中国科学院软件研究所 Institute of Software Chinese Academy of Sciences | 智能软件研究中心 Intelligent Software Research Center

写一个 LLVM Pass

廖春玉

2019/12/18

中国科学院软件研究所 Institute of Software Chinese Academy of Sciences
智能软件研究中心 Intelligent Software Research Center

CONTENT

目录

- 构建LLVM
- LLVM Pass简介
- 写一个MachineFunctionPass, RISCVDumpInsts
- 测试RISCVDumpInsts Pass

CONTENT

目录

构建LLVM

LLVM Pass简介

写一个MachineFunctionPass, RISCVDumpInsts

测试RISCVDumpInsts Pass

构建LLVM

- 构建环境：Ubuntu-18.04
- 源码下载方式：git clone https://github.com/llvm/llvm-project.git
- 构建命令：

① cd llvm-project
② mkdir build && cd build
③ cmake -DLLVM_TARGETS_TO_BUILD="X86;RISCV" -DLLVM_ENABLE_PROJECTS=clang -G "Unix Makefiles" ../llvm
④ make

CONTENT

目录

构建LLVM

LLVM Pass简介

写一个MachineFunctionPass, RISCVDumpInsts

测试RISCVDumpInsts Pass

Pass定义

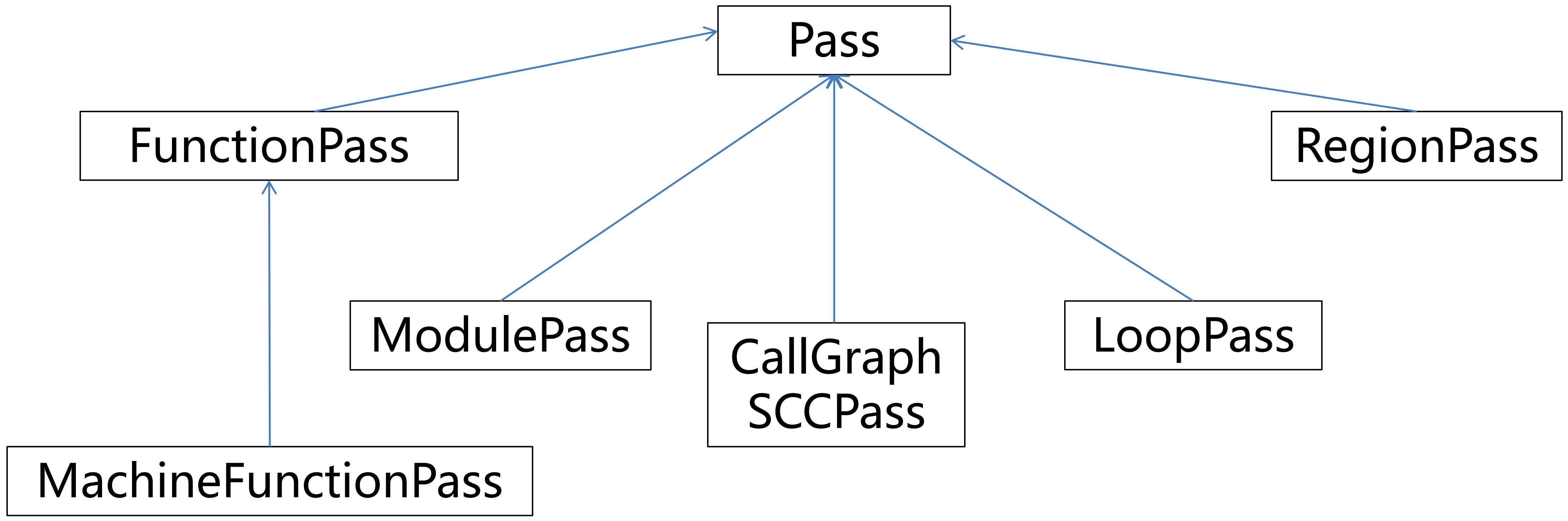
- The LLVM Pass Framework is an important part of the LLVM system, because LLVM passes are where most of the interesting parts of the compiler exist. Passes perform the transformations and optimizations that make up the compiler, they build the analysis results that are used by these transformations, and they are, above all, a structuring technique for compiler code.

