibASTMatchersReference.html#narrowing-matchers
*/

#include "vector-dsl.h"
#include <assert.h>
#include <stddef.h>
#include <stdio.h>

#ifdef USE_RVV
#include <riscv_vector.h>
#endif

#define array_size(a) (sizeof(a) / sizeof((a)[0]))

const Precision bit16 = Bit16;

uint32_t t[33] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
↳24, 25, 26, 27, 28, 29, 30, 31, 32,
};

UVec32::UVec32(uint32_t *A, size_t aSize) {
    data = A;
    size = aSize;
}

UVec32& UVec32::Mul(const uint32_t Scalar) {
    static UVec32 res(t, size);
    size_t n = size;
    uint32_t *a = res.data;
    uint32_t *b = data;
#ifdef USE_RVV
    while (n > 0) {
        size_t vl = vsetvl_e32m8(n);

```

(continues on next page)

(continued from previous page)

```

    vuint32m8_t vb = vle32_v_u32m8(b, vl);
    vuint32m8_t va = vmul(vb, Scalar, vl);
    vse32(a, va, vl);
    a += vl;
    b += vl;
    n -= vl;
}
#else
    for (int i=0; i < size; i++) {
        a[i] = Scalar*b[i];
    }
#endif
    return res;
}

UVec32& UVec32::Mul(const UVec32 &arg) {
    static UVec32 res(t, size);
    size_t n = size;
    uint32_t *a = res.data;
    uint32_t *b = data;
    uint32_t *c = arg.data;
    assert(size == arg.size);
#ifdef USE_RVV
    while (n > 0) {
        size_t vl = vsetvl_e32m8(n);
        vuint32m8_t vb = vle32_v_u32m8(b, vl);
        vuint32m8_t vc = vle32_v_u32m8(c, vl);
        vuint32m8_t va = vmul(vb, vc, vl);
        vse32(a, va, vl);
        a += vl;
        b += vl;
        c += vl;
        n -= vl;
    }
#else
    for (int i=0; i < size; i++) {
        a[i] = b[i]*c[i];
    }
#endif
    return res;
}

UVec32& UVec32::operator*(const uint32_t Scalar) {
    return Mul(Scalar);
}

UVec32& UVec32::operator*(const UVec32 &arg) {
    return Mul(arg);
}

UVec32& UVec32::operator+(const UVec32 &arg) {
    static UVec32 res(t, size);

```

(continues on next page)

(continued from previous page)

```

size_t n = size;
uint32_t *a = res.data;
uint32_t *b = data;
uint32_t *c = arg.data;
assert(size == arg.size);
#ifdef USE_RVV
while (n > 0) {
    size_t vl = vsetvl_e32m8(n);
    vuint32m8_t vb = vle32_v_u32m8(b, vl);
    vuint32m8_t vc = vle32_v_u32m8(c, vl);
    vuint32m8_t va = vadd(vb, vc, vl);
    vse32(a, va, vl);
    a += vl;
    b += vl;
    c += vl;
    n -= vl;
}
#else
for (int i=0; i < size; i++) {
    a[i] = b[i]+c[i];
}
#endif
return res;
}

void UVec32::Print() {
    for (int i=0; i < size; i++) {
        printf("%u ", data[i]);
    }
    printf("\n");
}

UVec32& operator*(const uint32_t Scalar, UVec32 &B) {
    return B * Scalar;
}

uint32_t gV1[33] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
    ↪ 24, 25, 26, 27, 28, 29, 30, 31, 32,
};

uint32_t gV2[33] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
    ↪ 24, 25, 26, 27, 28, 29, 30, 31, 32,
};

uint32_t gV3[33] = {
    0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23,
    ↪ 24, 25, 26, 27, 28, 29, 30, 31, 32,
};

int main() {

```

(continues on next page)

(continued from previous page)

