Tutorial: Creating an LLVM Toolchain for the Cpu0 Architecture

Release 12.0.16

Chen Chung-Shu

May 24, 2025

CONTENTS

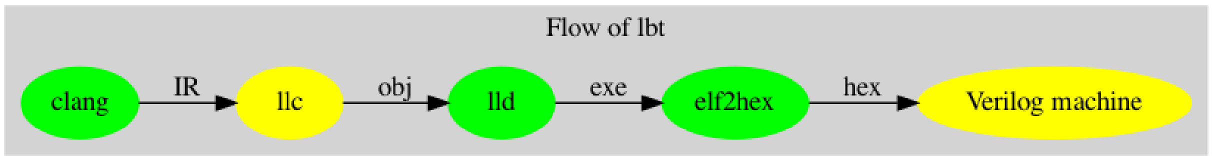


Fig. 1: This book's flow

CHAPTER

ONE

ABOUT

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 12.0.17, not released yet.

Version 12.0.16, Released May 24, 2026.

> Chatgpt: Refine my English for the input rst format texts I input the following and output rst format with <= 80 characters each line.

Version 12.0.15, skip.

> Skip for syncing same version with lbd.

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 Feburary 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 Auguest 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

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.

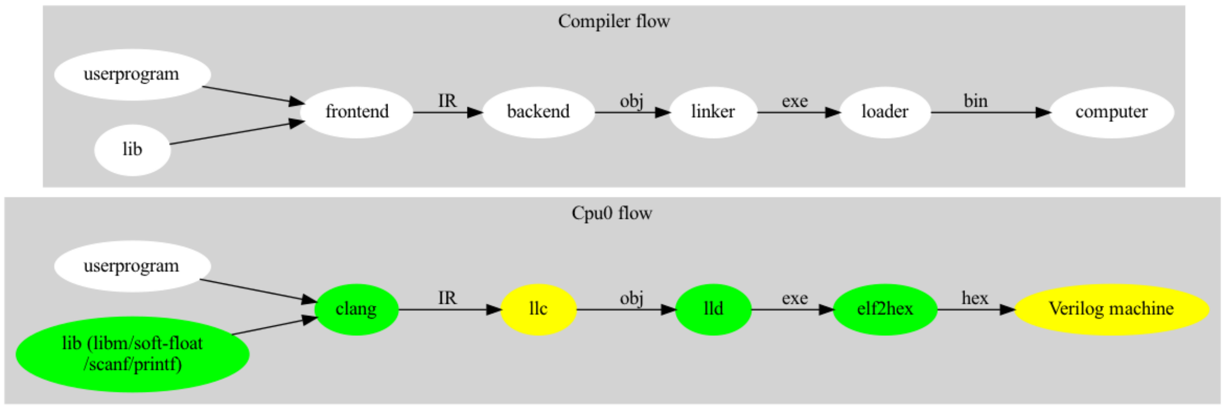


Fig. 1.1: Code generation and execution flow

Table 1.1: Toolchain components[Page 6, 23]

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.

[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

Optimization:

Backend independent optimaization.

Library:

Software floating point library and standard C library supporting.

CHAPTER

TWO

CLANG

This chapter adds the Cpu0 target to the 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("_ABIO32", "1");
    Builder.defineMacro("_CPU0_SIM", "_ABIO32");
  } 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

A builtin function is a function that may map to one or more hardware instructions for performance improvement. The Cpu0 backend ports compiler-rt's builtins in the section "Compiler-rt's Builtins" of the Library chapter[1].

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 device[2].

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.
//
//===----------------------------------------------------------------------===//

```

(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 OpenCL2.0, 3rd edition"

(continued from previous page)

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

// 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
```

CHAPTER

THREE

CPU0 ELF LINKER

LLD changes frequently, and the figures in this chapter are not up to date. Like LLVM, the LLD linker includes support for multiple targets in ELF format.

The term "Cpu0 backend" used in this chapter can refer to ELF format handling for the Cpu0 target under LLD, the LLVM compiler backend, or both. It is assumed that readers will understand the intended meaning from the context.

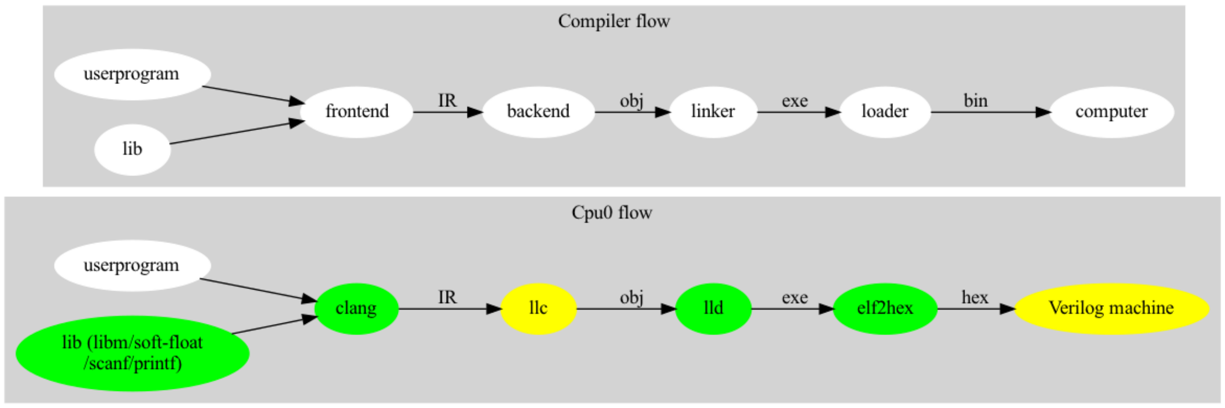


Fig. 3.1: Code generation and execution flow

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 the Cpu0 backend as well as elf2hex to replace the Cpu0 loader.

After linking with LLD, the program with global variables can be allocated in ELF file format layout. This means relocation records of global variables are resolved.

In addition, elf2hex is implemented to generate Hex files from ELF.

With these two tools, global variables in sections .data and .rodata can be accessed and transferred to Hex files which feed the Verilog Cpu0 machine running on your PC/Laptop.

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

For more about LLD, please refer to the LLD website here[1] and the LLD install requirements on Linux here[2].

Currently, LLD can be built by gcc and clang on Ubuntu. On iMac, LLD can be built by clang with the Xcode version as described in the next subsection.

If you run inside a Virtual Machine (VM), please set the physical memory size over 1GB to avoid insufficient memory link errors.

3.1 ELF to Hex

The code to convert ELF file format into Hex format for Cpu0 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})
```

(continues on next page)

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

(continued from previous page)

```
endif()

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

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() {};
};
```

(continues on next page)

(continued from previous page)

```
// Split HexOut from Reader::DisassembleObject() for separating hex output
// functions.
class VerilogHex : public HexOut {
public:
  VerilogHex(std::unique_ptr<MCInstPrinter>& instructionPointer,
             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;

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
// 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;

static StringRef ToolName;
static StringRef 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)
```

(continues on next page)

(continued from previous page)

```
    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 = "";

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>
```

(continues on next page)

(continued from previous page)

```
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()) {
    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());
    }
```

(continues on next page)

(continued from previous page)

```
    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());
    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));
```

(continues on next page)

(continued from previous page)

```
    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));
    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;
```

(continues on next page)

(continued from previous page)

```
  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,
                                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
```

(continues on next page)

(continued from previous page)

```
  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";

  // 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;
```

(continues on next page)

(continued from previous page)

```
  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.
  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;
```

(continues on next page)

(continued from previous page)

```
}

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

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

// Porting from DisassembleObject() of llvm-objump.cpp
void Reader::DisassembleObject(const ObjectFile *Obj
/*, 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;
    unwrapOrError(Section.getContents(), Obj->getFileName());
    BaseAddr = Section.getAddress();
    uint64_t SectSize = Section.getSize();
    if (!SectSize)
      continue;

    if (BaseAddr < 0x100)
      continue;

  #ifdef ELF2HEX_DEBUG
    StringRef SectionName = unwrapOrError(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;
```

(continues on next page)

(continued from previous page)

```
        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
    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.
```

(continues on next page)

(continued from previous page)

```
      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),
                                   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-objump.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);
```

(continues on next page)

(continued from previous page)

```
  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()));

  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]);
```

(continues on next page)

(continued from previous page)

```
  //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.
  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;
}
```

To support the commands **llvm-objdump -d** and **llvm-objdump -t** for Cpu0, the following code is added to llvm-objdump.cpp:

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, the linker performs relocation record resolution as described in the ELF support chapter. Some optimizations cannot be completed during the compiler stage. One such optimization opportunity in the 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();
```

(continues on next page)

(continued from previous page)

```
}
```

The above code can be simplified to Fig. 3.2 to perform mark and sweep in the graph for Dead Code Stripping.