- LLVM Pass是LLVM的重要组成部分
- LLVM转换和优化工作都是由Pass实现的

Pass分类

- 按功能划分
  - Analysis passes
    - 用于信息计算，其它Pass可以使用和调试相关信息
  - Transform passes
    - 可以使用Analysis passes
    - 以某种方式转换程序，通常是要对程序做出相应的改变
  - Utility passes
    - 提供了一些实用功能
    - 例如-view-cfg可以查看函数的控制流图

Pass类及其子类


Pass
FunctionPass
RegionPass
ModulePass
CallGraph SCCPass
LoopPass
MachineFunctionPass

Pass接口函数

- virtual bool doInitialization(Module &) { return false; }
  - 通过override该虚函数，可以在 Pass 运行之前做一些必要的初始化
- virtual bool doFinalization(Module &) { return false; }
  - 通过override该虚函数，可以在 Pass 运行之后做一些必要的清除操作
- virtual void getAnalysisUsage(AnalysisUsage &) const;
  - 如果需要其它 Pass 产生的分析信息来完成工作，那么需要override该虚函数
  - getAnalysis<AnalysisType>()

ModulePass

- ModulePass 以整个模块为单元运行
  - 可以删除函数，或者以不可预测的顺序访问函数体

- 主要功能接口
  - bool runOnModule(Module &M) = 0;

FunctionPass

- FunctionPass 遍历每个函数
  - 针对每个函数独立运行
    - 不能查询或修改其它函数
    - 不能增加或删除当前 Module 的函数及全局变量

- 主要功能接口
  - bool doInitialization (Module &M);
  - bool runOnFunction(Function &F);
  - bool doFinalization(Module &M);

CallGraphSCCPass

- CallGraphSCCPass 以调用图上自底向上的顺序遍历程序
  - 使用限制
    - 不能查询或修改除当前 SCC 以及 SCC 的直接调用者和被调用者外的函数
    - 需要保留当前的 CallGraph 结构
    - 不能添加或者删除当前 Module 的 SCC
    - 可以添加和删除当前 Module 的全局变量

- 主要功能接口
  - bool doInitialization(CallGraph &CG)
  - bool runOnSCC(CallGraphSCC &SCC)
  - bool doFinalization(CallGraph &CG)

LoopPass

- LoopPass 遍历每个 Loop
  - 针对每个 Loop 独立运行
    - 以 loop 迭代顺序处理，最外层 loop 最后处理
    - 可以通过 LPPassManager 接口更新 loop 迭代

- 主要功能接口
  - bool doInitialization(Loop *, LPPassManager &LPM);
  - bool runOnLoop(Loop *, LPPassManager &LPM);
  - bool doFinalization();

RegionPass

- RegionPass 遍历函数中每个单入单出 region
  - 针对每个 Region 独立运行
  - 以迭代顺序处理每个 Region, 最外层 Region 最后处理
  - 可以通过 RGPassManager 接口更新 Region 树

- 主要功能接口
  - bool doInitialization(Region *, RGPassManager &RGM);
  - bool runOnRegion(Region *, RGPassManager &RGM);
  - bool doFinalization()