```
UVec32 A(gV1,array_size(gV1)), B(gV2,array_size(gV2)), C(gV3,array_size(gV3));
uint32_t alpha = (uint32_t)2;
A = B + C;
A = B * C;
A = alpha*B;
A = alpha*B + C;
uint32_t a = 0; uint32_t b = 2; uint32_t c = 3;
a = alpha*b + c;
printf("B: "); B.Print();
printf("C: "); C.Print();
printf("A: "); A.Print();
printf("\n\n");

// multiply and add, madd
// A = B * C + D;
// Solve above by class is not efficient, class does not use madd.
// DSL: extend clang to parsing the left and calling A = madd(B,C,D); can get better
↪performance.

return 0;
}
```

- References⁴.

⁴ <https://www.youtube.com/watch?v=-iOtb6luK1Q>

APPENDIX C: POCL

- *OpenCL*
- *Run OpenCL*
 - *Macos*
- *Open sources*
- *POCL*
- *Structure*
- *GDB on POCL*
- *Examples of Compiling and running on POCL*
- *RISCV OpenCL*
- *SYCL*
- *DPC++*

As Fig. 9.1, one possible HW platform is that one Host + Device with 2 RISCV processors and each RISCV has each dedicated memory.

- `info::local_mem_type::global` -> coherent, CPU cache (SRAM)
- `info::local_mem_type::local` -> dedicated local memory, GPU cache (SRAM)

OpenCL includes Runtime API, Driver and Compiler three components. POCL is an open source implementation for Runtime API.

OpenCL Implementation¹. Wiki OpenCL Open source implementations².

¹ <https://www.iwocl.org/resources/opencl-implementations/>

² https://en.wikipedia.org/wiki/OpenCL#Open_source_implementations

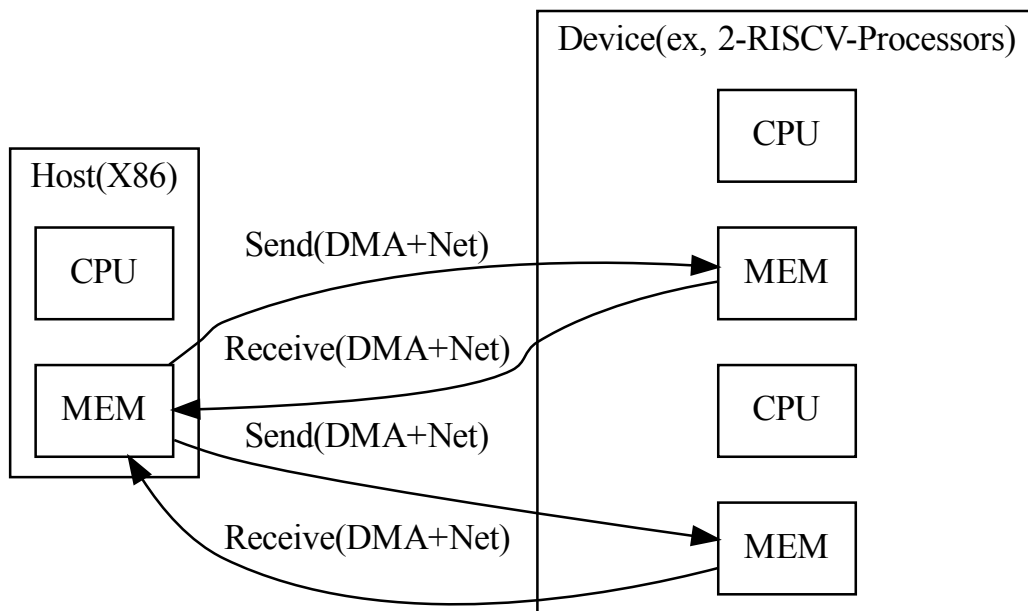


Fig. 9.1: OpenCL with discrete memory (Device: 2 processor+memory)

9.1 OpenCL

Builtin-function and builtin-kernel reference here⁶. Books⁷. Papers⁸⁹¹⁰¹¹¹² in lbt/papers/pocl.

9.2 Run OpenCL

9.2.1 MacOS

~/git/lbt/exlbt/openc/opencv-saxpy/opencv-saxpy.xcodeproj can be created by the following:

Xcode – Create a new Xcode project – macOS – Command Line Tool – Product Name: opencv-saxpy, Language: C – Next – Create – “Choose folder ~/git/lbt/exlbt/openc/” – on left opencv-saxpy folder: delete C main – Move to trash – “copy saxpy.c to ~/git/lbt/exlbt/openc/opencv-saxpy/opencv-saxpy” – on left opencv-saxpy folder: right click mouse “Add files to “opencv-saxpy”” – save