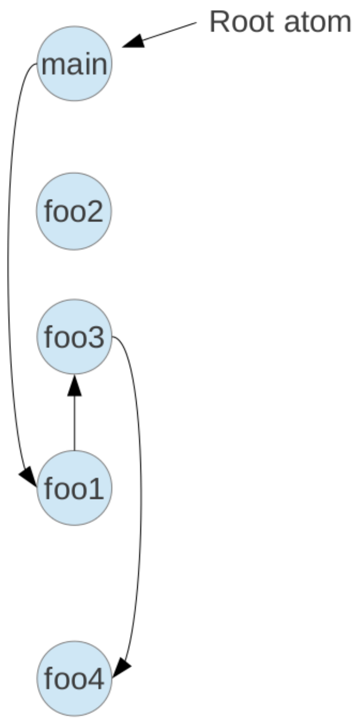


Fig. 3.2: Atom classified (from lld web)

As shown in the above example, the function *foo2()* is an isolated node without any references. It is dead code and should be removed during linker optimization. We tested this example using *Makefile.ch13_1* and found that *foo2()* was not removed.

There are two possible reasons for this behavior. First, the dead code stripping optimization in LLD might not be enabled by default in the command line. Second, LLD may not have implemented this optimization yet, which is reasonable given that LLD was in its early stages of development at the time.

We did not investigate further since the Cpu0 backend tutorial only requires a linker to complete relocation record resolution and to run the program on a PC.

Note that *llvm-linker* works at the LLVM IR level and performs linker optimizations there. However, if you only have object files (e.g., *a.o*), the native linker (such as LLD) has the opportunity to perform dead code stripping, while the IR linker cannot.

3.2.2 Static linker

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

File `printf-stdarg.c`[7] comes from an internet download and is under the GPL2 license. GPL2 is more restrictive than the 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.

    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;
```

(continues on next page)

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

(continued from previous page)

```
  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 fffffffd
 * 0 message(s)
 * 0 message(s) with %
 * justif: "left      "
 * justif: "     right"
 *  3: 0003 zero padded
 *  3: 3    left justif.
```

(continues on next page)

(continued from previous page)

```
 *  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.
```

(continues on next page)

(continued from previous page)

```
#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;
    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
```

(continues on next page)

(continued from previous page)

```
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';
    u /= b;
  }

  if (neg) {
    if( width && (pad & PAD_ZERO) ) {
      printchar (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 == '%') {
```

(continues on next page)

(continued from previous page)

```
      ++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;
      }
      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;
    }
```

(continues on next page)

(continued from previous page)

```
  }
  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;
  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 fffffffd
 * 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/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
```

(continues on next page)

(continued from previous page)

```
  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)
```

(continues on next page)

(continued from previous page)

```
# 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="$(CC)" CFLAGS="$(CFAGS)" 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
ifdef 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)
ifdef LIBBUILTINS_DIR
        cd $(LIBBUILTINS_DIR) && $(MAKE) -f Makefile clean
endif
ifdef NEWLIB_DIR
        cd $(NEWLIB_DIR) && rm -rf build-$(CPU)-$(ENDIAN)/*
endif
```

(continues on next page)

(continued from previous page)

```
# 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()` function from the GPL source code, we can write more test code to verify the previously generated LLVM Cpu0 backend program. The following code serves 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");
  asm("ori $7, $7, 0xfdff");
  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) {
```

(continues on next page)

(continued from previous page)

```
      printf("Harware interrupt 0\n");
      asm("lui $7, 0xffff");
      asm("ori $7, $7, 0x7fff");
      asm("and $sw, $sw, $7");
    }
    else if (int2) {
      printf("Harware 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, 0xdfff");
    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");
  asm("ld $14, 48($at)");
  asm("c0mov $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_lld_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 exceptionOccur;
extern int test_detect_exception(bool exception);

extern int test_staticlink();
```

exlbt/input/ch_lld_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);
```

(continues on next page)

(continued from previous page)

```
  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;
  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__
```

(continues on next page)

(continued from previous page)

```
  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);
  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();
```

(continues on next page)

(continued from previous page)

```
  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
    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
```

(continues on next page)

(continued from previous page)

```
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 "useage: bash $sh_name cpu_type ENDIAN"
    echo "  cpu_type: cpu032I or cpu032II"
    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;
```

(continues on next page)

(continued from previous page)

```
  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
}

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
```

(continues on next page)

(continued from previous page)

```
$ 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
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
```

(continues on next page)

(continued from previous page)

```
hex 00 = 00
signed -3 = unsigned 4294967293 = hex fffffffd
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
exceptionOccur= 0, PASS
inlineasm_global() = 4, PASS
20
10
5
test_cpp_polymorphism() = 0, PASS
taskInterrupt(011)
Software interrupt
taskInterrupt(011)
Harware interrupt 0
taskInterrupt(011)
Harware interrupt 1
...
RET to PC < 0, finished!
```

The above test includes verification of the `printf` format. Let's check the result by comparing it with the output of the PC program `printf-stdarg-1.c` 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!
```

(continues on next page)

(continued from previous page)

```
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 fffffffd
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 described above, by leveraging open source code, Cpu0 gained a more stable implementation of the `printf()` function. Once the Cpu0 backend can translate the open source C `printf()` program into machine instructions, the LLVM Cpu0 backend can be effectively verified using `printf()`.

With the high-quality open source `printf()` code, the Cpu0 toolchain extends from just a compiler backend to also support the C standard library. (Note that while some GPL open source code may be of lower quality, many are well-written.)

The message "Overflow exception is printed twice" means the ISR() in `debug.cpp` is called twice from `ch4_1_2.cpp`.

The printed messages `taskInterrupt(001)` and `taskInterrupt(011)` are just trace messages from `cpu0.v` code.

3.2.3 Dynamic linker

The dynamic linker demonstration was removed from version 3.9.0 because its implementation with lld 3.9 was unclear and required extensive additions to `elf2hex`, Verilog, and the Cpu0 backend in lld.

However, it can still be run with LLVM 3.7 using 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 read the corresponding section in `lld.html` for more details.

3.3 Summary

3.3.1 Create a new backend based on LLVM

Thanks to the LLVM open source project, writing a linker and ELF-to-Hex tools for a new CPU architecture is both easy and reliable.

Combined with the LLVM Cpu0 backend code and Verilog code programmed in previous chapters, we designed a software toolchain to compile C/C++ code, link it, and run it on the Verilog Cpu0 simulator without any real hardware investment.

If you buy FPGA development hardware, we believe this code can run on an FPGA CPU even though we did not test it ourselves.

Extending system programming toolchains to support a new CPU instruction set can be completed just like we have shown here.

School knowledge of system programming, compiler, linker, loader, computer architecture, and CPU design has been translated into real work and demonstrated running in practice. Now this knowledge is not limited to paper.

We designed it, programmed it, and ran it in the real world.

The total code size of the LLVM Cpu0 backend compiler, Cpu0 LLD linker, elf2hex, and Cpu0 Verilog code is around 10,000 lines including comments.

In comparison, the total code size of Clang, LLVM, and LLD exceeds 1,000,000 lines excluding tests and documentation.

This means the Cpu0 backend, based on Clang, LLVM, and LLD, is only about 1% of the size of the entire infrastructure.

Moreover, the LLVM Cpu0 backend and LLD Cpu0 backend share about 70% similarity with LLVM MIPS.

Based on this fact, we believe LLVM has a well-defined architecture for compiler design.

3.3.2 Contribute Back to Open Source Through Working and Learning

Finally, 10,000 lines of source code in the Cpu0 backend is very small for a UI program, but it is quite complex for system programming based on LLVM. Open source code gives programmers the best opportunity to understand, enhance, and extend the code. However, documentation is the next most important factor to improve open source development.

The Open Source Organization recognized this and started the Open Source Document Project[8][9][10][11][12]. Open Source has grown into a giant software infrastructure powered by companies[13][14], academic research, and countless passionate engineers.

It has ended the era when everyone tried to reinvent the wheel. Extending your software from reusable source code is the right way. Of course, if you work in business, you should consider open source licensing.

Anyone can contribute back to open source through the learning process. This book was written by learning the LLVM backend and contributing back to the LLVM open source project.

[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/

We believe this book cannot exist in traditional paper form, since few readers are interested in studying LLVM backend, despite many compiler books published. Therefore, it is published electronically and tries to follow the Open Document License Exception[15].

There is a gap between concept and realistic program implementation. For learning a large, complex software such as the LLVM backend the concept of compiler knowledge alone is not enough.

We all learned knowledge through books in and after school. If you cannot find a good way to produce documentation, consider writing documents like this book. This book's documentation uses the Sphinx tool, just like the LLVM development team. Sphinx uses the restructured text format here[16][17][18].

Appendix A of the lbd book explains how to install the Sphinx tool.

Documentation work helps you re-examine your software and improve your program structure, reliability, and—most importantly—extend your code to places you did not expect.

