
Tutorial: Creating an LLVM Toolchain for the Cpu0 Architecture

Release 12.0.0

Chen Chung-Shu

Aug 12, 2021

CONTENTS

1	About	1
1.1	Authors	1
1.2	Acknowledgments	1
1.3	Revision history	1
1.4	Licensing	2
1.5	Outline of Chapters	2
2	Cpu0 ELF linker	5
2.1	ELF to Hex	6
2.2	Create Cpu0 backend under LLD	21
2.3	Summary	45
3	Optimization	47
3.1	LLVM IR optimization	47
3.2	Project	51
4	Library	53
4.1	Compiler-rt	53
4.2	Avr libc	54
4.3	Software Float Point Support	54
5	Build steps	59
6	Book example code	61
7	Alternate formats	63
8	Presentation files	65
9	Search this website	67

ABOUT

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

1.1 Authors

陳鍾樞

Chen Chung-Shu gamma_chen@yahoo.com.tw
<http://jonathan2251.github.io/web/index.html>

1.2 Acknowledgments

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

1.3 Revision history

Version 12.0.1, Not release yet.

Version 12.0.0, Release August 11, 2021.

Version 3.9.1, Released April 29, 2020 Enable tailcall test option in build-slinker.sh

Version 3.9.0, Released November 22, 2016 Porting to llvm 3.9.

Version 3.7.4, Released September 22, 2016 Split elf2hex-dlinker.cpp from elf2hex.cpp in exlbt/elf2hex.

Version 3.7.3, Released July 20, 2016 Refine code-block according sphinx lexers. Add search this book.

Version 3.7.2, Released June 29, 2016 Dynamic linker change display from ret \$t9 to jr \$t9. Move llvm-objdump -elf2hex to elf2hex. Upgrade sphinx to 1.4.4.

Version 3.7.1, Released November 7, 2015 Remove EM_CPU0_EL. Add IR blockaddress and indirectbr support. Add ch_9_3_detect_exception.cpp test. Change display “ret \$rx” to “jr \$rx” where \$rx is not \$lr. Add Phi node test.

Version 3.7.0, Released September 24, 2015 Porting to lld 3.7.

Version 3.6.2, Released May 4, 2015 Move some test from lbt to lbd. Remove warning in build Cpu0 code.

Version 3.6.1, Released March 22, 2015 Correct typing.

Version 3.6.0, Released March 8, 2015 Porting to lld 3.6.

1.4 Licensing

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

1.5 Outline of Chapters

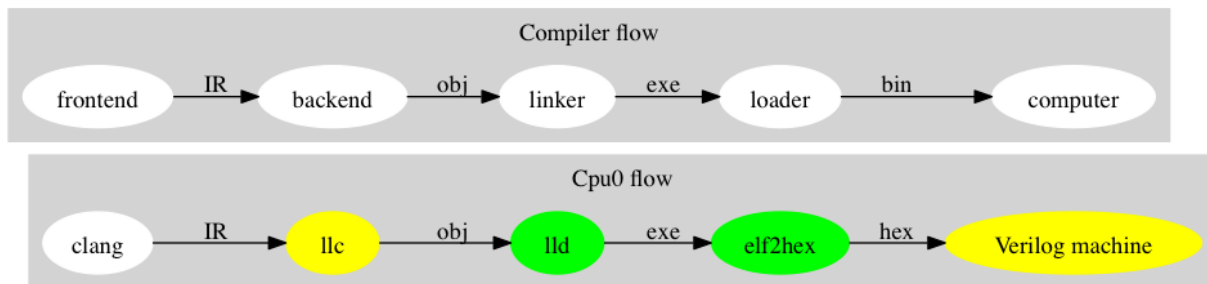


Fig. 1.1: Code generation and execution flow

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-objdump. Chapter 2.
2. Optimization. Chapter 3.
3. Porting C standard library from avr libc and software floating point library from LLVM compiler-rt.

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

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.