Build and run saxpy opencv as follows,

Open ~/git/lbt/exlbt/openc/opencv-saxpy/opencv-saxpy.xcodeproj – Product – Build – Product – Run, then choose the icons of the following Fig. 9.2 and get the compile options and run-result.

9.3 Open sources

Table 9.1: Open sources for OpenCL

Project	Runtime	Driver	Compiler	Library
POCL	V	V		
clang			V	
libclc				V

Table 9.2: Open sources for OpenCL 2

project	Host Compiler	Device Compiler	Device Lib
POCL	X86	X86,ARM,AMD,TCE,PTX	
clang	X86	NVIDIA-gpu, AMD-gpu	
libclc			Spir, Spir64, NVIDIA-gpu, AMD-gpu

libclc is an open source, BSD/MIT dual licensed implementation of the library requirements of the OpenCL C programming language³.

⁶ <http://jonathan2251.github.io/lbt/clang.html#builtin-functions>

⁷ <https://dahlan.unimal.ac.id/files/ebooks2/2015%203rd%20Heterogeneous%20Computing%20with%20OpenCL%202.0.pdf>

⁸ <https://mlsys.org/Conferences/doc/2018/84.pdf>

⁹ https://www.researchgate.net/publication/350512301_Automatic_Graph_Partitioning_for_Very_Large-scale_Deep_Learning

¹⁰ <https://easychair.org/publications/preprint/GjhX>

¹¹ https://www.researchgate.net/publication/221084751_A_comprehensive_performance_comparison_of_CUDA_and_OpenCL

¹² <https://arxiv.org/pdf/2002.03794.pdf>

³ <https://libclc.lvm.org>

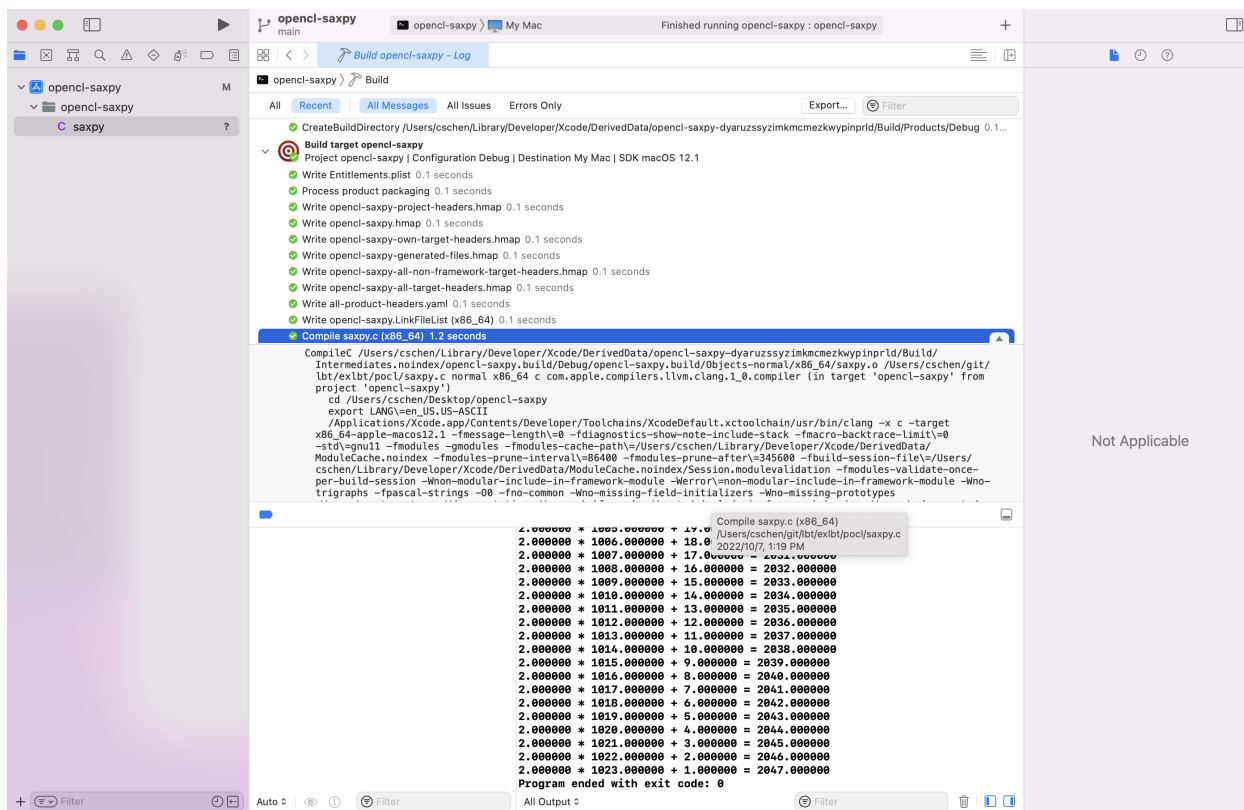


Fig. 9.2: Build and run openc1-saxpy.xcodeproj

9.4 POCL

Pocl web⁴ and documentation⁵.

PoCL uses Clang as an OpenCL C frontend and LLVM for kernel compiler implementation, and as a portability layer. Thus, if your desired target has an LLVM backend, it should be able to get OpenCL support easily by using PoCL^{Page 151, 4}.

Note: OpenCL is C function with keyword `__kernel` and some attributes in data type. If clang is able to compile `input.cl` to `input.ll`, then llvm backend can compile `input.ll` and link to library.

Build as the following bash:

exlbt/pocl/pocl-install.sh