[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

CHAPTER

FOUR

OPTIMIZATION

This chapter introduces LLVM optimization.

4.1 LLVM IR Optimization

The *llvm-link* tool provides optimization at the IR level, which can be applied across different programs developed in multiple languages. Of course, it can also be applied to the same language that supports separate compilation.

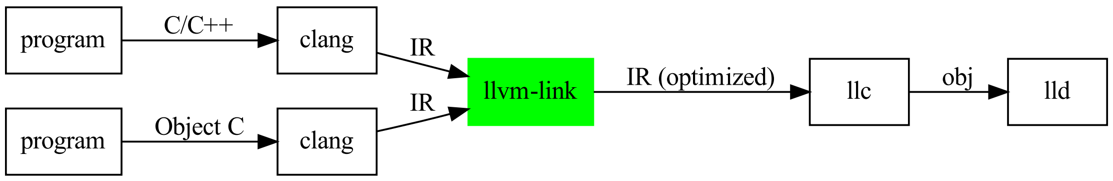


Fig. 4.1: llvm-link flow

Clang provides optimization options to optimize code from high-level languages to IR. However, since many languages like C/C++ support separate compilation, there is no opportunity for inter-procedure optimization if functions come from different source files.

To solve this problem, LLVM provides **llvm-link** to link all **.bc* files into a single IR file, and then uses **opt** to perform inter-procedure optimization[1].

Beyond the DAG local optimization discussed in Chapter 2 of lbd, LLVM supports global optimizations based on inter-procedure analysis[2].

The following steps and examples demonstrate this optimization approach 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 above, the **opt** output contains a smaller number of IR instructions. As a result, the backend-generated code will also be more efficient, as shown below.

```
JonathantekiiMac:input Jonathan$ ~/llvm/test/build/
bin/llc -march=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
        .cprestore    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 seperate compile, it come with the cost in compile time. As you can imagine, any one statement change may change the output IR of llvm-link. And the obj binary code have to re-compile. Compare to the seperate 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 page[3] will get the information.

[3] http://cs.boisestate.edu/~uh/

CHAPTER

FIVE

LIBRARY

5.1 The theory of Floating Point Implementation

Fixed-point representation is used to implement floating-point numbers, as illustrated in Fig. 5.1. The calculation is described below.

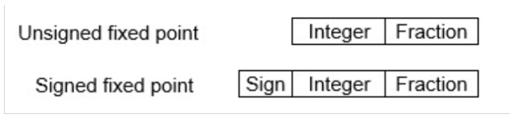


Fig. 5.1: Fixed point representation

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 xor 0) (11 00 * 00 10) >> 2} = {0 01 10} = 1.5*

The layout for half-precision floating-point format is shown in Fig. 5.2.

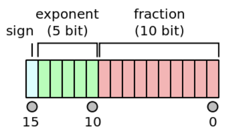


Fig. 5.2: IEEE 754 half precision of Floating point representationPage 71, 2

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

Floating-point arithmetic can be implemented in both software and hardware.

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

The 16-bit product *a * b* can be computed by first converting both *a* and *b* to fixed-point format using more bits of memory or registers. After performing the multiplication as fixed-point arithmetic, the result is converted back, as described on this website[1].

An example of multiplication using exponent base 2 is given below:

- Precondition: *a* and *b* are normalized IEEE half-precision (16-bit) floating point values[2]. The exponent bias is 15. For example: 15 -> 0, 1 -> -14, 30 -> 15, and 31 -> NaN.
- Transformation for *a * b*:
  1. *{sign-bit(a) xor sign-bit(b)} {exponent(a) + exponent(b) - 15} {significand(a) * significand(b) >> 10}*
  2. Normalize the result.
- Example:

  a = 0.01 (binary) = *{0 01110 1000000000}* b = 1.1 (binary) = *{0 10000 1100000000}*

  1. *a * b = {0 xor 0} {01110 + 10000 - 01111 = 01111} {1000000000 * 1100000000 >> 10 = 0110000000}*
  2. Normalize: *{0 01111 0110000000}* $\rightarrow$ *{0 01110 1100000000}* = 0.011

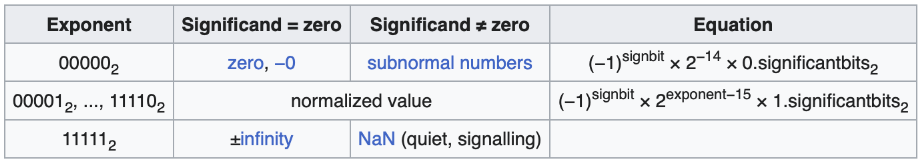
Division is handled similarly:

- Transformation for *a / b*:
  1. *{sign-bit(a) xor sign-bit(b)} {exponent(a) - exponent(b) + 15} {significand(a) / significand(b) >> 10}*
  2. Normalize the result.
- Example:

  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}* $\rightarrow$ *{0 01101 1000000000}* = 0.01

The IEEE-754 floating-point standard also includes special cases such as NaN (Not a Number), which can result from operations like 0/0, and *infty*, as illustrated in Fig. 5.3.



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

Fig. 5.3: Encoding of exponent for IEEE 754 half precision[Page 71, 2]

Since normalization in floating-point arithmetic is a critical operation, the Cpu0 hardware provides *clz* (count leading zeros) and *clo* (count leading ones) instructions to speed up the normalization process.

The *compiler-rt* library implements floating-point multiplication by handling special cases like NaN and ∞ in the same way as described in the implementation above. It also uses *clz* and *clo* instructions to accelerate normalization, as shown below:

[1] https://witscad.com/course/computer-architecture/chapter/floating-point-arithmetic

~/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 {
    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 builtins

Since Cpu0 does not have hardware floating-point instructions, it requires a software floating-point library to perform floating-point operations.

The LLVM *compiler-rt* project provides a software floating-point implementation (Fig. 5.4), so I chose it for this purpose.

As *compiler-rt* assumes a Unix/Linux rootfs structure, we bridge the gap by adding a few empty header files in *exlbt/include*.

The dependencies for *compiler-rt* on *libm* are shown in Fig. 5.6.

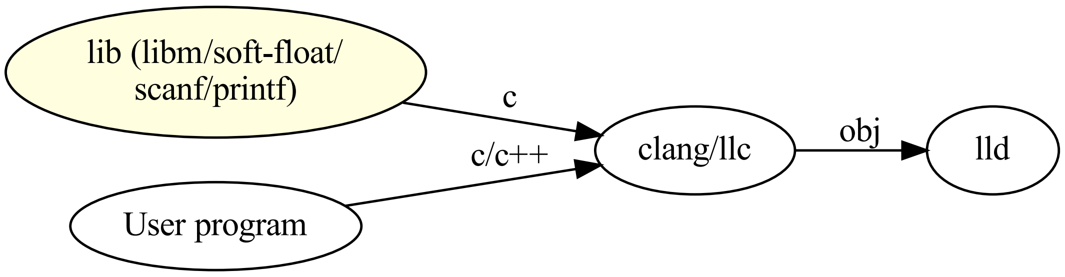


Fig. 5.4: compiler-rt/lib/builtins' software float library

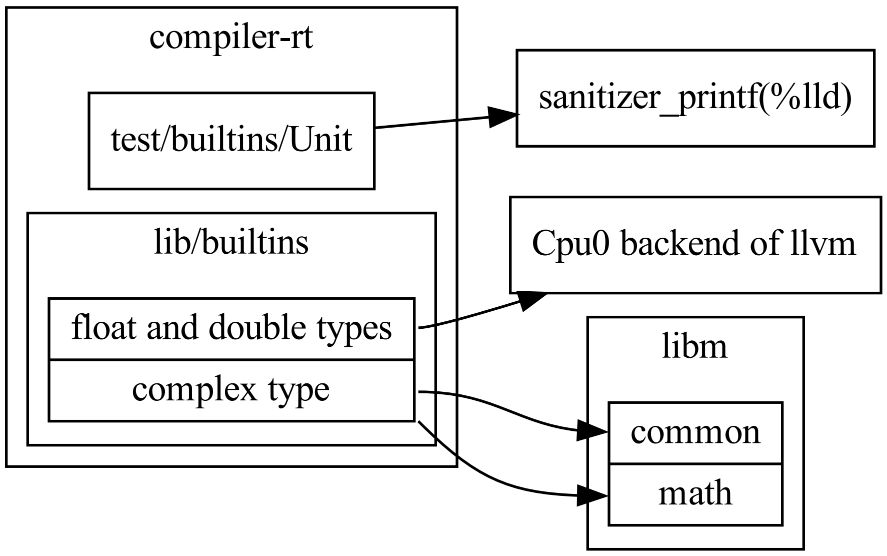


Fig. 5.5: Dependences for compiler-rt on libm

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
sanitizter_printf.c	builtins of compiler-rt

5.3 C Library (Newlib)

Since the complex type in *compiler-rt* depends on *libm*, I port Newlib in this section.

Newlib is a C library designed for bare-metal platforms. It consists of two libraries: *libc* and *libm*. The *libc* library supports I/O, file, and string functions, while *libm* provides mathematical functions.

The official website for Newlib is available here[3], and the *libm* library can be found here[4].

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/.
```