CPU0 ELF LINKER

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

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

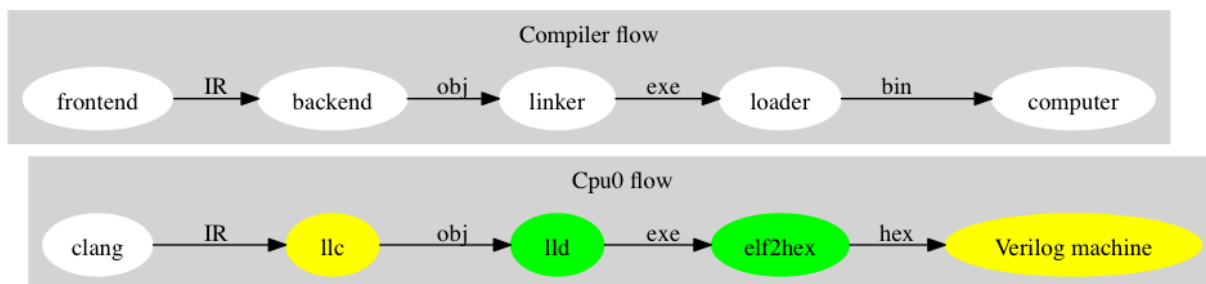


Fig. 2.1: Code generation and execution flow

As depicted in Fig. 2.1 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 transferred to Hex file which feeds to Verilog Cpu0 machine and run on your PC/Laptop.

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

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

2.1 ELF to Hex

As follows,

exlbt/elf2hex/CMakeLists.txt

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

add_llvm_tool(elf2hex
  elf2hex.cpp
)

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

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

¹ <http://lld.llvm.org/>

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

exlbt/elf2hex/elf2hex.h

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

#ifndef LLVM_TOOLS_ELF2HEX_ELF2HEX_H
#define LLVM_TOOLS_ELF2HEX_ELF2HEX_H

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

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

#define BOOT_SIZE 16

#define DLINK
// #define ELF2HEX_DEBUG

namespace llvm {
namespace elf2hex {

using namespace object;

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

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

(continues on next page)

(continued from previous page)

```

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

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

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

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

//using namespace llvm;

#endif

```

exlbt/elf2hex/elf2hex.cpp

```

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

```

(continues on next page)

(continued from previous page)

```

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

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

using namespace llvm;
using namespace llvm::object;

staticStringRef ToolName;
staticStringRef CurrInputFile;

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

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

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

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

std::string TripleName = "";

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

```

(continues on next page)

(continued from previous page)

```

    BaseAddr = Section.getAddress();

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

using namespace llvm::elf2hex;

Reader reader;

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

    return;
}

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

    if (BaseAddr < 0x100)
      continue;

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

   StringRef name;
    getName(Section, name);

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

    const Target *TheTarget = getTarget(Obj);

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

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

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

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

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

    Elf2Hex(o);
}

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

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

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

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

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

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

    // Register the target printer for --version.

```

(continues on next page)

(continued from previous page)

```

cl::AddExtraVersionPrinter(TargetRegistry::printRegisteredTargetsForVersion);

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

ToolName = argv[0];

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

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

return EXIT_SUCCESS;
}

```

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

`exlbt/llvm-objdump/llvm-objdump.cpp`

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

2.2 Create Cpu0 backend under LLD

2.2.1 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. 2.2](#) 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 build-ch13_1.sh 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

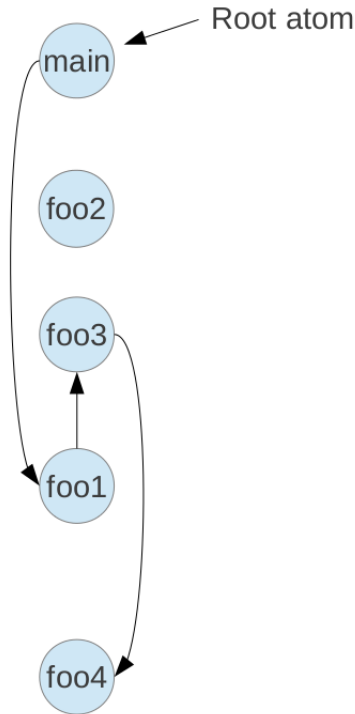


Fig. 2.2: Atom classified (from lld web)

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.

