Tutorial: Creating an LLVM Toolchain for the Cpu0 Architecture

Release 12.0.4

Chen Chung-Shu

Jan 22, 2022

CONTENTS

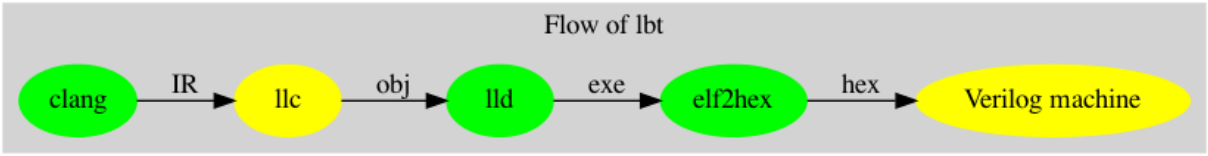


Fig. 1: This book's flow

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 12.0.5, not released yet.

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

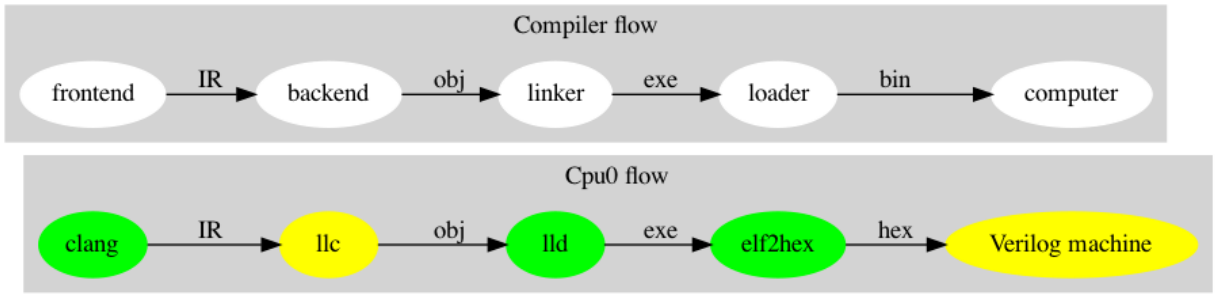


Fig. 1.1: Code generation and execution flow

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

This book include:

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

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.

Cpu0 ELF linker:

Develop ELF linker for Cpu0 backend based on lld project.

Optimization:

Backend independent optimaization.

Library:

Software floating point library and standard C library supporting. Under working.

Clang:

Add Cpu0 target to clang.

CHAPTER

TWO

CLANG

This chapter add Cpu0 target to frontend clang.

2.1 Cpu0 target

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
BUILTIN(__builtin_cpu0_gcd, "iii", "n")

#undef BUILTIN
```

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
//
//===----------------------------------------------------------------------===//

#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,
```

(continues on next page)

(continued from previous page)

```
                      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;

// 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);
  }
```

(continues on next page)

(continued from previous page)

```
  // 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);
    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:
```

(continues on next page)

(continued from previous page)

```
    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")
      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;
```

(continues on next page)

(continued from previous page)

```
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,
                 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;
```

(continues on next page)

(continued from previous page)

```
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
  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;
    }
```

(continues on next page)

(continued from previous page)

```
    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) {
    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"},
```

(continues on next page)

(continued from previous page)

```
        {{"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.
//
//===----------------------------------------------------------------------===//

#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"
```

(continues on next page)

(continued from previous page)

```
};

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 {
    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
```

(continues on next page)

(continued from previous page)

```
    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;
  }

  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
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: 0.12s
```

(continues on next page)

(continued from previous page)

```
  Passed: 1
```

CHAPTER

THREE

CPU0 ELF LINKER

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

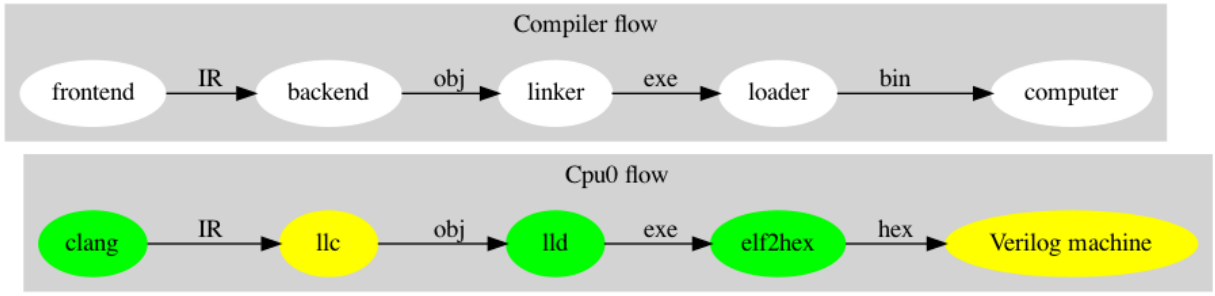


Fig. 3.1: Code generation and execution flow

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

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

About lld please refer LLD web site here[1] and LLD install requirement on Linux here[2]. Currently, lld can be built by: gcc and clang compiler on Ubuntu. On iMac, lld can be built by clang with the Xcode version as the next sub section. If you run with Virtual Machine (VM), please keep your phisical memory size setting over 1GB to avoid insufficient memory link error.