(continues on next page)

[3] https://sourceware.org/newlib/
[4] https://sourceware.org/newlib/libm.html

(continued from previous page)

```
  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 ..
}

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 "${host_cpu}" in
   fr30)
         machine_dir=fr30
         ;;
+  cpu0)
+        machine_dir=cpu0
+        newlib_cflags="${newlib_cflags} -DCOMPACT_CTYPE"
+        ;;
   frv)
         machine_dir=frv
         ;;
@@ -751,6 +755,9 @@ newlib_cflags="${newlib_cflags} -DCLOCK_PROVIDED -DMALLOC_PROVIDED -
→DEXIT_PROVID
   fr30-*-*)
         syscall_dir=syscalls
         ;;
+  cpu0-*)
+        syscall_dir=syscalls
+        ;;
   frv-*-*)
         syscall_dir=syscalls
         default_newlib_io_long_long="yes"
diff --git a/newlib/libc/include/machine/ieeefp.h b/newlib/libc/include/machine/ieeefp.h
index 3c1f41e03..1e79a6b26 100644
--- a/newlib/libc/include/machine/ieeefp.h
+++ b/newlib/libc/include/machine/ieeefp.h
@@ -249,6 +249,16 @@
 #define __IEEE_BIG_ENDIAN
 #endif

+// 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 __CPU0EL__
+#define __IEEE_LITTLE_ENDIAN
```

(continues on next page)

(continued from previous page)

```
+#endif
+#ifdef __CPU0EB__
+#define __IEEE_BIG_ENDIAN
+#endif
+
 #ifdef __MMIX__
 #define __IEEE_BIG_ENDIAN
 #endif
@@ -507,4 +517,3 @@

 #endif /* not __IEEE_LITTLE_ENDIAN */
 #endif /* not __IEEE_BIG_ENDIAN */
-
diff --git a/newlib/libc/include/sys/unistd.h b/newlib/libc/include/sys/unistd.h
index 3cc313080..605929173 100644
--- a/newlib/libc/include/sys/unistd.h
+++ b/newlib/libc/include/sys/unistd.h
@@ -50,7 +50,7 @@ int     dup3 (int __fildes, int __fildes2, int flags);
 int        eaccess (const char *__path, int __mode);
 #endif
 #if __XSI_VISIBLE
-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


        .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:
```

(continues on next page)

(continued from previous page)

```
        addu    $2,$0,$5
        ret        $lr
```

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

The *libm.a* library depends on the *errno* variable from *libc*, which is defined in *sys/errno.h*.

- libgloss is BSP license[5]

5.4 Compiler-rt's builtins

Compiler-rt is a project for runtime libraries implementation[6]. The *compiler-rt/lib/builtins* directory provides functions for basic operations such as +, -, *, /, etc., on *float* or *double* types. It also supports type conversions between floating-point and integer, or conversions involving types wider than 32 bits, such as *long long*.

The *compiler-rt/lib/builtins/README.txt*[7] lists the dependent functions used throughout the builtins. These dependent functions are a small subset of *libm*, which are defined in *compiler-rt/lib/builtins/int_math.h*[8].

~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 shown in the Makefile above, Newlib uses the *libm/math* directory.

The dependencies for the builtin functions of compiler-rt on *libm* are shown in Fig. 5.6.

In this section, I copied test cases for verification of software floating point (SW FP) 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 the test cases in *compiler-rt/test/builtins/Unit* require this format to verify SW FP results, I ported *sanitizer_printf.cpp* and *sanitizer_internal_defs.h* to *lbt/exlbt/input/* from *compiler-rt/lib/sanitizer_common/*.

[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

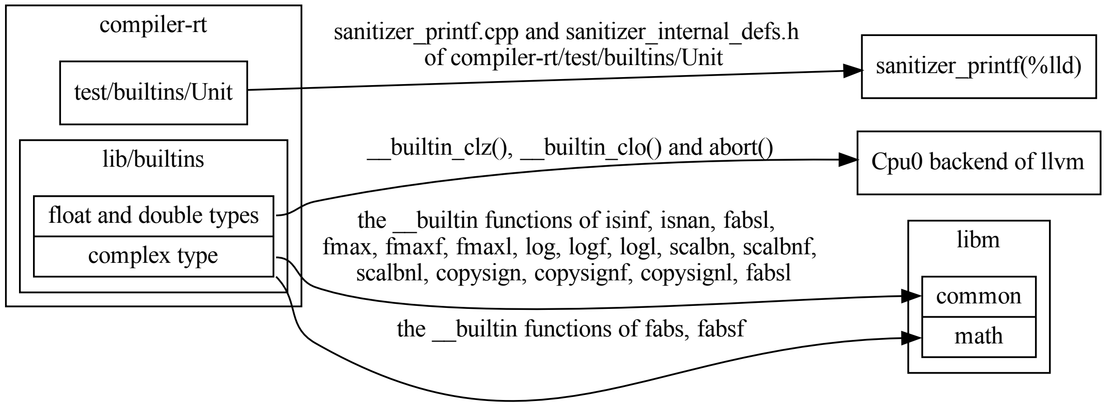

compiler-rt
test/builtins/Unit
lib/builtins
float and double types
complex type
sanitizer_printf.cpp and sanitizer_internal_defs.h of compiler-rt/test/builtins/Unit
sanitizer_printf(%lld)
__builtin_clz(), __builtin_clo() and abort()
Cpu0 backend of llvm
the __builtin functions of isinf, isnan, fabsl, fmax, fmaxf, fmaxl, log, logf, logl, scalbn, scalbnf, scalbnl, copysign, copysignf, copysignl, fabsl
the __builtin functions of fabs, fabsf
libm
common
math

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

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	scalblnl.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 dependencies on any other library.
- Only the *complex* type in *compiler-rt/lib/builtins* depends on libm. Other types (float and double) only depend on *__builtin_clz()*, *__builtin_clo()*, and *abort()*. I have ported *abort()* in *lbt/exlbt/compiler-rt/cpu0/abort.c*.
- All test cases in *compiler-rt/test/builtins/Unit* depend on *printf(%lld or %llX, …)*. I ported this functionality from *compiler-rt/lib/sanitizer_common/sanitizer_printf.cpp* to *lbt/exlbt/input/sanitizer_printf.cpp*.
- The dependent functions for *complex* type have been ported from *newlib/libm*.
- Except for *builtins*, the other three components—sanitizer runtimes, profile and BlocksRuntime in *compiler-rt*

are not needed for my embedded Cpu0.

The libgcc integer and soft float libraries[9][10][11] are functionally equivalent to the builtins in *compiler-rt*.

In *compiler-rt/lib/builtins*, the file-level dependencies are listed in the following table.

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

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

Though the 'rt' stands for Runtime Libraries, most functions in the *builtins* library are written in target-independent C code. These functions can be compiled and statically linked into the target.

When you compile the following C code, *llc* will generate a **call to __addsf3** to invoke the compiler-rt floating-point function for Cpu0.

This is because Cpu0 does not have hardware floating-point instructions, so the Cpu0 backend does not handle the DAG for *__addsf3*. As a result, LLVM treats the DAG for *__addsf3* as a function call, rather than a direct float-add instruction.