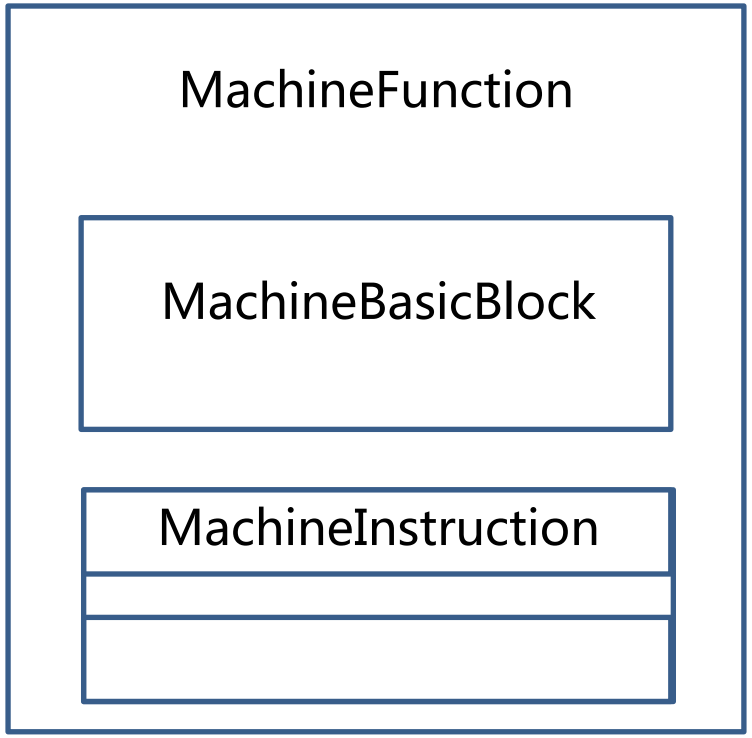
MachineFunctionPass

- MachineFunctionPass是LLVM代码生成器的一部分
  - FunctionPass的限制均适用
  - 属于MIR

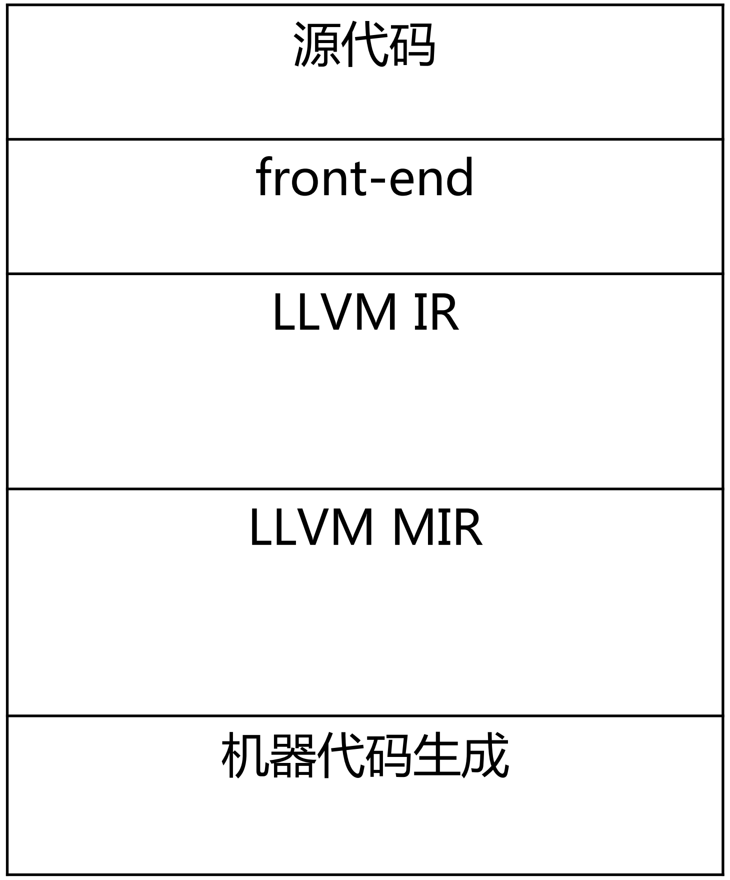
- 主要功能接口
  - virtual bool runOnMachineFunction(MachineFunction &MF) = 0;

MIR简介

- Machine IR的简称
- 可读的序列化格式，用于表示LLVM特定机器的中间表示
  - class MachineFunction (MF)
  - class MachineBasicBlock (MBB)
  - class MachineInstruction (MI)