3.1 ELF to Hex

As follows,

exlbt/elf2hex/CMakeLists.txt

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

add_llvm_tool(elf2hex
  elf2hex.cpp
  )

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

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

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

exlbt/elf2hex/elf2hex.h

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

#ifndef LLVM_TOOLS_ELF2HEX_ELF2HEX_H
#define LLVM_TOOLS_ELF2HEX_ELF2HEX_H

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

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

#define BOOT_SIZE 16

#define DLINK
//#define ELF2HEX_DEBUG

namespace llvm {

namespace elf2hex {

using namespace object;

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

//using namespace llvm;

#endif
```

exlbt/elf2hex/elf2hex.cpp

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

(continues on next page)

(continued from previous page)

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

#define ELF2HEX

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

using namespace llvm;
using namespace llvm::object;

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)
    return std::move(*EO);
  assert(0 && "error in unwrapOrError()");
}

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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


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

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

(continues on next page)

(continued from previous page)

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

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

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

(continues on next page)

(continued from previous page)

```
    BaseAddr = Section.getAddress();

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

using namespace llvm::elf2hex;

Reader reader;

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

  return;
}

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

    StringRef name;
    getName(Section, name);

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

// Porting from disassembleObject() of llvm-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);

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

  Elf2Hex(o);
}

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

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

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

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

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

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

  // Register the target printer for --version.
```

(continues on next page)

(continued from previous page)

```
  cl::AddExtraVersionPrinter(TargetRegistry::printRegisteredTargetsForVersion);

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

  ToolName = argv[0];

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

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

  return EXIT_SUCCESS;
}
```

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

exlbt/llvm-objdump/llvm-objdump.cpp

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

3.2 Create Cpu0 backend under LLD

3.2.1 LLD introduction

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

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

a.h

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

a.cpp

```
#include "a.h"

static signed int i = 0;

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

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

int foo1(void) {
  int data = 0;

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

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

ch13_1.cpp

```
#include "a.h"

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

int foo4(void) {
  return 5;
}

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

Above code can be reduced to Fig. 3.2 to perform mark and swip in graph for Dead Code Stripping.

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

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

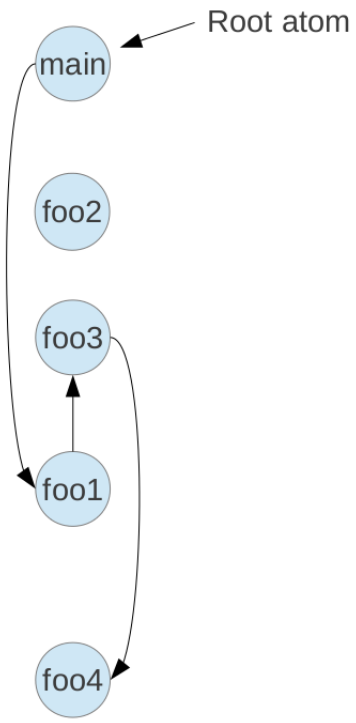


Fig. 3.2: Atom classified (from lld web)

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

3.2.2 Static linker

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

File printf-stdarg.c come from internet download which is GPL2 license. GPL2 is more restricted than LLVM license. File printf-stdarg-1.c is the file for testing the printf() function which implemented on PC OS platform. Let's run printf-stdarg-2.cpp on Cpu0 and compare it against the result of PC's printf() as below.

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
```

(continues on next page)

(continued from previous page)

```
    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;
  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);
```

(continues on next page)

(continued from previous page)

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

#endif
```

exlbt/input/printf-stdarg-2.cpp

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

#define TEST_PRINTF

extern "C" int putchar(int c);

extern "C" {
#include "printf-stdarg.c"
}
```

exlbt/input/printf-stdarg-def.c

```
#include "print.h"

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

  return 0;
}
```

exlbt/input/printf-stdarg.c

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

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

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

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

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

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

#include <stdarg.h>

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

(continues on next page)

(continued from previous page)

```
  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)
{
  register int pc = 0, padchar = ' ';

  if (width > 0) {
    register int len = 0;
    register 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

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

  if (i == 0) {
    print_buf[0] = '0';
```

(continues on next page)

(continued from previous page)

```
    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 )
{
  register int width, pad;
  register int pc = 0;
  char scr[2];

  for (; *format != 0; ++format) {
    if (*format == '%') {
      ++format;
      width = pad = 0;
      if (*format == '\0') break;
      if (*format == '%') goto out;
      if (*format == '-') {
        ++format;
        pad = PAD_RIGHT;
      }
      //bool have_ll = (format[0] == 'l' && format[1] == 'l');
      //pc += have_ll * 2;
```

(continues on next page)

(continued from previous page)

```
      while (*format == '0') {
        ++format;
        pad |= PAD_ZERO;
      }
      for ( ; *format >= '0' && *format <= '9'; ++format) {
        width *= 10;
        width += *format - '0';
      }
      if( *format == 's' ) {
        register 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;
    }
  }
  if (out) **out = '\0';
  va_end( args );
  return pc;
}

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

(continues on next page)

(continued from previous page)

```
        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);
  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
```

(continues on next page)

(continued from previous page)

```
 * 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 "dynamic_linker.h"
#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
  int *gpaddr;
  gpaddr = (int*)GPADDR;
  __asm__ __volatile__("ld  $gp, %0"
                       : // no output register, specify output register to $gp
                       :"m"(*gpaddr)
                       );
  initRegs();
  main();
  asm("addiu $lr, $ZERO, -1");
```

(continues on next page)

(continued from previous page)