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 -
      ...
```

(continues on next page)

[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

(continued from previous page)

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

For some bare-metal or embedded applications, the C code does not need the file and high-level I/O features provided by *libc*.

libm provides a wide range of functions to support software floating-point operations beyond basic arithmetic[12].

libc provides file handling, high-level I/O functions, and some basic float operations[13].

Cpu0 uses *compiler-rt/lib/builtins* and *compiler-rt/lib/sanitizer_common/sanitizer_printf.cpp* to support software floating-point.

The *compiler-rt/lib/builtins* is a target-independent C implementation of a software floating-point library. Cpu0 currently implements only *compiler-rt-12.x/cpu0/abort.c* to support this functionality.

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

Though these *compiler-rt* built-in functions are written in C, CPUs can provide hardware float or high-level instructions to accelerate them. Compilers like Clang can convert float-type operations in C into LLVM IR. Then, the backend compiles them into specific hardware instructions for performance.

To optimize *libm* functions, many CPUs include hardware floating-point instructions.

For example, the Clang compilation and backend translation go as follows:

- *float a, b, c; a = b * c;* → (Clang) → *%add = fmul float %0, %1*[15]

MIPS backend compiles *fmul* into hardware instructions:

- *%add = fmul float %0, %1* → (LLVM-MIPS) → *mul.s*[Page 83, 15][14]

Cpu0 backend, without hardware float support, compiles *fmul* into a library function call:

- *%add = fmul float %0, %1* → (LLVM-Cpu0) → *jsub fmul*[15]

For high-level math functions, Clang compiles float-type operations in C into LLVM intrinsic functions. Then, LLVM backends for different CPUs compile these intrinsics into hardware instructions when available.

For example, Clang compiles *pow()* into *@llvm.pow.f32* as follows:

- *%pow = call float @llvm.pow.f32(float %x, float %y)*[16]

The AMDGPU backend compiles *@llvm.pow.f32* into a sequence of instructions:

- *%pow = call float @llvm.pow.f32(float %x, float %y)* → (LLVM-AMDGPU) → … + *v_exp_f32_e32 v0, v0* + …[16]

The MIPS backend compiles *@llvm.pow.f32* into a function call:

- *%pow = call float @llvm.pow.f32(float %x, float %y)* → (LLVM-MIPS) → *jal powf*[16]

Clang treats these *libm* functions as built-ins and compiles them into LLVM IR or intrinsics. Then, the LLVM backend can either lower them into hardware instructions (if available) or generate function calls to built-in implementations in *libm*.

[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_fmul.c
[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

The following quote is from Clang's documentation[17]:

```
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, supports *printf("%lld")*. Its implementation calls some floating-point library 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)
  #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
```

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

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.
//===----------------------------------------------------------------------===//
#ifndef SANITIZER_DEFS_H
#define SANITIZER_DEFS_H
```

(continues on next page)

(continued from previous page)

```
// 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 {

#if defined(_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)
#if defined(__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;
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);

#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
```

(continues on next page)

(continued from previous page)

```
// 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;
    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) {
```

(continues on next page)

(continued from previous page)

```
  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);
    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 */,
```

(continues on next page)

(continued from previous page)

```
                         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) {
      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);
```

(continues on next page)

(continued from previous page)

```
        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;
      }
      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)
{
```

(continues on next page)

(continued from previous page)

```
  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);
  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;
}
```

The above two *sanitizer_*.** files are ported from compiler-rt. I added code to support left-justify for number printf and right-justify for string printf. The following *ch_float.cpp* tests the float library.

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

```
# Thanks .c .cb Vranish (https://spin.a.cmi.cbject..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 \
  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 \
```

(continues on next page)

(continued from previous page)

```
  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 \
  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 \
```

(continues on next page)

(continued from previous page)

```
  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 \
  truncdfsf2.c \
  truncsfhf2.c \
  ucmpdi2.c \
  ucmpti2.c \
  udivdi3.c \
  udivmoddi4.c \
  udivmodsi4.c \
  udivmodti4.c \
  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
OBJS := $(SRCS:%=$(BUILD_DIR)/%.o)

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

(continues on next page)

(continued from previous page)

```
# 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 -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.
CPPFLAGS := -MMD -MP -target cpu0$(ENDIAN)-unknown-linux-gnu -static \
  -fintegrated-as $(INC_FLAGS) -mcpu=$(CPU) -mllvm -has-lld=true

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

# 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 = 0x100000007fffffff;
  printf("a: %llX, %llx, %lld\n", a, a, a);
  int b = 0x10000000;
  printf("b: %x, %d\n", b, b);
  sprintf(buf, "b: %x, %d\n", b, b); printf("%s", buf);
```

(continues on next page)

(continued from previous page)

```
  // 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);
}

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;
}
```

(continues on next page)

(continued from previous page)

```
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;

  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>(0x001666660000000a, 48LL); // 0x16 = 22
  check_result("test_shift_right<long long>(0x001666660000000a, 48LL)", a, 22LL);
```

(continues on next page)

(continued from previous page)

```
// call __lshrdi3
  b = test_shift_right<unsigned long long>(0x001666660000000a, 48LLu); // 0x16 = 22
  check_result("test_shift_right<unsigned long long>(0x001666660000000a, 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
  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);
```

(continues on next page)

(continued from previous page)

```
#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: 100000007FFFFFFF, 100000007fffffff, 1152921506754330623
b: 10000000, 268435456
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 fffffffd
0 message(s)
0 message(s) with %
justif: "left      "
justif: "     right"
 3: 0003 zero padded
 3: 3    left justif.
 3:    3 right justif.
```

(continues on next page)

(continued from previous page)

```
-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>(0x001666660000000a, 48LL) = 22
test_shift_right<unsigned long long>(0x001666660000000a, 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* directory is 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();
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();
```

(continues on next page)

(continued from previous page)

```
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();
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();
```

(continues on next page)

(continued from previous page)

```
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 lshrti3_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();
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, ...
```

(continues on next page)

(continued from previous page)

```
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 trunctfdf2_test();
extern "C" int trunctfhf2_test();
extern "C" int trunctfsf2_test();
extern "C" int ucmpdi2_test();
extern "C" int ucmpti2_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();
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();
  }
}
```

(continues on next page)

(continued from previous page)

```
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();
  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);
```

(continues on next page)

(continued from previous page)

```
#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);

//  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();
```

(continues on next page)

(continued from previous page)

```
  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);
#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
```

(continues on next page)

(continued from previous page)

```
  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);

  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();
```

(continues on next page)

(continued from previous page)

```
  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();
  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);
```

(continues on next page)

(continued from previous page)

```
#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();
  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);
```

(continues on next page)

(continued from previous page)

```
  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();
  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);
```

(continues on next page)

(continued from previous page)

```
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 = truncdfsf2_test();
show_result("truncdfsf2_test()", res);

res = truncdfsf2vfp_test();
show_result("truncdfsf2vfp_test()", res);

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

res = trunctfdf2_test();
show_result("trunctfdf2_test()", res);

res = trunctfhf2_test();
show_result("trunctfhf2_test()", res);

res = trunctfsf2_test();
show_result("trunctfsf2_test()", res);

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

res = ucmpti2_test();
show_result("ucmpti2_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);
```

(continues on next page)

(continued from previous page)

```
  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 \
```

(continues on next page)

(continued from previous page)

```
  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 \
  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/lshrdi3_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 \
```

(continues on next page)

(continued from previous page)

```
  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 \
  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 \
```

(continues on next page)

(continued from previous page)

```
  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 the tests as follows,

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

(continues on next page)

(continued from previous page)

```
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!
lesf2vfp_test(): SKIPPED!
letf2_test(): SKIPPED!
lshrdi3_test(): PASS!
lshrti3_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!
```

(continues on next page)

(continued from previous page)

```
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!
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!
```

CHAPTER

SIX

LLDB

With *scanf* from newlib/libc and *printf* from compiler-rt, we are now ready to start porting LLDB. Beginning…

CHAPTER

SEVEN

APPENDIX A: RISCV

This chapter shows the installation of the RISC toolchain, including GNU, LLVM, and simulators on Linux, as illustrated in the figure and table below.

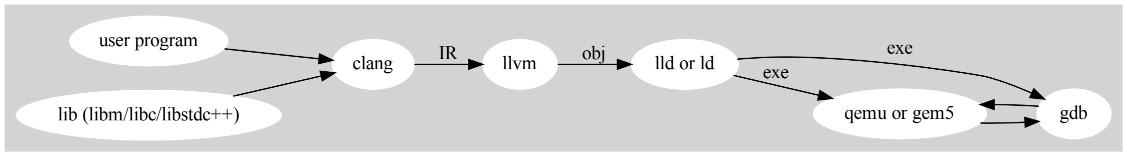


Fig. 7.1: RISCV toolchain flow

Table 7.1: RISCV toolchain[Page 122, 1]

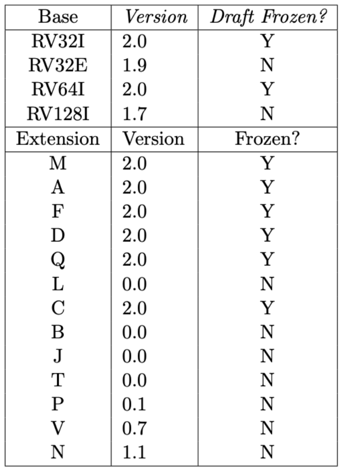
Component	name	github
C/C++ Compiler	clang/llvm	llvm-project
LLVM Assembler	llvm integrated assembler	"
LLVM Linker	ld.lld	"
debug tool	lldb	"
Utils	llvm-ar, llvm-objdump etc.	"
gcc Assembler	as	riscv-gnu-toolchain
gcc Linker	ld.bfd ld.gold	"
Runtime	libgcc	"
Unwinder	libgcc_s	"
C library	libc	"
C++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

7.1 ISA

As shown in Fig. 7.2 and Fig. 7.3, RISC has 32/64/128 bit variants. The I (integer) extension is the base part, and others are optional. G = IMAFD, the general extensions (i.e., I, M, A, F, D)[2][3][4].