2.2.2 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

#ifndef 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 ffffffff
 * 0 message(s)
 * 0 message(s) with %
 * justif: "left      "
 * justif: "      right"
 * 3: 0003 zero padded
 * 3: 3    left justif.
 * 3:   3 right justif.
 * -3: -003 zero padded
 * -3: -3   left justif.
 * -3:  -3 right justif.
 */
#endif

```

exlbt/input/printf-stdarg-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*)OUT_MEM;
    *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;
            }
            while (*format == '0') {
                ++format;
            }

```

(continues on next page)

(continued from previous page)

```

    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;

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

```

(continues on next page)

(continued from previous page)

```

}

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
 * this line is testing an invalid % at the end of the format string.
 */

```

(continues on next page)

(continued from previous page)

```

* this should display (on 32bit int machine) :
*
* Hello world!
* printf test
* (null) is null pointer
* 5 = 5
* -2147483647 = - max int
* char a = 'a'
* hex ff = ff
* hex 00 = 00
* signed -3 = unsigned 4294967293 = hex ffffffff
* 0 message(s)
* 0 message(s) with %
* justif: "left      "
* justif: "      right"
* 3: 0003 zero padded
* 3: 3    left justif.
* 3:    3 right justif.
* -3: -003 zero padded
* -3: -3   left justif.
* -3:   -3 right justif.
*/

#endif

```

exlbt/input/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:");

    asm("lui $sp, 0x7");
    asm("addiu $sp, $sp, 0xfffc");
    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");
    asm("ret $lr");
}

```

(continues on next page)

(continued from previous page)

```
}
```

exlbt/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 {
entry:
```

(continues on next page)

(continued from previous page)

```

    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/functions.sh

```

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

    if [ $argNum == 0 ]; then
        echo "usage: 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 [ $arg1 != cpu032I ] && [ $arg1 != 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

    CPU=$arg1

```

(continues on next page)

(continued from previous page)

```

echo "CPU =" "${CPU}"

if [ "$arg2" != "" ] && [ $arg2 != le ] && [ $arg2 != be ]; then
    echo "2nd argument is be (big endian, default) or le (little endian)"
    exit 1
fi
if [ "$arg2" == "" ] || [ $arg2 == be ]; then
    endian=
else
    endian=el
fi
echo "endian =" "${endian}"

bash clean.sh
}

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

elf2hex() {
    ${TOOLDIR}/elf2hex -le=${le} 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;
}

```

exlbt/input/build-printf-stdarg-2.sh

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

source functions.sh

sh_name=build-printf-stdarg-2.sh
argNum=$(( $# ))
arg1=$1
arg2=$2

prologue;

${CLANG} -target mips-unknown-linux-gnu -c start.cpp -emit-llvm -o start.bc
${CLANG} -target mips-unknown-linux-gnu -c debug.cpp -emit-llvm -o debug.bc
${CLANG} -target mips-unknown-linux-gnu -c printf-stdarg-def.c -emit-llvm \
-o printf-stdarg-def.bc
${CLANG} -target mips-unknown-linux-gnu -c printf-stdarg-2.cpp -emit-llvm -o \
printf-stdarg-2.bc
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj start.bc -o start.cpu0.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj debug.bc -o debug.cpu0.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj printf-stdarg-def.bc -o printf-stdarg-def.cpu0.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj printf-stdarg-2.bc -o printf-stdarg-2.cpu0.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj lib_cpu0.ll -o lib_cpu0.o
${TOOLDIR}/lld -flavor gnu \
start.cpu0.o debug.cpu0.o printf-stdarg-def.cpu0.o printf-stdarg-2.cpu0.o \
lib_cpu0.o -o a.out

epilogue;
```

lbdex/verilog/Makefile

```
#TRACE=-D TRACE
all:
    iverilog ${TRACE} -o cpu0Is cpu0.v
    iverilog ${TRACE} -D CPU0II -o cpu0IIs cpu0.v

.PHONY: clean
clean:
    rm -rf cpu0.hex cpu0Is cpu0IIs
    rm -f *~ cpu0.config
```