```
#!/usr/bin/env bash

# add apt repo at first time
# sudo apt-add-repository 'deb https://apt.llvm.org/bionic/   llvm-toolchain-bionic-13.
↪main'

# Set POCL_PARENT_DIR for the parent folder of pocl git hub checkout to.
export POCL_PARENT_DIR=$HOME/git

# After "sudo make install", pocl installed in /usr/local/share/pocl,
# /usr/local/lib/libpocl.so, /usr/local/lib/pocl and /usr/local/bin/poclcc.

# Ubuntu 18.04 only can use LLVM_VERSION 13. 14 is too new for the dependent
# packages of Ubuntu 18.04.
export LLVM_VERSION=13
export LLVM_PATH=/usr/lib/llvm-13/bin
#Not work for the following
#LLVM_PATH=$HOME/llvm/13/llvm-project/build/bin

# Ubuntu 22.04 only can use LLVM_VERSION 14. 13 has no /usr/lib/llvm-13/clang
# after install_dependencies().
#LLVM_VERSION=14
#LLVM_PATH=/usr/lib/llvm-14/bin

# Todo:
# Trace test_clCreateKernel.c and test_enqueue_kernel_from_binary.c for running an.
↪OpenCL example.

install_dependencies() {
    echo "LLVM_VERSION: $LLVM_VERSION"
    sudo apt-get install -y build-essential ocl-icd-libopencl1 cmake git pkg-config \
    libclang-${LLVM_VERSION}-dev clang-${LLVM_VERSION} llvm-${LLVM_VERSION} make ninja-
↪build \
    ocl-icd-libopencl1 ocl-icd-dev ocl-icd-opencl-dev libhwloc-dev zlib1g \
```

(continues on next page)

⁴ <http://portablecl.org/>

⁵ <http://portablecl.org/docs/html/>

(continued from previous page)