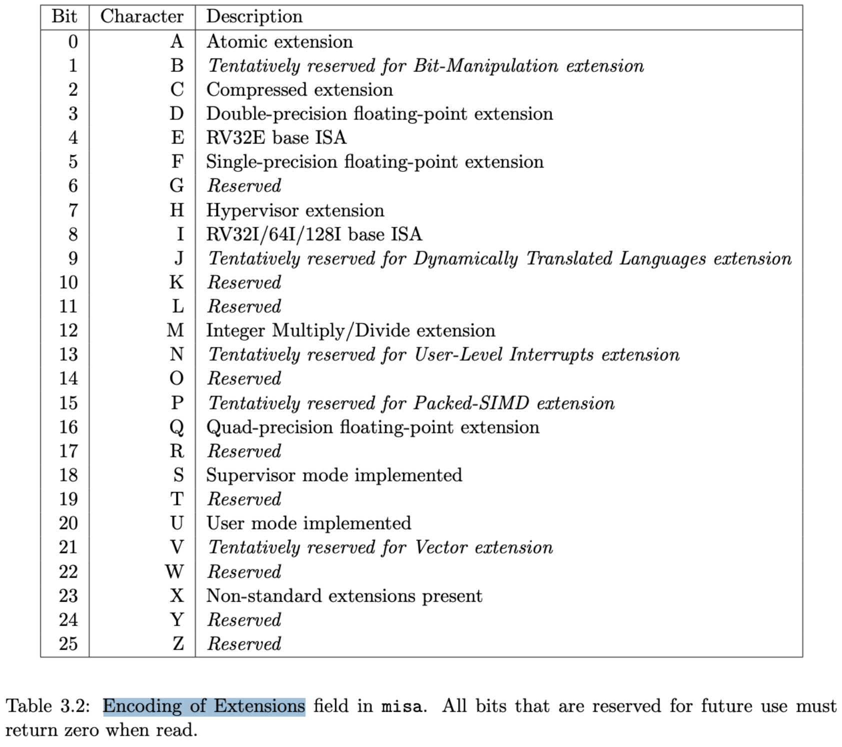
Since RISC-V has vector instructions supporting variable-length data and allows vendors to encode variable-length instruction sets, little endian is the dominant format in the market[5].

[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: RISCV ISA



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

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

Fig. 7.3: RISCV ISA Description

7.2 Mem

- I-cache, D-cache: Size ranges from 4KB to 64KB in Andes N25f.
- ILM, DLM: Size ranges from 4KB to 16MB[6].
  - For deterministic and efficient program execution
  - Flexible size selection to fit diversified needs
- DRAM

7.3 RISC compiler toolchain installation

First, install the dependent packages following https://github.com/riscv-collab/riscv-gnu-toolchain#readme. Next, create your $HOME/riscv and $HOME/riscv/git directories. 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/andescore-processors/riscv-n25f/ , http://www.andestech.com/en/products-solutions/andestar-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␣
↪incurs 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 support for both bare-metal (Newlib) and Linux platforms.

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

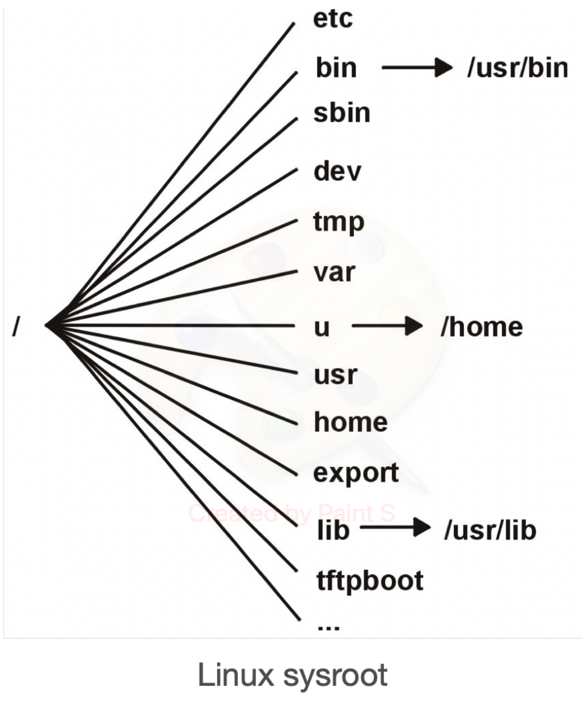


Fig. 7.4: RISCV ISA Description

The Linux sysroot is shown in 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 own memory map for the RISCV architecture. As a result, their HW may need to initialize $sp, $pc, $gp, and others. The GNU linker command language[15] allows users to specify the memory map for their HW.

The crt0.S and riscv64-virt.ld in lbt/exlbt/riscv are modified from the original 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:
<    elf64lriscv
<    elf32lriscv
<    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=rv64gcv1p0 -O0 -mabi=lp64d -T riscv64-virt.ld -nostartfiles crt0.S -v
```

If RAM is used with (rwx) permission, a warning may occur. This warning can be suppressed by adding *-Wl,–no-warn-rwx-segment* to the clang options.

QEMU in the next section can run the program without initializing $sp in crt0.S. Perhaps QEMU initializes $sp as shown in the following code.

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

```
qemu$ pwd
~/riscv/git/qemu
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 the system loader reads to know how to load the program into memory. The program headers of an ELF file can be displayed using *llvm-objdump -p*[16].

The instruction "la sp, __stack_top" is a pseudo-instruction in RISCV[17].

7.5 QEMU simulator

Install QEMU following the instructions at: https://gitlab.com/qemu-project/qemu as shown below.

```
$ 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, you can compile and run QEMU for bare-metal 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!
```

Also, compile and run QEMU for Linux[18] 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 requires linking with -lm for math.h functions, as shown below:

[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;
}
```

Assembly code of “Hello, World” can be compiled and run in bare-metal mode 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
```

Linking between assembly code and C for bare-metal can be done 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 to the following steps:

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

If you do not have `python3.x-config` on Ubuntu 18.04, as shown below:

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

Then install it using pip3 as follows[19]:

[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 scons
$ ls /usr/bin/python*
... /usr/bin/python3-config
```

After installing all dependencies, clone gem5 and build the RISC-V target 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 scons) ./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 the number of 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 configuration examples for gem5 can be found at the following reference: http://learning.gem5.org/book/part1/example_configs.html

7.7 GDB

LLVM 13.x fails to compile RVV C/C++ files with *clang -g*, while LLVM 14.x works. Run QEMU on terminal A with GDB on terminal B as follows[7]:

```
// terminal A:
$ pwd
$ $HOME/git/lbt/exlbt/riscv
```

(continues on next page)

[7] https://danielmangum.com/posts/risc-v-bytes-qemu-gdb/

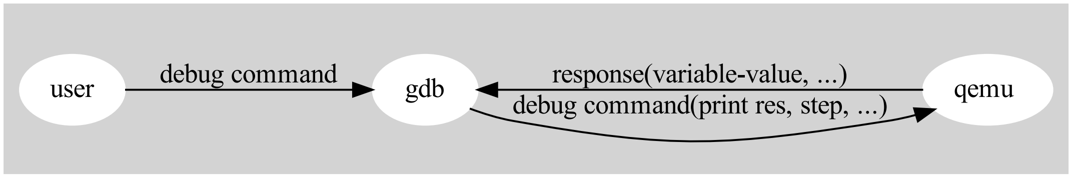

user
debug command
gdb
response(variable-value, ...)
debug command(print res, step, ...)
qemu

Fig. 7.5: GDB flow

(continued from previous page)

```
$ $HOME/riscv/14.x/riscv_newlib/bin/clang vadd1.c -menable-experimental-extensions \
  -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
0x0000000000010150 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:
...
vl: 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
```

(continues on next page)

(continued from previous page)

```
// terminal A:
...
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 v1.0, the above -march option changed from *rv64gv0p10* in LLVM 13.x to *rv64gcv1p0* in LLVM 14.x[8].

In the example above, both QEMU and GDB run in the same host environment.

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 connection[9]. 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 1234[10].

7.8 RISCV Calling Convention[Page 134, 11]

The RV32 register size is 32 bits. The RV64 register size is 64 bits.

In RV64, 32-bit types, such as int, are stored in integer registers with proper sign extension; that is, bits 63..32 are copies of bit 31.

There are two kinds of ABI: ilp32 and lp64, such as -mabi=ilp32, ilp32f, ilp32d, lp64, lp64f, lp64d. They differ in how float arguments are passed on integer, single-float, or double-float registers.

Table 7.2: ABI, caller passing integer/float/double arguments[12][13]

name	float	double
ilp32/lp64	a registers	a registers
ilp32f/lp64f	fa registers	a regsiters
ilp32d/lp64d	fa registers	fa registers

Both ilp32 and lp64 are Soft-Float Calling Conventions.