The build-printf-stdarg-2.sh 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 build-printf-stdarg-2.sh 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 ffffffff
0 message(s)
0 message(s) with \%
justif: "left      "
justif: "      right"
3: 0003 zero padded
3: 3    left justif.
3:      3 right justif.
-3: -003 zero padded

```

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
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 ffffffff
0 message(s)

```

(continues on next page)

(continued from previous page)

```
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/build-printf-stdarg-2.sh

```
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 build-printf-stdarg-2.sh 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 build-printf-stdarg-2.sh 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");
```

(continues on next page)

(continued from previous page)

```

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

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

```

exlbt/input/ch_1ld_staticlink.h

```

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

// #define PRINT_TEST

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

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

extern int test_staticlink();

```

exlbt/input/ch_lld_staticlink.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;
}

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");
}
```

(continues on next page)

(continued from previous page)

```

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)
    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

```

(continues on next page)

(continued from previous page)

```
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;
}
```

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/build-slinker.sh

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

source functions.sh

sh_name=build-slinker.sh
argNum=$#
arg1=$1
arg2=$2
```

(continues on next page)

(continued from previous page)

```

prologue;

${CLANG} -target mips-unknown-linux-gnu -c start.cpp -emit-llvm -o \
start.bc
${CLANG} -target mips-unknown-linux-gnu -c debug.cpp -emit-llvm -o \
debug.bc
${CLANG} -target mips-unknown-linux-gnu -c printf-stdarg-def.c \
-emit-llvm -o printf-stdarg-def.bc
${CLANG} -target mips-unknown-linux-gnu -c printf-stdarg.c -emit-llvm \
-o printf-stdarg.bc
${CLANG} -target mips-unknown-linux-gnu -c \
${LBDEDIR}/input/ch4_1_addsuboverflow.cpp -emit-llvm -o ch4_1_addsuboverflow.bc
${CLANG} -target mips-unknown-linux-gnu -c \
${LBDEDIR}/input/ch8_1_br_jt.cpp -emit-llvm -o ch8_1_br_jt.bc
${CLANG} -O3 -target mips-unknown-linux-gnu -c \
${LBDEDIR}/input/ch8_2_phinode.cpp -emit-llvm -o ch8_2_phinode.bc
${CLANG} -target mips-unknown-linux-gnu -c \
${LBDEDIR}/input/ch8_1_blockaddr.cpp -emit-llvm -o ch8_1_blockaddr.bc
${CLANG} -target mips-unknown-linux-gnu -c \
${LBDEDIR}/input/ch8_2_longbranch.cpp -emit-llvm -o ch8_2_longbranch.bc
${CLANG} -O1 -target mips-unknown-linux-gnu -c \
${LBDEDIR}/input/ch9_2_tailcall.cpp -emit-llvm -o ch9_2_tailcall.bc
${CLANG} -target mips-unknown-linux-gnu -c \
${LBDEDIR}/input/ch9_3_detect_exception.cpp -emit-llvm -o \
ch9_3_detect_exception.bc
${CLANG} -I${LBDEDIR}/input/ -target mips-unknown-linux-gnu -c \
ch_slinker.cpp -emit-llvm -o ch_slinker.bc
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true start.bc -o start.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true debug.bc -o debug.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true printf-stdarg-def.bc -o printf-stdarg-def.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj printf-stdarg.bc -o printf-stdarg.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true -cpu0-enable-overflow=true ch4_1_addsuboverflow.bc -o \
ch4_1_addsuboverflow.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true ch8_1_br_jt.bc -o ch8_1_br_jt.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj ch8_2_phinode.bc -o ch8_2_phinode.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true ch8_1_blockaddr.bc -o ch8_1_blockaddr.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=pic \
-filetype=obj -has-lld=true -force-cpu0-long-branch ch8_2_longbranch.bc -o \
ch8_2_longbranch.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true -enable-cpu0-tail-calls ch9_2_tailcall.bc -o ch9_2_tailcall.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true ch9_3_detect_exception.bc -o ch9_3_detect_exception.o

```

(continues on next page)

(continued from previous page)