```

    zlib1g-dev clinfo dialog apt-utils libxml2-dev libclang-cpp${LLVM_VERSION}-dev \
    libclang-cpp${LLVM_VERSION} llvm-${LLVM_VERSION}-dev
}

get_pocl() {
    pushd $POCL_PARENT_DIR
    git clone https://github.com/pocl/pocl
    cd pocl
#   git checkout 4627171d40543091e399989c277faa52fcee0ff8
    popd
}

check() {
    if [ ! -d "$POCL_PARENT_DIR" ]; then
        echo "POCL_PARENT_DIR: $POCL_PARENT_DIR not exist"
        exit 1
    fi
}

build_pocl() {
    pushd $POCL_PARENT_DIR/pocl
    mkdir build
    cd build
#   The default uses /usr/bin/cc in ubuntu 18.04
#   cmake -WITH_DLLVM_CONFIG=${LLVM_PATH}/llvm-config ..
#   Have verified the following using clang compiler
    cmake -DWITH_LLVM_CONFIG=${LLVM_PATH}/llvm-config \
        -DENABLE_ICD=OFF \
        -DCMAKE_C_COMPILER=${LLVM_PATH}/clang \
        -DCMAKE_CXX_COMPILER=${LLVM_PATH}/clang++ ..
    make
    sudo make install
    popd
}

# Verify tests/runtime/test_clCreateKernel.c using clang rather than cc as follows,
# jonathanchen@hz-compiler1:~/git/pocl/build$ touch ../tests/runtime/test_clCreateKernel.
# ↪ c
# jonathanchen@hz-compiler1:~/git/pocl/build$ make VERBOSE=1 |grep test_clCreateKernel.c
# [ 95%] Building C object tests/runtime/CMakeFiles/test_clCreateKernel.dir/test_
# ↪ clCreateKernel.c.o
# cd /home/jonathanchen/git/pocl/build/tests/runtime && /usr/lib/llvm-13/bin/clang ...
# -c /home/jonathanchen/git/pocl/build/tests/runtime/test_clCreateKernel.c

# http://portablecl.org/docs/html/development.html
check_pocl() {
    pushd $POCL_PARENT_DIR/pocl/build
    make check_tier1
    popd
}

install_dependencies;

```

(continues on next page)

(continued from previous page)

```
get_pocl;
check;
build_pocl;
check_pocl;
```

Add apt repo on Ubuntu at first time,

```
$ sudo apt-add-repository 'deb https://apt.llvm.org/bionic/ llvm-toolchain-bionic-13_
↪main'
$ grep "apt.llvm" /etc/apt/sources.list /etc/apt/sources.list.d/*
/etc/apt/sources.list:deb http://apt.llvm.org/bionic/ llvm-toolchain-bionic-13 main
/etc/apt/sources.list:# deb-src http://apt.llvm.org/bionic/ llvm-toolchain-bionic-13 main
/etc/apt/sources.list:deb https://apt.llvm.org/bionic/ llvm-toolchain-bionic-13 main
/etc/apt/sources.list:# deb-src https://apt.llvm.org/bionic/ llvm-toolchain-bionic-13_
↪main
```

Reference¹³.

9.5 Structure

Code of Runtime in pocl/lib/CL. Test cases of Runtime in pocl/tests/runtime.

9.6 GDB on POCL

After `bash pocl-install.sh`, can run `make check_tier1` but fail on run single test such as `./tests/runtime/test_clCreateKernelsInProgram` as follows,

```
~/git/pocl/build$ POCL_DEBUG=all ./tests/runtime/test_clCreateKernelsInProgram
...
[2022-07-15 08:43:44.140733258219579]POCL: in fn pocl_init_devices at line 529:
| WARNING | cschen:Loading /home/cschen/git/pocl/build/lib/CL/pocl/libpocl-devices-
↪basic.so failed: /home/cschen/git/pocl/build/lib/CL/pocl/libpocl-devices-basic.so:
↪cannot open shared object file: No such file or directory
...
[2022-10-06 08:48:06.140725365544843]POCL: in fn pocl_init_devices at line 584:
| ERROR | CL_DEVICE_NOT_FOUND no devices found. POCL_DEVICES=(null)
CL_DEVICE_NOT_FOUND in poclu_get_any_device on line 22
```

Fail to run single test of executable file is not acceptable for tracing code of pocl. Workaround fix as follows,