```
  asm("ret $lr");
}
```

exIbt/input/lib_cpu0.ll

```
; The @_start() exist to prevent lld linker error.
; Real entry (first instruction) is from cpu0BootAtomContent of
; Cpu0RelocationPass.cpp jump to asm("start:") of start.cpp.
define void @_start() nounwind {
entry:
  ret void
}

define void @__start() nounwind {
entry:
  ret void
}

define void @__stack_chk_fail() nounwind {
entry:
  ret void
}

define void @__stack_chk_guard() nounwind {
entry:
  ret void
}

define void @_ZdlPv() nounwind {
entry:
  ret void
}

define void @__dso_handle() nounwind {
entry:
  ret void
}

define void @_ZNSt8ios_base4InitC1Ev() nounwind {
entry:
  ret void
}

define void @__cxa_atexit() nounwind {
entry:
  ret void
}

define void @_ZTVN10__cxxabiv120__si_class_type_infoE() nounwind {
```

(continues on next page)

(continued from previous page)

```
entry:
  ret void
}

define void @_ZTVN10__cxxabiv117__class_type_infoE() nounwind {
entry:
  ret void
}

define void @_Znwm() nounwind {
entry:
  ret void
}

define void @__cxa_pure_virtual() nounwind {
entry:
  ret void
}

define void @_ZNSt8ios_base4InitD1Ev() nounwind {
entry:
  ret void
}
```

exlbt/input/Common.mk

```
# Thanks https://makefiletutorial.com

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

TOOLDIR := ~/llvm/test/build/bin
CC := $(TOOLDIR)/clang
LLC := $(TOOLDIR)/llc
LD := $(TOOLDIR)/ld.lld

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

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

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

(continues on next page)

(continued from previous page)

```
# 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 ${INC_FLAGS} -mcpu=${CPU} -mllvm -has-lld=true

LLFLAGS := -march=cpu0${ENDIAN} -mcpu=${CPU} -relocation-model=static \
  -filetype=obj -has-lld=true

#FIND_LIBFLOAT_DIR := $(shell find . -iname $(LIBFLOAT_DIR))

$(TARGET): $(OBJS) $(LIBS)
        echo "LIBFLOAT_DIR: $(LIBFLOAT_DIR), LIBS: $(LIBS)"
        $(LD) -o $@ $(OBJS) $(LIBS)

$(LIBS):
        $(MAKE) -C $(LIBFLOAT_DIR)

# 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 $@

$(BUILD_DIR)/lib_cpu0.ll.o: lib_cpu0.ll
        $(LLC) $(LLFLAGS) $< -o $@

.PHONY: clean
clean:
        rm -rf $(BUILD_DIR)
ifdef LIBFLOAT_DIR
        cd $(LIBFLOAT_DIR) && $(MAKE) -f Makefile clean
endif

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

exlbt/input/Makefile.printf-stdarg-2

```
# CPU and endian passed from command line, such as
#   "make -f Makefile.printf-stdarg-2 CPU=cpu032I ENDIAN=el"

SRCS := start.cpp debug.cpp printf-stdarg-def.c printf-stdarg-2.cpp lib_cpu0.ll
INC_DIRS := $(SRC_DIR) ../../lbdex/input
LIBFLOAT_DIR :=
LIBS :=

include Common.mk
```

The Makefile.printf-stdarg-2 is for my PC setting. Please change this script to the directory of your llvm/lld setting. After that run static linker example code as follows,

```
1-160-136-173:input Jonathan$ pwd
/Users/Jonathan/Downloads/exlbt/input
1-160-136-173:input Jonathan$ bash Makefile.printf-stdarg-2 cpu032I be
In file included from printf-stdarg-2.cpp:11:
./printf-stdarg.c:206:15: warning: conversion from string literal to 'char *'
is deprecated [-Wdeprecated-writable-strings]
  char *ptr = "Hello world!";
              ^
1 warning generated.

1-160-136-173:input Jonathan$ cd ../../lbdex/verilog/
1-160-136-173:verilog Jonathan$ pwd
/Users/Jonathan/Download/lbdex/verilog
1-160-136-173:verilog Jonathan$ make
1-160-136-173:verilog Jonathan$ ls
... cpu0Is ... cpu0IIs ...
1-160-136-173:verilog Jonathan$ ./cpu0Is
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
```

Let's check the result with PC program printf-stdarg-1.c output as follows,

```
1-160-136-173:input Jonathan$ clang printf-stdarg-1.c
```

(continues on next page)

(continued from previous page)

```
printf-stdarg-1.c:58:19: warning: incomplete format specifier [-Wformat]
  printf("%d %s(s)%", 0, "message");
                  ^
1 warning generated.
1-160-136-173:input Jonathan$ ./a.out
Hello world!
printf test
(null) is null pointer
5 = 5
-2147483647 = - max int
char a = 'a'
hex ff = ff
hex 00 = 00
signed -3 = unsigned 4294967293 = hex 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.
```

They are same. You can verify the slt instructions is work fine too by change variable cpu from cpu032I to cpu032II as follows,

exlbt/input/Makefile.printf-stdarg-2