```

${TOOLDIR}/lld -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true ch_slinker.bc -o ch_slinker.o
${TOOLDIR}/lld -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true lib_cpu0.ll -o lib_cpu0.o
${TOOLDIR}/lld -flavor gnu start.o \
debug.o printf-stdarg-def.o printf-stdarg.o ch4_1_addsuboverflow.o \
ch8_1_br_jt.o ch8_2_phinode.o ch8_1_blockaddr.o ch8_2_longbranch.o \
ch9_2_tailcall.o ch9_3_detect_exception.o ch_slinker.o lib_cpu0.o -o a.out

epilogue;

```

```

1-160-136-173:input Jonathan$ pwd
/Users/Jonathan/Downloads/exlbt/input
114-37-148-111:input Jonathan$ bash build-slinker.sh cpu032I le
...
In file included from ch_slinker.cpp:23:
./ch_lld_staticlink.cpp:8:15: warning: conversion from string literal to
'char *' is deprecated
    [-Wdeprecated-writable-strings]
    char *ptr = "Hello world!";
                ^
1 warning generated.
114-37-148-111:input Jonathan$ cd ../../lbdex/verilog/
114-37-148-111:verilog Jonathan$ ./cpu0IIs
WARNING: ./cpu0.v:369: $readmemh(cpu0.hex): Not enough words in the file for
the requested range [0:524287].
taskInterrupt(001)
...
test_nolld(): PASS
taskInterrupt(011)
Overflow exception
taskInterrupt(011)
Overflow exception
test_overflow = 0, 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
date1 = 2012 10 12 1 2 3, PASS
date2 = 2012 10 12 1 2 3, PASS
time2 = 1 10 12, PASS
time3 = 1 10 12, PASS
date1 = 2013 1 26 12 21 10, PASS
date2 = 2013 1 26 12 21 10, PASS
test_template() = 15, PASS
test_alloc() = 31, PASS
exceptionOccur= 1, PASS
exceptionOccur= 0, PASS
inlineasm_global() = 4, PASS
20

```

(continues on next page)

(continued from previous page)

```

10
5
test_cpp_polymorphism() = 0, PASS
taskInterrupt(011)
Software interrupt
taskInterrupt(011)
Hardware interrupt 0
taskInterrupt(011)
Hardware interrupt 1
...

```

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.

2.2.3 Dynamic linker

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