MIR简介



机器无关的中间表示; FunctionPass等

机器相关的中间表示; MachineFunctionPass

CONTENT

目录

构建LLVM

LLVM Pass简介

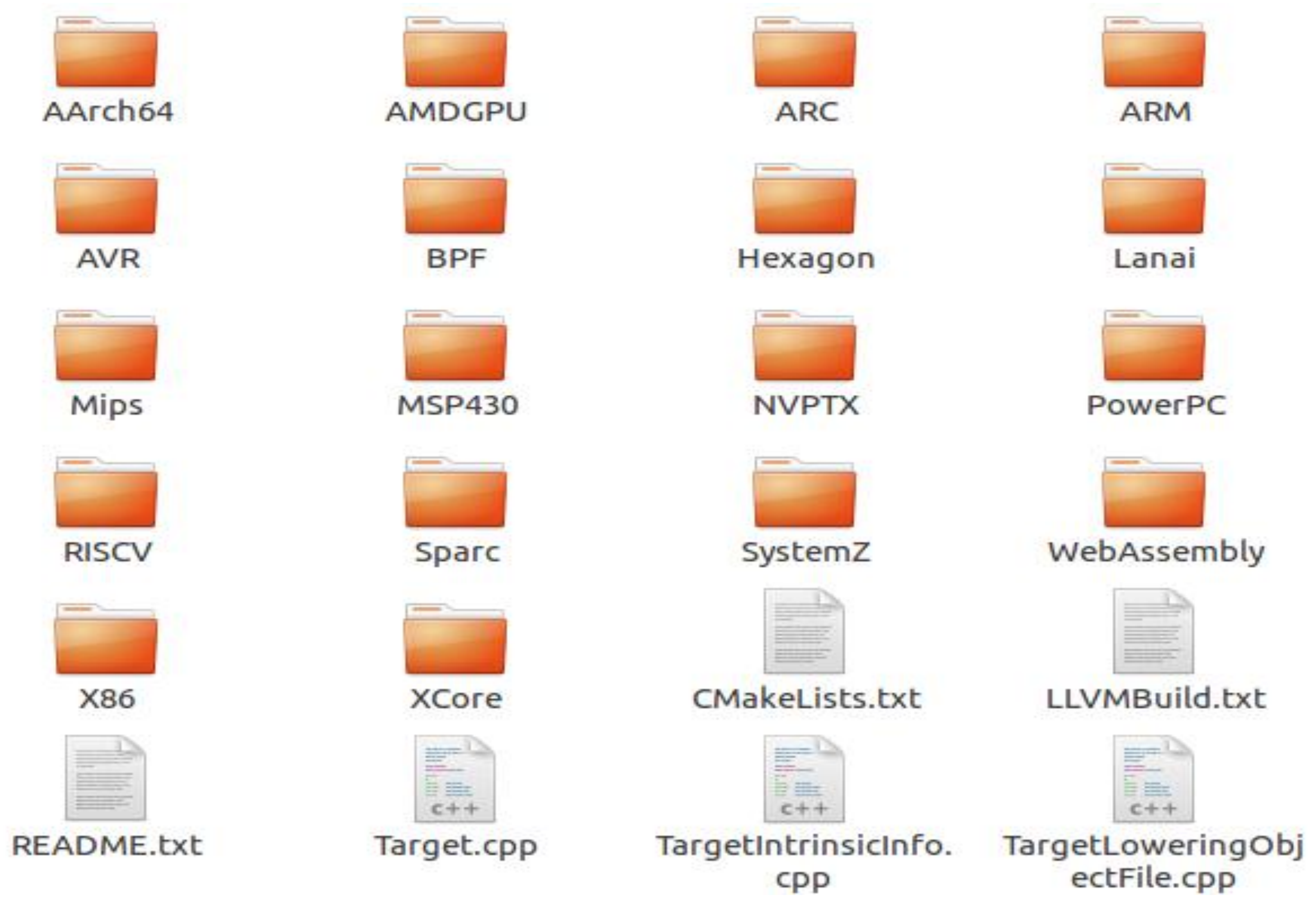
写一个MachineFunctionPass, RISCVDumpInsts

测试RISCVDumpInsts Pass

开始写RISCVDumpInsts Pass

- 该Pass预期完成的工作
  - 代码生成阶段实现MachineInstr的指令打印

LLVM所支持的Target

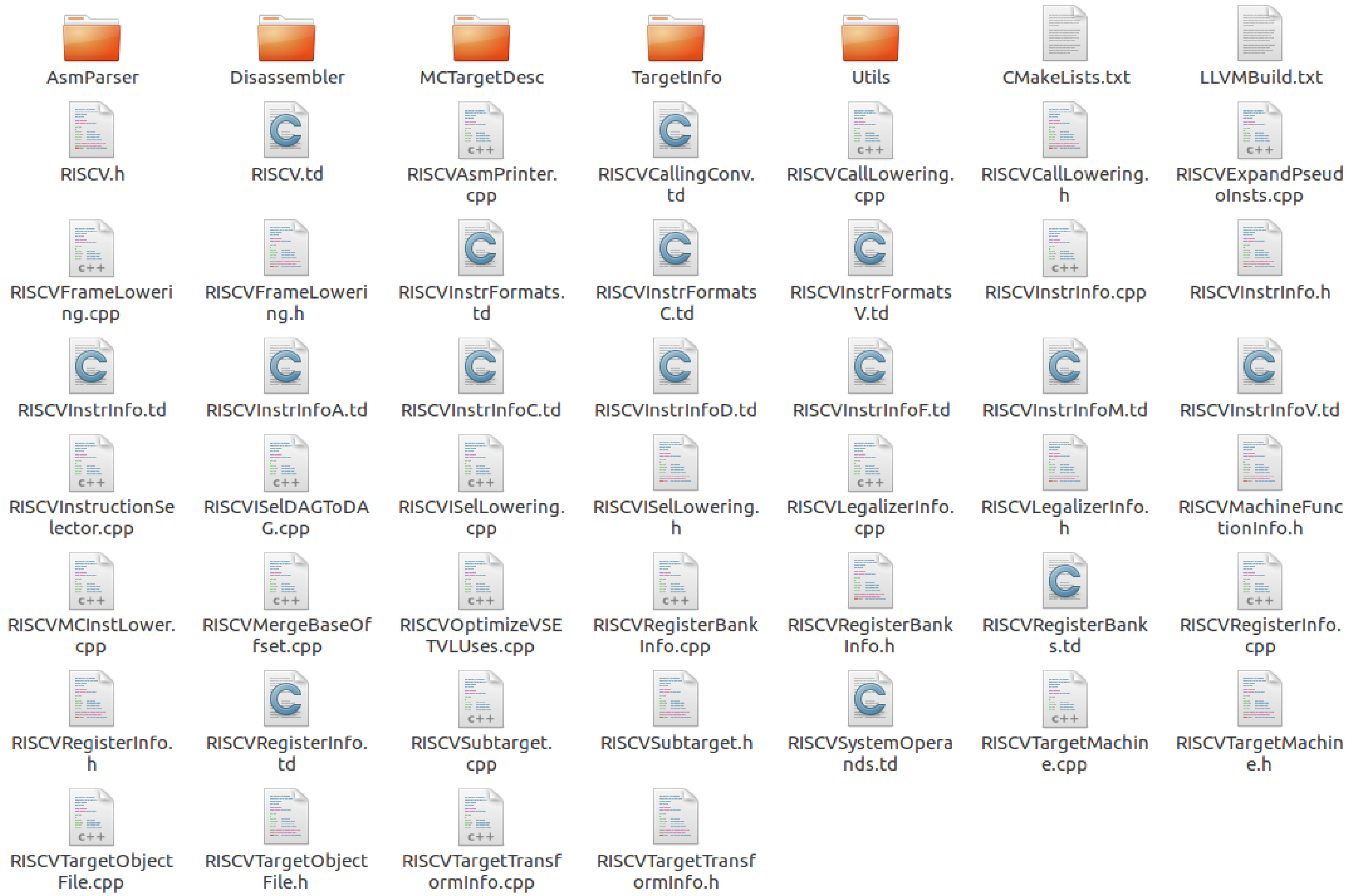


AArch64 AMDGPU ARC ARM
AVR BPF Hexagon Lanai
Mips MSP430 NVPTX PowerPC
RISCV Sparc SystemZ WebAssembly
X86 XCore CMakeLists.txt LLVMBuild.txt
README.txt Target.cpp TargetIntrinsicInfo.cpp TargetLoweringObjectFile.cpp

ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences

智能软件研究中心 Intelligent Software Research Center

Target - RISCV



AsmParser	Disassembler	MCTargetDesc	TargetInfo	Utils	CMakeLists.txt	LLVMBuild.txt
RISCV.h	RISCV.td	RISCVAsmPrinter.cpp	RISCVCallingConv.td	RISCVCallLowering.cpp	RISCVCallLowering.h	RISCVExpandPseudoInsts.cpp