```
1-160-136-173:verilog Jonathan$ pwd
/Users/Jonathan/Download/lbdex/verilog
1-160-136-173:verilog Jonathan$ cd ../../exlbt/input
1-160-136-173:input Jonathan$ pwd
/Users/Jonathan/Download/exlbt/input
1-160-136-173:input Jonathan$ bash Makefile.printf-stdarg-2 cpu032II be
...
1-160-136-173:input Jonathan$ cd ../lbdex/verilog/
1-160-136-173:verilog Jonathan$ ./cpu0IIs
```

The verilog machine cpu0IIs include all instructions of cpu032I and add slt, beq, . . . , instructions. Run Makefile.printf-stdarg-2 with cpu=cpu032II will generate slt, beq and bne instructions instead of cmp, jeq, . . . instructions.

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

exlbt/input/debug.cpp

```
#include "debug.h"

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

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

(continues on next page)

(continued from previous page)

```
    }
    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();
```

(continues on next page)

(continued from previous page)

```
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 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;
}
```

(continues on next page)

(continued from previous page)

```
int test_staticlink()
{
  int a = 0;

  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();
  printf("test_template() = %d", a); // a = 15
  if (a == 15)
```

(continues on next page)

(continued from previous page)

```
    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

```
# CPU and endian passed from command line, such as
#   "make -f Makefile.slinker CPU=cpu032I endian=le"

SRCS := start.cpp debug.cpp printf-stdarg-def.c printf-stdarg.c ch_slinker.cpp \
        lib_cpu0.ll
INC_DIRS := $(SRC_DIR) ../../lbdex/input
LIBFLOAT_DIR :=
LIBS :=

include Common.mk
```

exlbt/input/make.sh

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

# for example:
# bash make.sh cpu032II be Makefile.builtins
# bash make.sh cpu032I le Makefile.slinker
# bash make.sh cpu032II be Makefile.float
# bash make.sh cpu032II be Makefile.printf-stdarg-2
# bash make.sh cpu032II be Makefile.ch13_1
# bash make.sh cpu032I le Makefile.sanitizer-printf

ARG_NUM=$#
CPU=$1
```

(continues on next page)

(continued from previous page)

```
ENDIAN=$2

prologue() {
  LBDEXDIR=../../lbdex

  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 be"
    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}"

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

  bash clean.sh
  builtins="../libsoftfloat/compiler-rt/builtins"
  if [ ! -L $builtins ]; then
    ln -s $HOME/llvm/llvm-project/compiler-rt/lib/builtins $builtins
  fi
}

isLittleEndian() {
  echo "ENDIAN = " "$ENDIAN"
  if [ "$ENDIAN" == "LittleEndian" ] ; then
    le="true"
  elif [ "$ENDIAN" == "BigEndian" ] ; then
    le="false"
  else
```

(continues on next page)

(continued from previous page)

```
    echo "!ENDIAN unknown"
    exit 1
  fi
}

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

epilogue() {
  ENDIAN=`${TOOLDIR}/llvm-readobj -h 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}

cp ./build/a.out .

epilogue;
```

```
1-160-136-173:input Jonathan$ pwd
/Users/Jonathan/Downloads/exlbt/input
114-37-148-111:input Jonathan$ bash make.sh cpu032I le Makefile.slinker
...
endian =  LittleEndian
ISR address:00020780
1   /* 0: big endian, 1: little endian */
chungshu@ChungShudeMacBook-Air 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
```

(continues on next page)

(continued from previous page)

```
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
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!
```

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

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

3.2.3 Dynamic linker

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

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

Then reading this section in lld.html for it.

3.3 Summary

3.3.1 Create a new backend base on LLVM

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

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

3.3.2 Contribute back to Open Source through working and learning

Finally, 10 thousands lines of source code in Cpu0 backend is very small in UI program. But it's quite complex in system program which based on llvm. We spent 600 pages of pdf to explain these code. Open source code give programmers best opportunity to understand the code and enhance/extend the code function. But it can be better, we believe the documentation is the next most important thing to improve the open source code development. The Open Source Organization recognized this point and set Open Source Document Project years ago[7][8][9][10][11]. Open Source grows up and becomes a giant software infrastructure with the forces of company[12][13], school research team and countless talent engineers passion. It terminated the situation of everyone trying to re-invent wheels during 10 years ago. Extend your software from the re-usable source code is the right way. Of course you should consider an open source license if you are working with business. Actually anyone can contribute back to open source through the learning process. This book is written through the process of learning llvm backend and contribute back to llvm open source project. We think this book cannot exists in traditional paper book form since only few number of readers interested in study llvm backend even though there are many paper published books in concept of compiler. So, this book is published via electric media form and try to match the Open Document License Expection[14]. There are distance between the concept and the realistic program implemenation. Keep note through learning a large complicate software such as this llvm backend is not enough. We all learned the knowledge through books during school and after school. So, if you cannot find a good way to produce documents, you can consider to write documents like this book. This book document uses sphinx tool just like the llvm development team. Sphinx uses restructured text format here[15][16][17]. Appendix A of lbd book tell you how to install sphinx tool. Documentation work will help yourself to re-examine your software and make your program better in structure, reliability and more important "Extend your code to somewhere you didn't expect".

[7] http://en.wikipedia.org/wiki/BSD_Documentation_License
[8] http://www.freebsd.org/docproj/
[9] http://www.freebsd.org/copyright/freebsd-doc-license.html
[10] http://en.wikipedia.org/wiki/GNU_Free_Documentation_License
[11] http://www.gnu.org/copyleft/fdl.html
[12] http://www.apple.com/opensource/
[13] https://www.ibm.com/developerworks/opensource/
[14] http://www.gnu.org/philosophy/free-doc.en.html
[15] http://docutils.sourceforge.net/docs/ref/rst/restructuredtext.html
[16] http://docutils.sourceforge.net/docs/ref/rst/directives.html
[17] http://docutils.sourceforge.net/rst.html

CHAPTER

FOUR

OPTIMIZATION

This chapter introduce llvm optimization.