Soft-Float Calling Convention means floating-point arguments are passed and returned **in integer registers**, using the same rules as integer arguments of the corresponding 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.

[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

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 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 change[14].

7.9 RVV

Clang/LLVM provides RVV (RISC-V Vectors) written in C rather than inline assembly.

Although enabled by the clang option *-target-feature +experimental-v*, using C is shorter, more user-friendly, and easier to remember than inline assembly.

Builtins are C functions and also user-friendly. RVV code 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;
  }
}
```

(continues on next page)

[14] https://gcc.gnu.org/onlinedocs/gcc/RISC-V-Options.html

(continued from previous page)

```
// 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;
    vuint32m8_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)
```

(continues on next page)

(continued from previous page)

```
      printf("...");
    assert(a[i] == i * 2);
  }
  printf("\nThe results of vadd:\tPASS\n");

  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" version 0.1[Page 122, 4].

For *-mabi*, see the section above.

Clang/LLVM provides 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 vfdiv  : RVVFloatingBinBuiltinSet;
```

$HOME/riscv/git/llvm-project/llvm/include/llvm/IR/IntrinsicsRISCV.td

```
def int_riscv_vsetvli   : Intrinsic<...
...
defm vmadd : RISCVTernaryAAXA;
...
defm vfdiv : RISCVBinaryAAX;
```

Refer to Clang/LLVM test cases in the following folders.

```
$HOME/riscv/git/llvm-project/clang/test/CodeGen/RISCV/rvv-intrinsics$ ls
... vfdiv.c ... vfmadd.c

$HOME/riscv/git/llvm-project/llvm/test/CodeGen/RISCV/rvv$ ls
... vfdiv-rv64.ll ... vfmadd-rv64.ll ...
```

7.10 Atomic instructions

RISCV atomic instructions[20].

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 RISCV[21]. FreeBSD[22].

[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

Code[23][24]. Documents[25][26].

7.12.3 Zephyr

RISCV Boards[27].

7.12.4 Andes

Amazon-FreeRTOS[28], Xilinux has RISCV inside in MicroZed[29] and vivado[30].

Zephyr[31] .

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 in exlbt/riscv/riscv-toolchain-setup-macos-intel.sh, this script currently completes building both riscv_newlib and riscv_linux.

However, qemu does not create qemu-riscv64 for baremetal on macOS, so verification cannot be done 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 does not work. It fails without changing crt0.S as well.

```
// terminal A:
riscv % ~/riscv/14.x/riscv_newlib/bin/clang hello.c -menable-experimental-extensions \
-march=rv64gcv1p0 -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:
```

(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-acce
html

(continued from previous page)

```
riscv % ~/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
0x0000000080000408 in ?? ()
(gdb) c
Continuing.
```

The qemu-system-riscv64 with -nographic option also fails to run properly.

CHAPTER

EIGHT

APPENDIX B: CLANG LIBRARY AND TOOLS

This chapter shows how to use 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

Refer example code, 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 apply on Linux and macOS as follows, respectively:

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

Refer to *clang-tools-extra/loop-convert* for examples and usage: [clang-lamt] and [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

References:[1],[2],[3].

RVV reference: ~/riscv/riscv_newlib/lib/clang/13.0.1/include/riscv_vector.h.

[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

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

~/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:
```

(continues on next page)

(continued from previous page)

```
/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;
      ^~~~~~~~~~~
/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;
}
```

(continues on next page)

(continued from previous page)

```
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);
    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) {
```

(continues on next page)

(continued from previous page)

```
  return Mul(Scalar);
}

UVec32& UVec32::operator*(const UVec32 &arg) {
  return Mul(arg);
}

UVec32& UVec32::operator+(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 = 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,
```

(continues on next page)

(continued from previous page)

```
};

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() {
  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;
}
```

References:[4].

[4] https://www.youtube.com/watch?v=-iOtb6luK1Q

CHAPTER

NINE

APPENDIX C: POCL

As Fig. 9.1, one possible HW platform has one Host and Device with 2 RISCV processors, each with dedicated memory.

- info::local_mem_type::global -> coherent, CPU cache (SRAM)
- info::local_mem_type::local -> dedicated local memory, GPU cache (SRAM)

OpenCL includes three components: Runtime API, Driver, and Compiler. POCL is an open-source implementation of the Runtime API.

OpenCL Implementation[1]. Wiki OpenCL Open Source Implementations[2].

9.1 OpenCL

Builtin-function and builtin-kernel reference here[6]. Books[7]. Papers[8][9][10][11][12] in lbt/papers/pocl.

[1] https://www.iwocl.org/resources/opencl-implementations/
[2] https://en.wikipedia.org/wiki/OpenCL#Open_source_implementations
[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

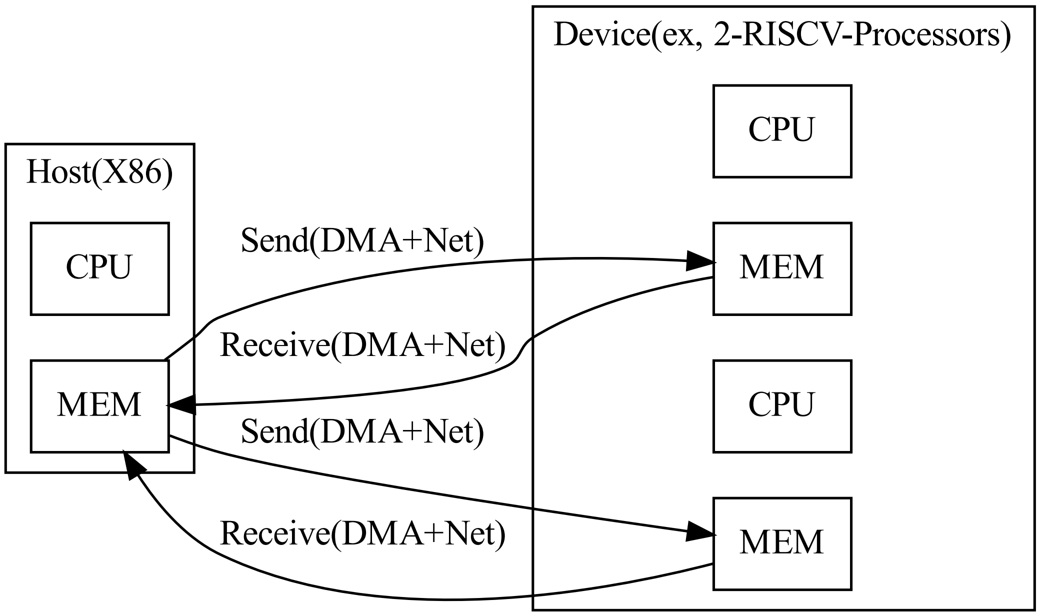


Fig. 9.1: OpenCL with discrete memory (Device: 2 processor+memory)

9.2 Run OpenCL

9.2.1 MacOS

You can create the project ~/git/lbt/exlbt/opencl/opencl-saxpy/opencl-saxpy.xcodeproj by:

- Open Xcode
- Create a new Xcode project
- Select macOS → Command Line Tool
- Set Product Name: opencl-saxpy, Language: C
- Click Next → Create
- Choose folder ~/git/lbt/exlbt/opencl
- On the left, in the opencl-saxpy folder, delete the default main.c → Move to Trash
- Copy saxpy.c into ~/git/lbt/exlbt/opencl/opencl-saxpy/opencl-saxpy
- On the left, right-click opencl-saxpy folder → Add Files to “opencl-saxpy”
- Save the project

Build and run the saxpy OpenCL program:

- Open ~/git/lbt/exlbt/opencl/opencl-saxpy/opencl-saxpy.xcodeproj
- Choose Product → Build
- Choose Product → Run

Refer to the icons shown in Fig. 9.2 to see compile options and run results.