RISCVFrameLowering.cpp	RISCVFrameLowering.h	RISCVInstrFormats.td	RISCVInstrFormatsC.td	RISCVInstrFormatsV.td	RISCVInstrInfo.cpp	RISCVInstrInfo.h
RISCVInstrInfo.td	RISCVInstrInfoA.td	RISCVInstrInfoC.td	RISCVInstrInfoD.td	RISCVInstrInfoF.td	RISCVInstrInfoM.td	RISCVInstrInfoV.td
RISCVInstructionSelector.cpp	RISCVISelDAGToDAG.cpp	RISCVISelLowering.cpp	RISCVISelLowering.h	RISCVLegalizerInfo.cpp	RISCVLegalizerInfo.h	RISCVMachineFunctionInfo.h
RISCVMCInstLower.cpp	RISCVMergeBaseOffset.cpp	RISCVOptimizeVSETVLUses.cpp	RISCVRegisterBankInfo.cpp	RISCVRegisterBankInfo.h	RISCVRegisterBanks.td	RISCVRegisterInfo.cpp
RISCVRegisterInfo.h	RISCVRegisterInfo.td	RISCVSubtarget.cpp	RISCVSubtarget.h	RISCVSystemOperands.td	RISCVTargetMachine.cpp	RISCVTargetMachine.h
RISCVTargetObjectFile.cpp	RISCVTargetObjectFile.h	RISCVTargetTransformInfo.cpp	RISCVTargetTransformInfo.h			