4.1 LLVM IR optimization

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

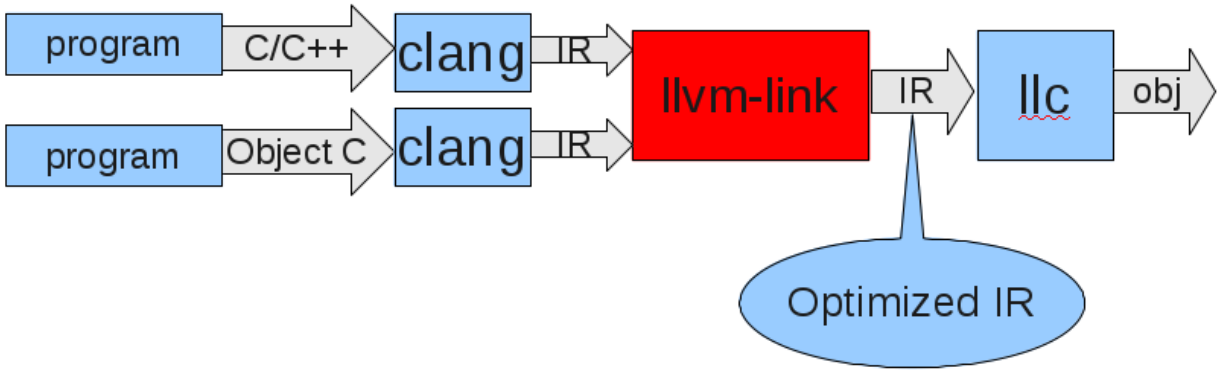


Fig. 4.1: llvm-link flow

Clang provide optimization options to do optimation from high level language to IR. But since many languages like C/C++ support separate compilation, it meaning there is no chance to do inter-procedure optimization if the functions come from different source files. To solve this problem, llvm provide **llvm-link** to link all *.bc into a single IR file, and through **opt** to finish the inter-procedure optimation[1]. Beyond the DAG local optimization mentioned in Chapter 2, there are global optimization based on inter-procedure analysis[2]. The following steps and examples show this optimization solution in llvm.

[1] http://www.cs.cmu.edu/afs/cs/academic/class/15745-s12/public/lectures/L3-LLVM-Part1.pdf
[2] Refer chapter 9 of book Compilers: Principles, Techniques, and Tools (2nd Edition)

exlbt/input/optimizen/1.cpp

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

exlbt/input/optimize/2.cpp

```
extern int callee(const int *X);

int caller() {
  int T;

  T = 4;

  return callee(&T);
}
```

```
JonathantekiiMac:input Jonathan$ clang -O3 -target mips-unknown-linux-gnu
-c 1.cpp -emit-llvm -o 1.bc
JonathantekiiMac:input Jonathan$ clang -O3 -target mips-unknown-linux-gnu
-c 2.cpp -emit-llvm -o 2.bc
JonathantekiiMac:input Jonathan$ llvm-link -o=a.bc 1.bc 2.bc
JonathantekiiMac:input Jonathan$ opt -O3 -o=a1.bc a.bc
JonathantekiiMac:input Jonathan$ llvm-dis a.bc -o -
...
; Function Attrs: nounwind readonly
define i32 @_Z6calleePKi(i32* nocapture readonly %a) #0 {
  %1 = load i32* %a, align 4, !tbaa !1
  %2 = add nsw i32 %1, 1
  ret i32 %2
}

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

JonathantekiiMac:input Jonathan$ llvm-dis a1.bc -o -
...
; Function Attrs: nounwind readonly
define i32 @_Z6calleePKi(i32* nocapture readonly %a) #0 {
  %1 = load i32* %a, align 4, !tbaa !1
  %2 = add nsw i32 %1, 1
  ret i32 %2
}
```

(continues on next page)

(continued from previous page)

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

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

```
JonathantekiiMac:input Jonathan$ ~/llvm/test/build/
bin/llc -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 will 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

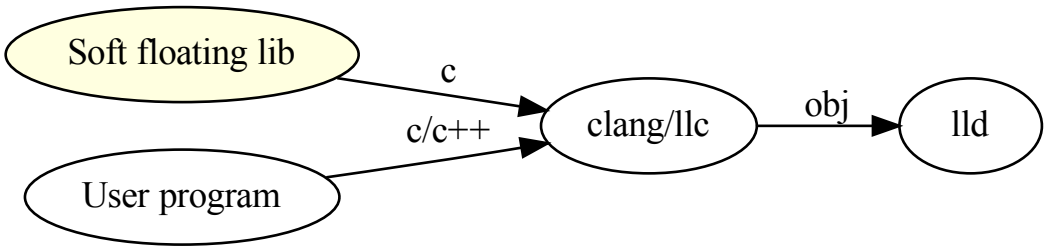
FIVE

LIBRARY

Since Cpu0 has not hardware float point instructions, it needs soft float point library to finish the floating point operation. LLVM compiler-rt project include software floating point library implementation Fig. 5.1 , so we choose it as the implementation.

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

Fig. 5.1: compiler-rt/lib/builtins' software float library



5.1 Compiler-rt

Table 5.1: Toolchain components[Page 70, 2]

Component	LLVM	GNU[Page 70, 1]
C/C++ Compiler	clang/llvm	gcc
Assembler	llvm integrated assembler	as
Linker	ld.lld	ld.bfd ld.gold
Runtime	compiler-rt	libgcc[3]
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 soft float library is here[4] . Compiler-rt is a project with runtime libraries implentation[5] . Compiler-rt/lib/builtins provides functions for basic operations such as +, -, *, /, ... on type of float or double and for conversion between float and integer. Though the 'rt' means RunTime libaraies, most of these functions written in target-independent C form and can be compiled and static-linked into target. When you compile the following c code, llc generates **jsub __addsf3** to call compiler-rt float function since Cpu0 hasn't hardware float-instructions so Cpu0 backend doesn't handle it, and llvm treats it as a function call for 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 -o ch_call_compilerrt_func.ll
chungshu@ChungShudeMacBook-Air input % cat ch_call_compilerrt_func.ll
  ...
  %4 = load float, float* %1, align 4
```