9.3 Open sources

Table 9.1: Open sources for OpenCL

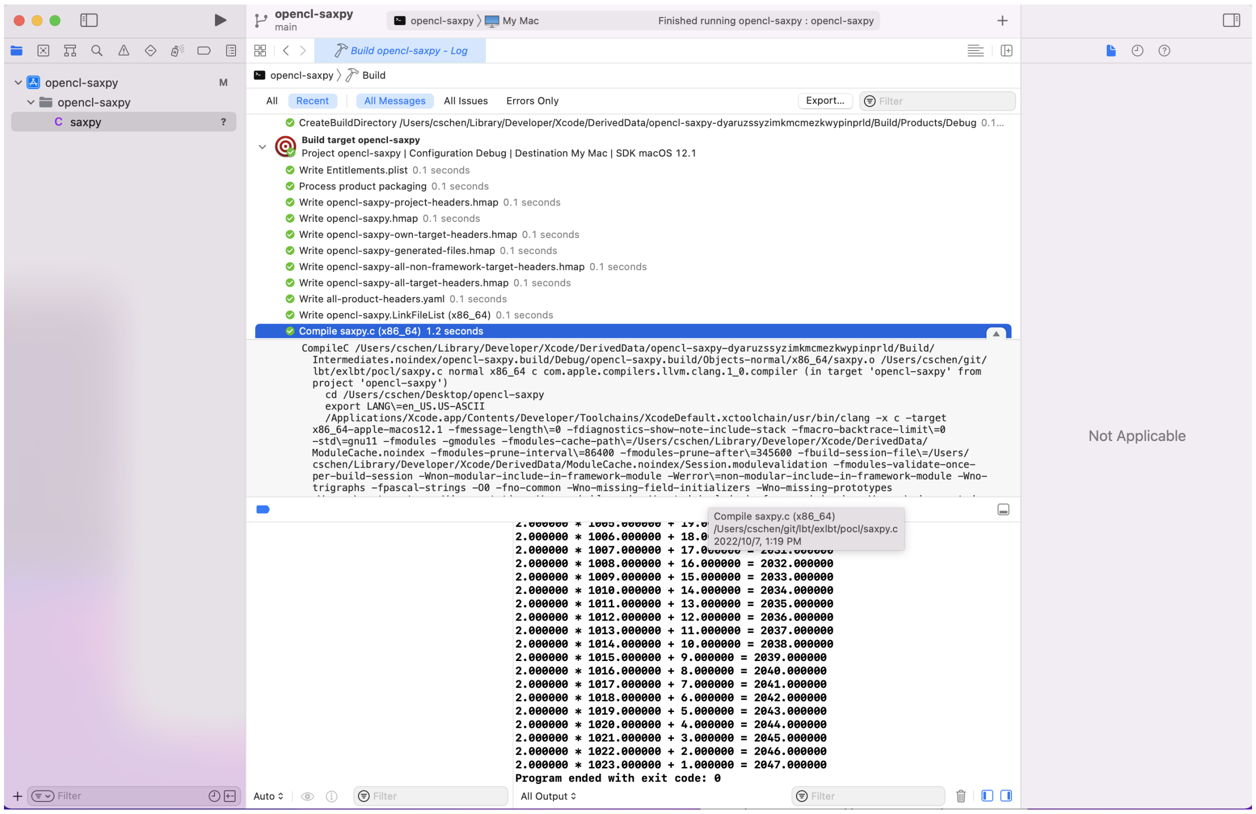
Project	Runtime	Driver	Compiler	Library
POCL	V	V		
clang			V	
libclc				V

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 language[3].

[3] https://libclc.llvm.org



```
CreateBuildDirectory /Users/cschen/Library/Developer/Xcode/DerivedData/opencl-saxpy-dyaruzssyzimkmcmezkwypinprld/Build/Products/Debug 0.1...
Build target opencl-saxpy
Project opencl-saxpy | Configuration Debug | Destination My Mac | SDK macOS 12.1
Write Entitlements.plist 0.1 seconds
Process product packaging 0.1 seconds
Write opencl-saxpy-project-headers.hmap 0.1 seconds
Write opencl-saxpy.hmap 0.1 seconds
Write opencl-saxpy-own-target-headers.hmap 0.1 seconds
Write opencl-saxpy-generated-files.hmap 0.1 seconds
Write opencl-saxpy-all-non-framework-target-headers.hmap 0.1 seconds
Write opencl-saxpy-all-target-headers.hmap 0.1 seconds
Write all-product-headers.yaml 0.1 seconds
Write opencl-saxpy.LinkFileList (x86_64) 0.1 seconds
Compile saxpy.c (x86_64) 1.2 seconds
```

```
CompileC /Users/cschen/Library/Developer/Xcode/DerivedData/opencl-saxpy-dyaruzssyzimkmcmezkwypinprld/Build/
  Intermediates.noindex/opencl-saxpy.build/Debug/opencl-saxpy.build/Objects-normal/x86_64/saxpy.o /Users/cschen/git/
  lbt/exlbt/pocl/saxpy.c normal x86_64 c com.apple.compilers.llvm.clang.1_0.compiler (in target 'opencl-saxpy' from
  project 'opencl-saxpy')
    cd /Users/cschen/Desktop/opencl-saxpy
    export LANG\=en_US.US-ASCII
    /Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/clang -x c -target
  x86_64-apple-macos12.1 -fmessage-length\=0 -fdiagnostics-show-note-include-stack -fmacro-backtrace-limit\=0
  -std\=gnu11 -fmodules -gmodules -fmodules-cache-path\=/Users/cschen/Library/Developer/Xcode/DerivedData/
  ModuleCache.noindex -fmodules-prune-interval\=86400 -fmodules-prune-after\=345600 -fbuild-session-file\=/Users/
  cschen/Library/Developer/Xcode/DerivedData/ModuleCache.noindex/Session.modulevalidation -fmodules-validate-once-
  per-build-session -Wnon-modular-include-in-framework-module -Werror\=non-modular-include-in-framework-module -Wno-
  trigraphs -fpascal-strings -O0 -fno-common -Wno-missing-field-initializers -Wno-missing-prototypes
```

```
Compile saxpy.c (x86_64)
/Users/cschen/git/lbt/exlbt/pocl/saxpy.c
2022/10/7, 1:19 PM
```

```
2.000000 * 1006.000000 + 18.0
2.000000 * 1007.000000 + 17.0
2.000000 * 1008.000000 + 16.000000 = 2032.000000
2.000000 * 1009.000000 + 15.000000 = 2033.000000
2.000000 * 1010.000000 + 14.000000 = 2034.000000
2.000000 * 1011.000000 + 13.000000 = 2035.000000
2.000000 * 1012.000000 + 12.000000 = 2036.000000
2.000000 * 1013.000000 + 11.000000 = 2037.000000
2.000000 * 1014.000000 + 10.000000 = 2038.000000
2.000000 * 1015.000000 + 9.000000 = 2039.000000
2.000000 * 1016.000000 + 8.000000 = 2040.000000
2.000000 * 1017.000000 + 7.000000 = 2041.000000
2.000000 * 1018.000000 + 6.000000 = 2042.000000
2.000000 * 1019.000000 + 5.000000 = 2043.000000
2.000000 * 1020.000000 + 4.000000 = 2044.000000
2.000000 * 1021.000000 + 3.000000 = 2045.000000
2.000000 * 1022.000000 + 2.000000 = 2046.000000
2.000000 * 1023.000000 + 1.000000 = 2047.000000
Program ended with exit code: 0
```

Fig. 9.2: Build and run opencl-saxpy.xcodeproj

9.4 POCL

Pocl web[4] and documentation[5].

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 155, 4].

Note: OpenCL is a C-based language with the keyword *__kernel* and special attributes in data types. If Clang can compile *input.cl* to LLVM IR (*input.ll*), then the LLVM backend can compile *input.ll* and link it with libraries.

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_dependences().
#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_dependences() {
  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/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_dependences;
```

(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
```

Reference[13].

9.5 Structure

The runtime code is located in *pocl/lib/CL*. Test cases for the runtime can be found in *pocl/tests/runtime*.

9.6 GDB on POCL

After running *bash pocl-install.sh*, you can execute *make check_tier1*. However, running a single test such as *./tests/runtime/test_clCreateKernelsInProgram* fails, as shown below:

```
~/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
```

Failing to run a single test executable is not acceptable for tracing the pocl code. A workaround fix is 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
../../poclu/libpoclu.a ../../lib/CL/libOpenCL.so.2.9.0
```

(continues on next page)

[13] http://portablecl.org/docs/html/install.html#configure-build

(continued from previous page)

```
-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
```

Change the link path (*rpath*) from */home/cschen/git/pocl/build/lib/CL* to */usr/local/lib* for *libpocl-devices-basic.so*, then it passes 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
../../poclu/libpoclu.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 traced from Clang compilation options in the last section, I added a *compile.sh* script to run OpenCL programs 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/poclu -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
```

(continues on next page)

(continued from previous page)

```

/usr/lib/llvm-13/bin/clang -g  -pie $FILE.c.o -o a.out  -Wl,-rpath,/usr/local/lib ~/git/
↪pocl/build/poclu/libpoclu.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: 18000000000
  Hello World!
~/git/lbt/exlbt/pocl/ex$ POCL_DEBUG=
~/git/lbt/exlbt/pocl/ex$ ./a.out
  Hello World!
```

References[14].

9.8 RISCV OpenCL

In LLVM, both MIPS and Cpu0 can compile *.ll* files generated by Clang from OpenCL input. However, RISCV fails to do so on both LLVM 14.x and 15.x, as shown below:

```
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

Webs[15]. An example[16].

9.10 DPC++

Book[17].

exlbt/pocl/dpc++.txt

[15] https://www.khronos.org/sycl
[16] https://en.wikipedia.org/wiki/SYCL
[17] https://itbook.store/books/9781484255735

CHAPTER

TEN

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