```

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

```

Then reading this section in lld.html for it.

2.3 Summary

2.3.1 Create a new backend base on LLVM

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

and CPU design has been translated into a real work and see how it is running. Now, these school books knowledge is not limited on paper. We design it, program it, and run it on real world.

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

2.3.2 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 ago⁷⁸⁹¹⁰¹¹. Open Source grows up and becomes a giant software infrastructure with the forces of company¹²¹³, 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 Expection¹⁴. 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 here¹⁵¹⁶¹⁷. Appendix A of lld 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>

OPTIMIZATION

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

This chapter introduce llvm optimization.

3.1 LLVM IR optimization

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

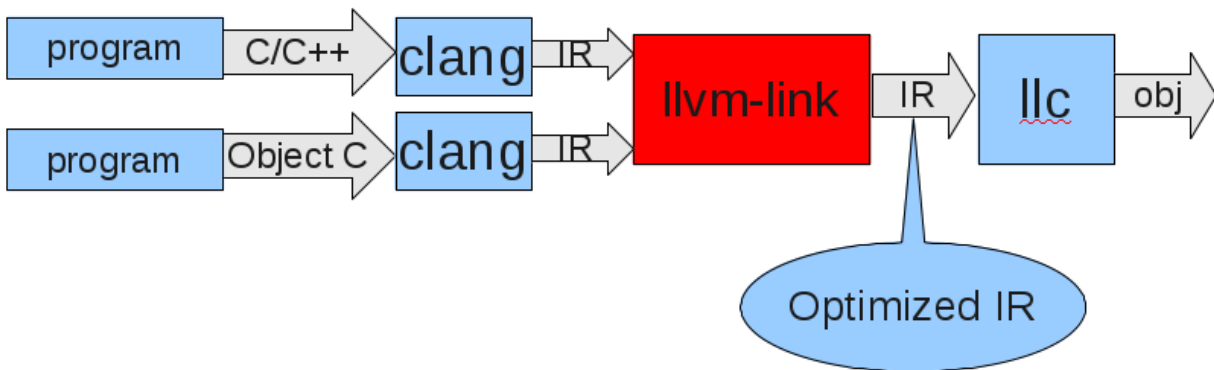


Fig. 3.1: llvm-link flow

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 optimization¹. Beyond the DAG local optimization mentioned in Chapter 2, there are global optimization based on inter-procedure analysis². The following steps and examples show this optimization solution in llvm.

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

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

exlbt/input/optimizen/1.cpp

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

exlbt/input/optimize/2.cpp

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

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

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

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

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

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

(continues on next page)

(continued from previous page)

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

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

```
JonathantekiiMac:input Jonathan$ ~/llvm/test/build/
bin/llc -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
    .cpstore     8
    addiu $2, $zero, 4
    st     $2, 24($sp)
    addiu $2, $sp, 24
    st     $2, 0($sp)
    ld     $t9, %call16(_Z6calleePKi)($gp)
    jalr   $t9
    ld     $gp, 8($sp)
    ld     $lr, 28($sp)          # 4-byte Folded Reload
    addiu $sp, $sp, 32
    ret    $lr
    .set    macro
    .set    reorder
    .end    _Z6callerv

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

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

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

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

```

(continues on next page)

(continued from previous page)

```

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

```

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

3.2 Project

3.2.1 LLVM-VPO

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

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

³ <http://cs.boisestate.edu/~uh/>

LIBRARY

- *Compiler-rt*
- *Avr libc*
- *Software Float Point Support*

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 the software floating point library implementation, so we choose it as the implementation.

Since compiler-rt uses unix/linux rootfs structure, we fill the gap by porting avr libc.

Both the compiler-rt and avr libc porting is under going, it's not finished. The flow as follows,

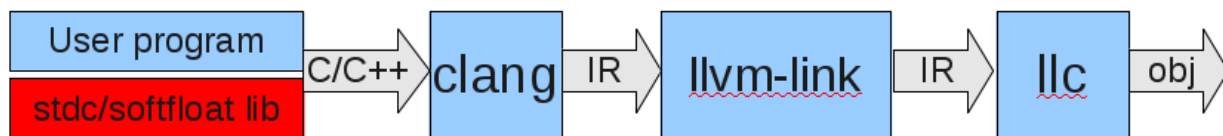


Fig. 4.1: libc/softfloat library flow

The llvm-link which introduced at last chapter can be hired for optimization.

4.1 Compiler-rt

Directory libex/libsoftfloat/compiler-rt include the floating point library support for Cpu0 backend. The compiler-rt¹ version we use is llvm 3.5 release.

¹ <http://compiler-rt.llvm.org/>

4.2 Avr libc

Directory `libex/libc/avr-libc-1.8.1` include the libc porting.

AVR Libc is a Free Software project whose goal is to provide a high quality C library for use with GCC on Atmel AVR microcontrollers. AVR Libc is licensed under a single unified license. This so-called modified Berkeley license is intended to be compatible with most Free Software licenses like the GPL, yet impose as little restrictions for the use of the library in closed-source commercial applications as possible².

The source code can be download from here³. Document are here^{4,5}.

4.3 Software Float Point Support

`exlbt/input/ch_float_necessary.cpp`