(continues on next page)

[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
[1] https://en.wikipedia.org/wiki/GNU_Compiler_Collection#cite_note-55
[3] https://gcc.gnu.org/onlinedocs/gccint/Libgcc.html
[4] https://gcc.gnu.org/onlinedocs/gccint/Soft-float-library-routines.html#Soft-float-library-routines
[5] http://compiler-rt.llvm.org/

(continued from previous page)

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

chungshu@ChungShudeMacBook-Air input % ~/llvm/test/build/bin/llc -march=cpu0 -
↪mcpu=cpu032II -relocation-model=static -filetype=asm ch_call_compilerrt_func.ll -o -
      ...
      ld      $4, 20($fp)
      ld      $5, 16($fp)
      jsub    __addsf3
```

For some brar-metal or embedded application, the C code doesn't need the file and high-level IO in libc. Libm provides a lots of functions to support software floating point beyond basic operations[6] . Libc provides file, high-level IO functions and basic float functions[7] .

Cpu0 hires Compiler-rt/lib/builtins and the tiny single module printf-stdarg.c[8] at this point. Directory exlib/libsoftfloat/compiler-rt-12.x/builtins is a symbolic link to llvm-project/compiler-rt/lib/builtins which is the floating point library from compiler-rt[?]. The compiler-rt/lib/builtins is a target-independent C form of software float library implementation. Cpu0 implements compiler-rt-12.x/cpu0/abort.c only at this point for supporting this feature.

5.2 Software Float Point Support

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

exlbt/include/math.h

```
#ifndef _MATH_H_
#define        _MATH_H_

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

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

 #if !defined(NAN)
  #define NAN (0.0F/0.0F)
 #endif

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

[6] https://www.programiz.com/c-programming/library-function/math.h
[7] https://www.cplusplus.com/reference/clibrary
[8] https://github.com/atgreen/FreeRTOS/blob/master/Demo/CORTEX_STM32F103_Primer_GCC/printf-stdarg.c

exlbt/include/stdio.h

```
#ifndef _STDIO_H_
#define        _STDIO_H_

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

#define size_t unsigned int

#endif
```

exlbt/include/stdlib.h

```
#ifndef _STDLIB_H_
#define        _STDLIB_H_

#ifdef __cplusplus
extern "C" {
#endif

void abort();

#ifdef __cplusplus
}
#endif

#endif
```

exlbt/include/string.h

```
#ifndef _STRING_H_
#define        _STRING_H_


#endif
```

exlbt/libsoftfloat/compiler-rt/cpu0/abort.c

```
/* Copyright (c) 2002, Marek Michalkiewicz
   All rights reserved.

   Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions are met:

   * Redistributions of source code must retain the above copyright
     notice, this list of conditions and the following disclaimer.
```

(continues on next page)

(continued from previous page)

```
   * Redistributions in binary form must reproduce the above copyright
     notice, this list of conditions and the following disclaimer in
     the documentation and/or other materials provided with the
     distribution.
   * Neither the name of the copyright holders nor the names of
     contributors may be used to endorse or promote products derived
     from this software without specific prior written permission.

  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
  AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
  IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
  ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE
  LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
  CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
  SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
  INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
  CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
  ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE
  POSSIBILITY OF SUCH DAMAGE. */


void
abort(void)
{
        for (;;);
}
```

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

// 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.
```

(continues on next page)

(continued from previous page)

```
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);

static void abort() {
  for (;;);
}

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

namespace __sanitizer {

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

// Appends number in a given base to buffer. If its length is less than
// |minimal_num_length|, it is padded with leading zeroes or spaces, depending
// on the value of |pad_with_zero|.
static int AppendNumber(char **buff, const char *buff_end, u64 absolute_value,
                        u8 base, u8 minimal_num_length, bool pad_with_zero,
                        bool negative, bool uppercase) {
  uptr const kMaxLen = 30;
  RAW_CHECK(base == 10 || base == 16);
```

(continues on next page)

(continued from previous page)

```
  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 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) {
#if 1
    abort();
#else
    // Make sure compiler doesn't insert call to memset here.
    internal_memset(&num_buffer[pos], 0,
                    sizeof(num_buffer[0]) * (minimal_num_length - pos));
    pos = minimal_num_length;
#endif
  }
  RAW_CHECK(pos > 0);
  pos--;
  for (; pos >= 0 && num_buffer[pos] == 0; pos--) {
    char c = (pad_with_zero || pos == 0) ? '0' : ' ';
    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);
  }
  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) {
  return AppendNumber(buff, buff_end, num, base, minimal_num_length,
                      pad_with_zero, false /* negative */, uppercase);
}