```
~/git/pocl/build$ touch ../tests/runtime/test_clCreateKernelsInProgram.c
~/git/pocl/build$ make VERBOSE=1|grep test_clCreateKernelsInProgram
...
/usr/lib/llvm-13/bin/clang -g -pie CMakeFiles/test_clCreateKernelsInProgram.dir/test_
↪clCreateKernelsInProgram.c.o -o test_clCreateKernelsInProgram -Wl,-rpath,/home/cschen/
↪git/pocl/build/lib/CL ../pocl/libpocl.a ../lib/CL/libOpenCL.so.2.9.0 -L/usr/
↪lib/x86_64-linux-gnu -lhwloc /usr/lib/llvm-13/lib/libclang-cpp.so /usr/lib/llvm-13/lib/
↪libLLVM-13.so -lrt -lm -ldl -lm -ldl
```

¹³ <http://portablecl.org/docs/html/install.html#configure-build>

Change link path *rpath* from `/home/cschen/git/pocl/build/lib/CL` to `/usr/local/lib` for `libpocl-devices-basic.so` then PASS as follows,

```
~/git/pocl/build$ cd /home/jonathanchen/git/pocl/build/tests/runtime
~/git/pocl/build/tests/runtime$ /usr/lib/llvm-13/bin/clang -g -pie CMakeFiles/test_
↪clCreateKernelsInProgram.dir/test_clCreateKernelsInProgram.c.o -o test_
↪clCreateKernelsInProgram -Wl,-rpath,/usr/local/lib ../../pocl/libpocl.a ../../lib/
↪CL/libOpenCL.so.2.9.0 -lhwloc /usr/lib/llvm-13/lib/libclang-cpp.so /usr/lib/llvm-13/
↪lib/libLLVM-13.so -lrt -lm -ldl -lm -pthread -ldl
~/git/pocl/build/tests/runtime$ cd ../../
~/git/pocl/build$ ./tests/runtime/test_clCreateKernelsInProgram
Hello
World
```

Then I can use `gdb` as follows,

```
~/git/pocl/build/$ gdb --args ./tests/runtime/test_clCreateKernelsInProgram
(gdb) b test_clCreateKernelsInProgram.c:21
Breakpoint 1 at 0x23b4: file /home/cschen/git/pocl/tests/runtime/test_
↪clCreateKernelsInProgram.c, line 21.
(gdb) r
...
Breakpoint 1, main (argc=1, argv=0x7fffffffdfa8) at /home/cschen/git/pocl/tests/runtime/
↪test_clCreateKernelsInProgram.c:23
21          err = poclu_get_any_device(&ctx, &did, &queue);
```

9.7 Examples of Compiling and running on POCL

As tracing clang compilation options in last section, I add `compile.sh` for running OpenCL program on pocl as follows,

`exlbt/pocl/ex/compile.sh`

```
#!/usr/bin/env bash

FILE=test_clCreateKernelsInProgram
#FILE=saxpy

/usr/lib/llvm-13/bin/clang -DCL_HPP_TARGET_OPENCL_VERSION=300 -DCL_TARGET_OPENCL_
↪VERSION=300 -DCL_USE_DEPRECATED_OPENCL_1_0_APIS -DCL_USE_DEPRECATED_OPENCL_1_1_APIS -
↪DCL_USE_DEPRECATED_OPENCL_1_2_APIS -DCL_USE_DEPRECATED_OPENCL_2_0_APIS -DCL_USE_
↪DEPRECATED_OPENCL_2_1_APIS -DCL_USE_DEPRECATED_OPENCL_2_2_APIS -I$HOME/git/pocl/build -
↪I$HOME/git/pocl/include -I$HOME/git/pocl/include/hpp -I$HOME/git/pocl/pocl -I$HOME/
↪git/pocl -g -fPIE -Werror=implicit-function-declaration -Wincompatible-pointer-types -
↪Wno-ignored-attributes -fvisibility=hidden -pthread -MD -MT saxpy.c.o -MF $FILE.c.o.d -
↪o $FILE.c.o -c $FILE.c