RISCVDumpInsts Pass

```
#include "RISCV.h"
#include "RISCVInstrInfo.h"

using namespace llvm;

#define DEBUG_TYPE "dumpinst"

#define RISCV_DUMP_INSTRUCTION_NAME "RISCV Dump Instruction pass"

namespace {

class RISCVDumpInsts : public MachineFunctionPass {
public:
  static char ID;

  RISCVDumpInsts() : MachineFunctionPass(ID) {
    initializeRISCVDumpInstsPass(*PassRegistry::getPassRegistry());
  }

  bool runOnMachineFunction(MachineFunction &MF) override;

};

char RISCVDumpInsts::ID = 0;
```

```
bool RISCVDumpInsts::runOnMachineFunction(MachineFunction &MF) {
  bool Modified = false;
  for (auto &MBB : MF) {
    for (auto &MI : MBB) {
      MI.dump();
    }
  }
  return Modified;
}

} // end of anonymous namespace

INITIALIZE_PASS(RISCVDumpInsts, "riscv-dump-insts",
                RISCV_DUMP_INSTRUCTION_NAME, false, false)
namespace llvm {

FunctionPass *createRISCVDumpInstsPass() { return new RISCVDumpInsts(); }

} // end of namespace llvm
```

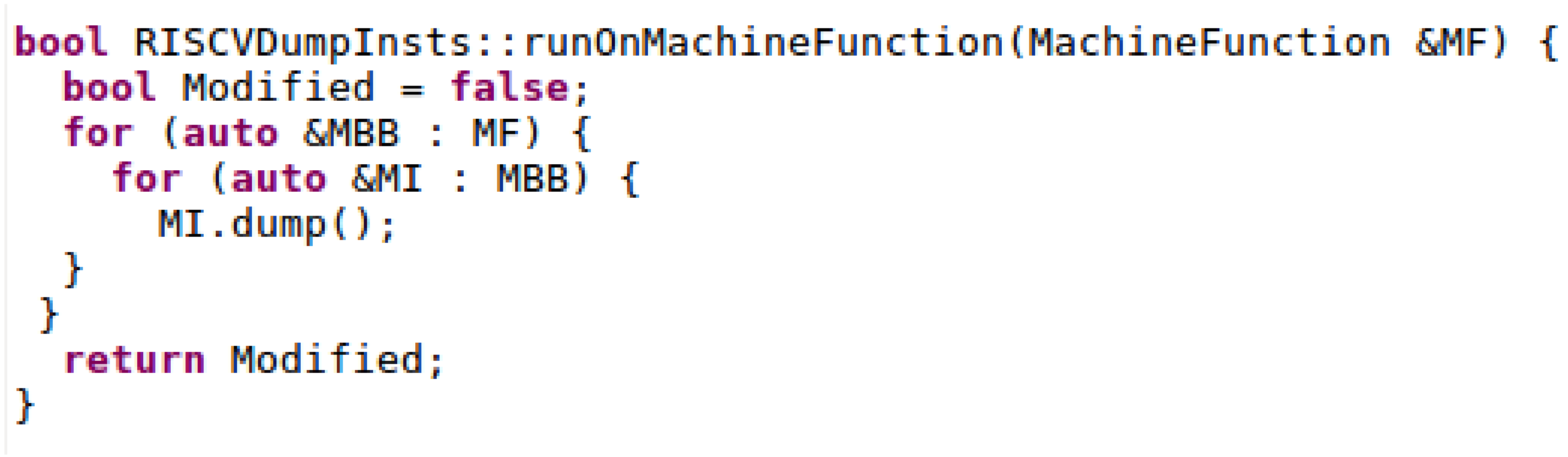
RISCVDumpInsts Pass

- 定义RISCVDumpInsts类

```
class RISCVDumpInsts : public MachineFunctionPass {
public:
  static char ID;

  RISCVDumpInsts() : MachineFunctionPass(ID) {
    initializeRISCVDumpInstsPass(*PassRegistry::getPassRegistry());
  }

  bool runOnMachineFunction(MachineFunction &MF) override;

};
```