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

(continues on next page)

(continued from previous page)

```
                      false /* uppercase */);
}


// 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, not implemented yet
static int AppendString(char **buff, const char *buff_end, int width,
                        int max_chars, const char *s) {
  if (!s)
    s = "<null>";
  int result = 0;
  for (; *s; s++) {
    if (max_chars >= 0 && result >= max_chars)
      break;
    result += AppendChar(buff, buff_end, *s);
  }
  // Only the left justified strings are supported.
  while (width < -result)
    result += AppendChar(buff, buff_end, ' ');
  return result;
}

static int AppendPointer(char **buff, const char *buff_end, u64 ptr_value) {
  int result = 0;
  result += AppendString(buff, buff_end, 0, -1, "0x");
  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 */);
  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;
    }
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
      }
      case 's': {
        RAW_CHECK_MSG(!have_length, kPrintfFormatsHelp);
        // Only left-justified width is supported.
        CHECK(!have_width || left_justified);
        result += AppendString(&buff, buff_end, left_justified ? -width : width,
                               precision, va_arg(args, char*));
        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)
{
  int pc = 0, padchar = ' ';

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

  return pc;
}

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;
```

(continues on next page)

(continued from previous page)

```
}
```

exlbt/input/ch_hello.c

```
// ~/llvm/release/build/bin/clang -target mips-unknown-linux-gnu -c hello.cpp -emit-llvm␣
↪-o hello.bc
// ~/llvm/test/build/bin/llc -march=cpu0 -mcpu=cpu032I -relocation-model=static -
↪filetype=obj ch_hello.bc -o ch_hello.cpu0.o

// start
extern int printf(const char *format, ...);

int main(void)
{
  char ptr[] = "Hello world!";

  long long a = 0x100000007fffffff;
  printf("%s\n", ptr);
  printf("a: %llX, %llx, %lld\n", a, a, a);
  int b = 0x10000000;
  printf("b: %x, %d\n", b, b);

  return 0;
}
```

exlbt/input/Makefile.sanitizer-printf

```
# CPU and endian passed from command line, such as
#   "make -f Makefile.sanitizer_printf CPU=cpu032II endian="
#   Ignore argument and set CPU to cpu032II since software float point library,
#   libsoftfloat, only supports cpu032II at this point.

SRCS := start.cpp debug.cpp sanitizer_printf.cpp printf-stdarg-def.c ch_hello.c \
        lib_cpu0.ll
LIBFLOAT_DIR := ../libsoftfloat/compiler-rt/
INC_DIRS := ./ ../include ../../lbdex/input $(HOME)/llvm/test/compiler-rt/lib/sanitizer_
↪common
LIBS := $(LIBFLOAT_DIR)/build/libFloat.a

include Common.mk
```

```
cschen@cschendeiMac input %  bash make.sh cpu032I le Makefile.sanitizer-printf

cschen@cschendeiMac verilog % ./cpu0Is
...
Hello world!
a: 100000007FFFFFFF, 100000007fffffff, 1152921506754330623
b: 10000000, 268435456
```

(continues on next page)

(continued from previous page)

```
total cpu cycles = 1266990
RET to PC < 0, finished!
```

The following ch_float.cpp test the float lib.

lbt/exlbt/libsoftfloat/compiler-rt-12.x/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 := libFloat.a
BUILD_DIR := ./build
TARGET := $(BUILD_DIR)/$(TARGET_LIB)

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

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 \
```

(continues on next page)

(continued from previous page)

```
  divmoddi4.c \
  divmodsi4.c \
  divmodti4.c \
  divsc3.c \
  divsf3.c \
  divsi3.c \
  divti3.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 \
```

(continues on next page)

(continued from previous page)

```
  mulvdi3.c \
  mulvsi3.c \
  mulvti3.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)/%) cpu0/abort.c

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

# String substitution (suffix version without %).
# As an example, ./build/absvdi2.c.o turns into ./build/absvdi2.c.d
```

(continues on next page)

(continued from previous page)

```
DEPS := $(OBJS:.o=.d)

# Every folder in ./src will need to be passed to GCC so that it can find header files
# stdlib.h, ..., etc existed in ../../include
INC_DIRS := $(shell find $(SRC_DIR) -type d)  ../../include
# Add a prefix to INC_DIRS. So moduleA would become -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 "../../lbdex/input/ch9_3_longlongshift.cpp"

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);
```

(continues on next page)

(continued from previous page)

```
}

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
  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>(0x0016666660000000a, 48LL); // 0x16 = 22
  check_result("test_shift_right<long long>(0x0016666660000000a, 48LL)", a, 22LL);

// call __lshrdi3
  b = test_shift_right<unsigned long long>(0x0016666660000000a, 48LLu); // 0x16 = 22
  check_result("test_shift_right<unsigned long long>(0x0016666660000000a, 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);
```

(continues on next page)

(continued from previous page)

```
  return 0;
}
```

exlbt/input/Makefile.float

```
# CPU and endian passed from command line, such as
#   "make -f Makefile.float CPU=cpu032II ENDIAN=el"
#   Ignore argument and set CPU to cpu032II since software float point library,
#   libsoftfloat, only supports cpu032II at this point.