```
//#include "debug.h"

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

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

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

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

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

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

(continues on next page)

² <http://www.nongnu.org/avr-libc/>

³ <http://download.savannah.gnu.org/releases/avr-libc/>

⁴ <http://www.atmel.com/webdoc/AVRLibcReferenceManual/index.html>

⁵ <http://courses.cs.washington.edu/courses/csep567/04sp/pdfs/avr-libc-user-manual.pdf>

(continued from previous page)

```

int main() {
    int a;

    // call __ashldi3
    a = (int)test_shift_left<long long>(0x12, 4); // 0x120 = 288
    printf("(int)test_shift_left<long long>(0x12, 4) = %d\n", a);

    // call __ashrdi3
    a = (int)test_shift_right<long long>(0x0016666600000000a, 48); // 0x16 = 22
    printf("(int)test_shift_right<long long>(0x0016666600000000a, 48) = %d\n", a);

    // call __lshrdi3
    a = (int)test_shift_right<unsigned long long>(0x0016666600000000a, 48); // 0x16 = 22
    printf("(int)test_shift_right<unsigned long long>(0x0016666600000000a, 48) = %d\n", a);

    // call __addsf3, __fixsfsi
    a = (int)test_add<float, float, float>(-2.2, 3.3); // (int)1.1 = 1
    printf("(int)test_add<float, float, float>(-2.2, 3.3) = %d\n", a);

    // call __mulsf3, __fixsfsi
    a = (int)test_mul<float, float, float>(-2.2, 3.3); // (int)-7.26 = -7
    printf("(int)test_mul<float, float, float>(-2.2, 3.3) = %d\n", a);

    // call __divsf3, __fixsfsi
    a = (int)test_div<float, float, float>(-1.8, 0.5); // (int)-3.6 = -3
    printf("(int)test_div<float, float, float>(-1.8, 0.5) = %d\n", a);

    // call __extendsfdf2, __adddf3, __fixdfsi
    a = (int)test_add<double, double, float>(-2.2, 3.3); // (int)1.1 = 1
    printf("(int)test_add<double, double, float>(-2.2, 3.3) = %d\n", a);

    // call __extendsfdf2, __muldf3, __fixdfsi
    a = (int)test_mul<double, float, double>(-2.2, 3.3); // (int)-7.26 = -7
    printf("(int)test_mul<double, float, double>(-2.2, 3.3) = %d\n", a);

    // call __extendsfdf2, __muldf3, __truncdfsf2, __fixdfsi
    // ! __truncdfsf2 in truncdfsf2.c is not work for Cpu0
    a = (int)test_mul<float, float, double>(-2.2, 3.3); // (int)-7.26 = -7
    printf("(int)test_mul<float, float, double>(-2.2, 3.3) = %d\n", a);

    // call __divdf3, __fixdfsi
    a = (int)test_div<double, double, double>(-1.8, 0.5); // (int)-3.6 = -3
    printf("(int)test_div<double, double, double>(-1.8, 0.5) = %d\n", a);

    return 0;
}

```

exlbt/input/build-float-necessary.sh

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

INCFLAG="-I../libsoftfloat/compiler-rt/builtins"

source functions.sh

sh_name=build-float.sh
argNum=#
arg1=$1
arg2=$2

prologue;

libsf=../libsoftfloat/compiler-rt
pushd ${libsf}
bash build.sh
popd
olibsf=${libsf}/obj