/usr/lib/llvm-13/bin/clang -g -pie $FILE.c.o -o a.out -Wl,-rpath,/usr/local/lib ~/git/
↪pocl/build/pocl/libpocl.a ~/git/pocl/build/lib/CL/libOpenCL.so.2.9.0 -lhwloc /usr/
↪lib/llvm-13/lib/libclang-cpp.so /usr/lib/llvm-13/lib/libLLVM-13.so -lrt -lm -ldl -lm -
↪pthread -ldl
```

```
~/git/lbt/exlbt/pocl/ex$ POCL_DEBUG=err,warn
~/git/lbt/exlbt/pocl/ex$ bash compile.sh
~/git/lbt/exlbt/pocl/ex$ ./a.out
** Final POCL_DEBUG flags: 180000000000
Hello World!
~/git/lbt/exlbt/pocl/ex$ POCL_DEBUG=
~/git/lbt/exlbt/pocl/ex$ ./a.out
Hello World!
```

References¹⁴.

9.8 RISCv OpenCL

In llvm, Mips and Cpu0 can compile .ll file from clang's output for input OpenCL while RISCv fail on both 14.x and 15.x as follows,

```
CodeGenOpenCL % pwd
$HOME/git/lbt/exlbt/pocl
% ~/llvm/14.x/llvm-project/build/bin/clang -cc1 -cl-std=CL2.0 -emit-llvm -triple spir-
→unknown-unknown test.cl
% ~/llvm/debug/build/bin/llc -march=mips test.ll
% ~/llvm/test/build/bin/llc -march=cpu0 test.ll
% ~/llvm/14.x/llvm-project/build/bin/llc -mtriple=riscv64 test.ll
LLVM ERROR: Unsupported calling convention
...
% ~/llvm/15.x/llvm-project/build/bin/clang -cc1 -cl-std=CL2.0 -emit-llvm -triple spir-
→unknown-unknown test.cl
% ~/llvm/15.x/llvm-project/build/bin/llc -mtriple=riscv64 test.ll
LLVM ERROR: Unsupported calling convention
...

CodeGenOpenCL % pwd
$HOME/llvm/14.x/llvm-project/clang/test/CodeGenOpenCL
CodeGenOpenCL % ~/llvm/debug/build/bin/clang -cc1 -cl-std=CL2.0 -emit-llvm -triple spir-
→unknown-unknown overload.cl
CodeGenOpenCL % ~/llvm/debug/build/bin/llc -march=mips overload.ll
CodeGenOpenCL % ~/llvm/test/build/bin/llc -march=cpu0 overload.ll
CodeGenOpenCL % ~/llvm/14.x/llvm-project/build/bin/llc -mtriple=riscv64 overload.ll
LLVM ERROR: Unsupported calling convention
...
```

¹⁴ <http://portablecl.org/docs/html/debug.html>

9.9 SYCL

Webs¹⁵. An example¹⁶.

9.10 DPC++

Book¹⁷.

`exlbt/pocl/dpc++.txt`

¹⁵ <https://www.khronos.org/sycl>

¹⁶ <https://en.wikipedia.org/wiki/SYCL>

¹⁷ <https://itbook.store/books/9781484255735>

RESOURCES

10.1 Build steps

<https://github.com/Jonathan2251/lbt/blob/master/README.md>

10.2 Book example code

The example code `exlbt.tar.gz` is available in:

<http://jonathan2251.github.io/lbt/exlbt.tar.gz>

10.3 Alternate formats

The book is also available in the following formats:

10.4 Presentation files

10.5 Search this website

- [search](#)

BIBLIOGRAPHY

- [clang-doc] <https://clang.llvm.org/docs/index.html>
- [clang-plugins] <https://clang.llvm.org/docs/ClangPlugins.html>
- [clang-ast-fa] <https://clang.llvm.org/docs/RAVFrontendAction.html>
- [clang-lamt] <https://clang.llvm.org/docs/LibASTMatchersTutorial.html>
- [clang-lam] <https://clang.llvm.org/docs/LibASTMatchers.html>