RISCVDumpInsts Pass

- 遍历打印每条机器指令



```
bool RISCVDumpInsts::runOnMachineFunction(MachineFunction &MF) {
  bool Modified = false;
  for (auto &MBB : MF) {
    for (auto &MI : MBB) {
      MI.dump();
    }
  }
  return Modified;
}
```

RISCVDumpInsts Pass

- 定义Pass debug的特定标识
  - #define DEBUG_TYPE "dumpinst"

- 定义Pass的名字
  - #define RISCV_DUMP_INSTRUCTION_NAME "RISCV Dump Instruction pass"

RISCVDumpInsts Pass

- 注册RISCVDumpInsts Pass
  - INITIALIZE_PASS(RISCVDumpInsts, "riscv-dump-insts", RISCV_DUMP_INSTRUCTION_NAME, false, false)

- 同时注册所依赖 Pass
  - INITIALIZE_PASS_BEGIN(X86ExecutionDomainFix, "x86-execution-domain-fix", "X86 Execution Domain Fix", false, false)
  - INITIALIZE_PASS_DEPENDENCY(ReachingDefAnalysis)
  - INITIALIZE_PASS_END(X86ExecutionDomainFix, "x86-execution-domain-fix", "X86 Execution Domain Fix", false, false)

CMakeLists.txt

- 包含RISCV中需要编译的代码
- 将RISCVDumpInsts.cpp加入，使被编译

```
diff --git a/llvm/lib/Target/RISCV/CMakeLists.txt b/llvm/lib/Target/RISCV/CMakeLists.txt
index 31a82be1981..25bc4ae1959 100644
--- a/llvm/lib/Target/RISCV/CMakeLists.txt
+++ b/llvm/lib/Target/RISCV/CMakeLists.txt
@@ -20,6 +20,7 @@ add_llvm_target(RISCVCodeGen
   RISCVAsmPrinter.cpp
   RISCVCallLowering.cpp
+ RISCVDumpInsts.cpp
   RISCVExpandPseudoInsts.cpp
   RISCVFrameLowering.cpp
   RISCVInstrInfo.cpp
   RISCVInstructionSelector.cpp
```

RISCVTargetMachine.cpp

- 用于调度所有MachineFunctionPass
- RISCVDumpInstsPass被放到RISCVExpandPseudoPass之前

```
diff --git a/llvm/lib/Target/RISCV/RISCVTargetMachine.cpp b/llvm/lib/Target/RISCV/RISCVTargetMachine.cpp
index 3be546886d1..9db0e5dd7bd 100644
--- a/llvm/lib/Target/RISCV/RISCVTargetMachine.cpp
+++ b/llvm/lib/Target/RISCV/RISCVTargetMachine.cpp
@@ -137,6 +137,7 @@ void RISCVPassConfig::addPreEmitPass2() {
   // Schedule the expansion of AMOs at the last possible moment, avoiding the
   // possibility for other passes to break the requirements for forward
   // progress in the LR/SC block.
+  addPass(createRISCVDumpInstsPass());
   addPass(createRISCVExpandPseudoPass());
 }
```

RISCV.h

- 对新添加Pass进行函数声明

```
diff --git a/llvm/lib/Target/RISCV/RISCV.h b/llvm/lib/Target/RISCV/RISCV.h
index f23f742a478..228bc9555c5 100644
--- a/llvm/lib/Target/RISCV/RISCV.h
+++ b/llvm/lib/Target/RISCV/RISCV.h
@@ -43,6 +43,9 @@ void initializeRISCVMergeBaseOffsetOptPass(PassRegistry &);
 FunctionPass *createRISCVExpandPseudoPass();
 void initializeRISCVExpandPseudoPass(PassRegistry &);

+FunctionPass *createRISCVDumpInstsPass();
+void initializeRISCVDumpInstsPass(PassRegistry &);
+
 InstructionSelector *createRISCVInstructionSelector(const RISCVTargetMachine &,
                                                     RISCVSubtarget &,
                                                     RISCVRegisterBankInfo &);
```