${CLANG} -target mips-unknown-linux-gnu -c start.cpp -emit-llvm -o start.bc
${CLANG} -target mips-unknown-linux-gnu -c debug.cpp -emit-llvm -o debug.bc
${CLANG} -target mips-unknown-linux-gnu -c printf-stdarg-def.c -emit-llvm \
-o printf-stdarg-def.bc
${CLANG} -target mips-unknown-linux-gnu -c printf-stdarg.c -emit-llvm \
-o printf-stdarg.bc
${CLANG} $INCFLAG -c ch_float_necessary.cpp -emit-llvm -o ch_float_necessary.bc
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true start.bc -o start.cpu0.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true debug.bc -o debug.cpu0.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true printf-stdarg-def.bc -o printf-stdarg-def.cpu0.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true printf-stdarg.bc -o printf-stdarg.cpu0.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true ch_float_necessary.bc -o ch_float_necessary.cpu0.o
${TOOLDIR}/llc -march=cpu0${endian} -mcpu=${CPU} -relocation-model=static \
-filetype=obj -has-lld=true lib_cpu0.ll -o lib_cpu0.o
${TOOLDIR}/lld -flavor gnu -o a.out \
    start.cpu0.o debug.cpu0.o printf-stdarg-def.cpu0.o printf-stdarg.cpu0.o \
    ch_float_necessary.cpu0.o lib_cpu0.o ${olibsf}/libFloat.o
# ${olibsf}/fixsfsi.o ${olibsf}/fixsfdi.o ${olibsf}/fixdfsi.o \
# ${olibsf}/addsf3.o ${olibsf}/mulsf3.o ${olibsf}/divsf3.o \
# ${olibsf}/adddf3.o ${olibsf}/muldf3.o ${olibsf}/divdf3.o \
# ${olibsf}/ashrdi3.o ${olibsf}/ashldi3.o ${olibsf}/lshrdi3.o \
# ${olibsf}/extendsfdf2.o ${olibsf}/truncdfsf2.o

epilogue;
```

Run as follows,

```

JonathantekiiMac:input Jonathan$ bash build-float-necessary.sh cpu032II be
...
endian = BigEndian
0 /* 0: big endian, 1: little endian */

JonathantekiiMac:input Jonathan$ iverilog -o cpu0IIs cpu0IIs.v
JonathantekiiMac:input Jonathan$ ./cpu0IIs
114-43-184-210:verilog Jonathan$ ./cpu0IIs
ARNING: cpu0.v:487: $readmemh(cpu0.hex): Not enough words in the file for the requested_
↪range [0:524287].
taskInterrupt(001)
(int)test_shift_left<long long>(0x12, 4) = 288
(int)test_shift_right<long long>(0x001666660000000a, 48) = 22
(int)test_shift_right<unsigned long long>(0x001666660000000a, 48) = 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_mul<float, float, double>(-2.2, 3.3) = -7
(int)test_mul<float, float, double>(-2.2, 3.3) = 0
(int)test_div<double, double, double>(-1.8, 0.5) = -3
total cpu cycles = 182585
RET to PC < 0, finished!

```

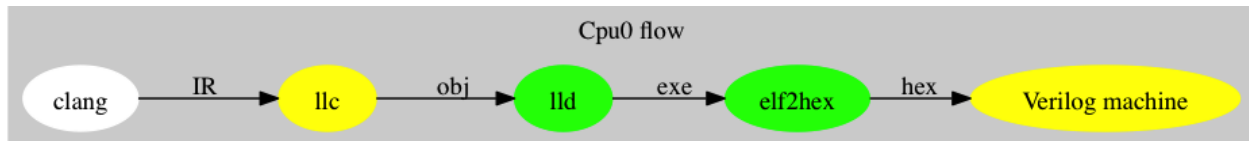


Fig. 4.2: This book's flow

BUILD STEPS

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

BOOK EXAMPLE CODE

The example code `exlbt.tar.gz` is available in:
<http://jonathan2251.github.io/lbt/exlbt.tar.gz>

ALTERNATE FORMATS

The book is also available in the following formats:

PRESENTATION FILES

SEARCH THIS WEBSITE

- search