SRCS := start.cpp debug.cpp sanitizer_printf.cpp printf-stdarg-def.c ch_float.cpp lib_
↪cpu0.ll
LIBFLOAT_DIR := ../libsoftfloat/compiler-rt
INC_DIRS := ../ ../include ../../lbdex/input $(HOME)/llvm/test/compiler-rt/lib/sanitizer_
↪common
LIBS := $(LIBFLOAT_DIR)/build/libFloat.a

include Common.mk
```

```
chungshu@ChungShudeMacBook-Air input % bash make.sh cpu032II be 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
WARNING: cpu0.v:489: $readmemh(cpu0.hex): Not enough words in the file for the requested␣
↪range [0:524287].
taskInterrupt(001)
test_longlong_shift1() = 289
test_longlong_shift2() = 22
test_shift_left<long long>(0x12, 4LL) = 288
test_shift_right<long long>(0x0016666600000000a, 48LL) = 22
test_shift_right<unsigned long long>(0x0016666600000000a, 48LLu) = 22
(int)test_add<float, float, float>(-2.2, 3.3) = 1
(int)test_mul<float, float, float>(-2.2, 3.3) = -7
(int)test_div<float, float, float>(-1.8, 0.5) = -3
(int)test_add<double, double, float>(-2.2, 3.3) = 1
(int)test_add<double, float, double>(-2.2, 3.3) = 1
(int)test_add<float, float, double>(-2.2, 3.3) = 1
(int)test_mul<double, float, double>(-2.2, 3.3) = -7
(int)test_mul<float, float, double>(-2.2, 3.3) = -7
(int)test_div<double, double, double>(-1.8, 0.5) = -3
total cpu cycles = 240170
RET to PC < 0, finished!
```

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

exlbt/input/ch_builtins.cpp

```
#include <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();
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: cabs, 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
```

(continues on next page)

(continued from previous page)

```
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();
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
```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```
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();
  }
}

int main() {
  int res = 0;

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

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

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

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

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

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

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

  res = addvti3_test();
```

(continues on next page)

(continued from previous page)

```
  show_result("addvti3_test()", res);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

```
#endif

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  res = muldf3vfp_test();
```

(continues on next page)

(continued from previous page)

```
  show_result("muldf3vfp_test()", res);

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  res = nedf2vfp_test();
```

(continues on next page)

(continued from previous page)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

  res = popcounttti2_test();
  show_result("popcounttti2_test()", res);

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

  res = powisf2_test();
```

(continues on next page)

(continued from previous page)

```
  show_result("powisf2_test()", res);

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

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

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

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

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

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

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

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

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

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

  res = 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();
```

(continues on next page)

(continued from previous page)

```
  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);

  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.float-necessary CPU=cpu032II endian="
#   Ignore argument and set CPU to cpu032II since software float point library,
#   libsoftfloat, only supports cpu032II at this point.

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

(continues on next page)

(continued from previous page)

```
  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 \
  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 \
```

(continues on next page)

(continued from previous page)

```
  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 \
  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 \
  ch_builtins.cpp lib_cpu0.ll
INC_DIRS := ./ ../../lbdex/input ../libsoftfloat/compiler-rt/builtins ../include
LIBFLOAT_DIR := ../libsoftfloat/compiler-rt
LIBS := $(LIBFLOAT_DIR)/build/libFloat.a

include Common.mk
```

Run as follows,

```
chungshu@ChungShudeMacBook-Air input % bash make.sh cpu032II be 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!
```

(continues on next page)

(continued from previous page)

```
addvsi3_test(): PASS!
addvti3_test(): SKIPPED!
ashldi3_test(): PASS!
ashlti3_test(): SKIPPED!
ashrdi3_test(): PASS!
ashrti3_test(): SKIPPED!
bswapdi2_test(): PASS!
bswapsi2_test(): PASS!
clzdi2_test(): PASS!
clzsi2_test(): PASS!
clzti2_test(): SKIPPED!
cmpdi2_test(): PASS!
cmpti2_test(): SKIPPED!
comparedf2_test(): PASS!
comparesf2_test(): PASS!
cpu_model_test(): SKIPPED!
ctzdi2_test(): PASS!
ctzsi2_test(): PASS!
ctzti2_test(): SKIPPED!
divdf3_test(): PASS!
divdf3vfp_test(): SKIPPED!
divdi3_test(): PASS!
divmodsi4_test(): PASS!
divmodti4_test(): SKIPPED!
divsf3_test(): PASS!
divsf3vfp_test(): SKIPPED!
divsi3_test(): PASS!
divtf3_test(): SKIPPED!
divti3_test(): SKIPPED!
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!
```

(continues on next page)

(continued from previous page)

```
modsi3_test(): PASS!
modti3_test(): SKIPPED!
muldf3vfp_test(): SKIPPED!
muldi3_test(): PASS!
mulodi4_test(): PASS!
mulosi4_test(): PASS!
muloti4_test(): SKIPPED!
mulsf3vfp_test(): SKIPPED!
multf3_test(): SKIPPED!
multi3_test(): SKIPPED!
mulvdi3_test(): PASS!
mulvsi3_test(): PASS!
mulvti3_test(): SKIPPED!
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!
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

RESOURCES

6.1 Build steps

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

6.2 Book example code

The example code exlbt.tar.gz is available in:

http://jonathan2251.github.io/lbt/exlbt.tar.gz

6.3 Alternate formats

The book is also available in the following formats:

6.4 Presentation files

6.5 Search this website

- search