CONTENT

目录

构建LLVM

LLVM Pass简介

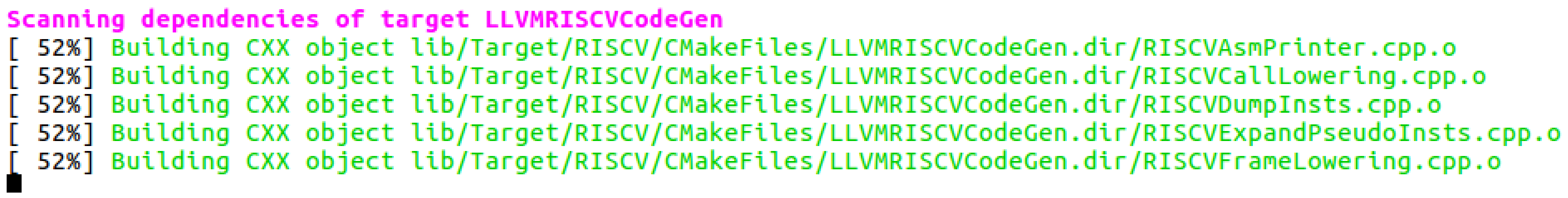
写一个MachineFunctionPass, RISCVDumpInsts

测试RISCVDumpInsts Pass

ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences | 智能软件研究中心 Intelligent Software Research Center

Pass测试

- 编译
  - 回到build目录：make



```
Scanning dependencies of target LLVMRISCVCodeGen
[ 52%] Building CXX object lib/Target/RISCV/CMakeFiles/LLVMRISCVCodeGen.dir/RISCVAsmPrinter.cpp.o
[ 52%] Building CXX object lib/Target/RISCV/CMakeFiles/LLVMRISCVCodeGen.dir/RISCVCallLowering.cpp.o
[ 52%] Building CXX object lib/Target/RISCV/CMakeFiles/LLVMRISCVCodeGen.dir/RISCVDumpInsts.cpp.o
[ 52%] Building CXX object lib/Target/RISCV/CMakeFiles/LLVMRISCVCodeGen.dir/RISCVExpandPseudoInsts.cpp.o
[ 52%] Building CXX object lib/Target/RISCV/CMakeFiles/LLVMRISCVCodeGen.dir/RISCVFrameLowering.cpp.o
█
```

  - 编译完成没有报错即可

llc命令

- 作用
  - 将LLVM源输入编译成特定体系结构的汇编语言

- 输入
  - LLVM assembly language format (.ll)
  - the LLVM bitcode format (.bc)

- 输出
  - 特定体系结构的汇编，-march指定输出体系结构

Pass测试

- 简单add.c程序实现 c = a + b

```
# cat add.c
...
   int a = 2;
   int b = 3;
   int c = a + b;
...
```

- 生成add.ll中间文件
  - clang -S -emit-llvm add.c

Pass测试

- 简单Dump所有Pass
  - llc add.ll --march=riscv32 --print-after-all -o asm >& log

- Dump特定Pass
  - llc add.ll --march=riscv32 -debug-only=dumpinst -o asm >& log

测试输出

```
# *** IR Dump After RISCV Dump Instruction pass ***:
......
bb.0 (%ir-block.0):
......
  renamable $x10 = ADDI $x0, 2
  SW killed renamable $x10, $x8, -16 :: (store 4 into %ir.2)
  renamable $x10 = ADDI $x0, 3
  SW killed renamable $x10, $x8, -20 :: (store 4 into %ir.3)
  renamable $x10 = LW $x8, -16 :: (dereferenceable load 4 from %ir.2)
  renamable $x11 = LW $x8, -20 :: (dereferenceable load 4 from %ir.3)
  renamable $x10 = nsw ADD killed renamable $x10, killed renamable $x11
......

# End machine code for function main
```

引用

- https://llvm.org/docs/GettingStarted.html
- https://llvm.org/docs/Passes.html
- http://llvm.org/docs/WritingAnLLVMPass.html
- http://llvm.org/docs/MIRLangRef.html
- https://llvm.org/docs/WritingAnLLVMBackend.html
- http://llvm.org/docs/CodeGenerator.html
- http://llvm.org/docs/MIRLangRef.html

谢谢

欢迎交流合作

2019/12/18