LLVM学习

软件所智能软件中心PLCT实验室 陈影 实习生

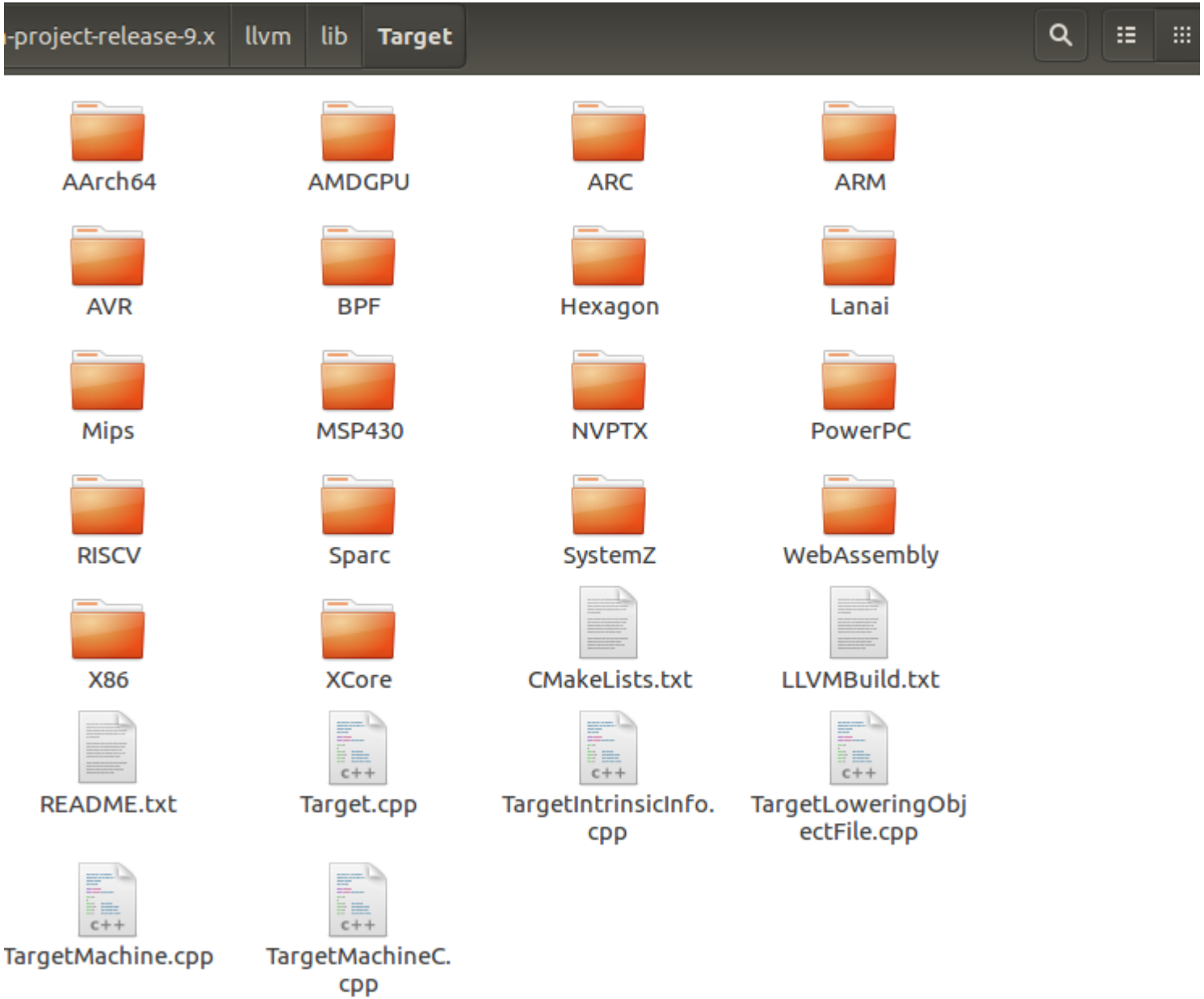
2020/2/12

目 录

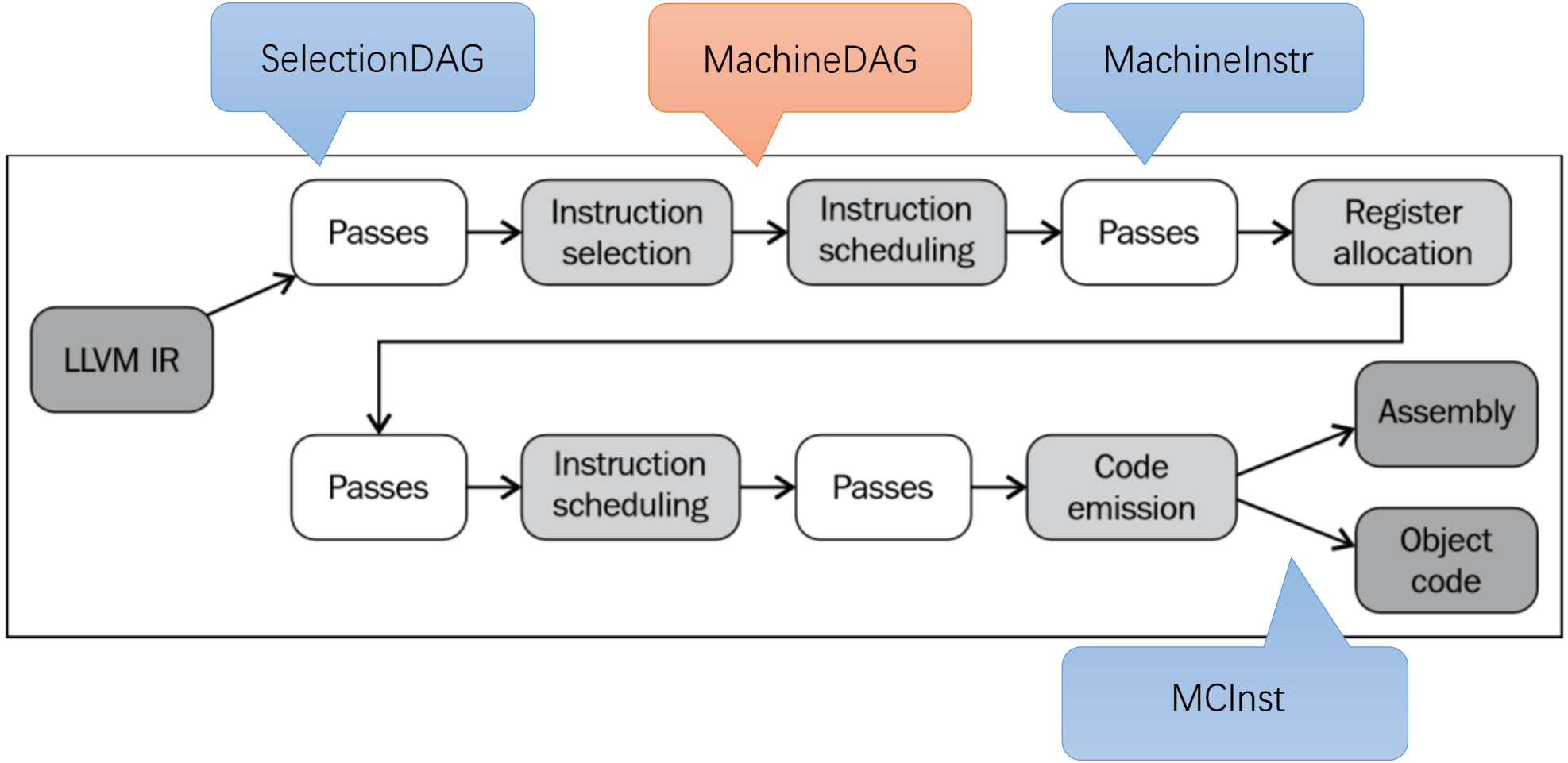
01 编译构建LLVM


```
$ git clone https://github.com/llvm/llvm-project.git
$ cd llvm-project
$ mkdir build && cd build
$ cmake -DLLVM_TARGETS_TO_BUILD="X86;RISCV" DLLVM_ENABLE_PROJECTS=clang  -G "Unix Makefiles" ../llvm
$ make
```

02 LLVM后端简介


-project-release-9.x
llvm
lib
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
TargetMachine.cpp
TargetMachineC.cpp

02 LLVM后端简介



图源：《Getting Started with LLVM Core Libraries》

02 LLVM后端简介

利用llc工具生成汇编码：

```
$ llc test.bc -o test.s
$ llc test.bc -filetype=obj -o test.o //生成目标代码
```

02 LLVM后端简介

代码结构：

后端实现分散在LLVM源树的不同目录中。
代码生成的主要库位于lib目录及其子文件夹CodeGen，MC，TableGen和Target中：

•CodeGen目录包含所有通用代码生成算法的实现文件和头文件：指令选择，调度程序，寄存器分配以及所有他们需要的分析。

•MC目录保留了汇编器（汇编解析器），松弛算法（反汇编器）和特定目标文件惯用语（例如ELF，COFF，MachO等）的低级功能的实现。

•TableGen目录包含TableGen工具的完整实现，该工具用于根据.td文件中的高级目标描述生成C代码。

•每个后端目标都在“Target”文件夹下的不同子文件夹中实现（例如，Target/Mips）

02 LLVM后端简介

官方文档中所述的编写一个后端的基本步骤：

1、建立一个TargetMachine的子类用于描述你的硬件特性。

2、描述目标机器的寄存器集。使用TableGen从目标机器的**RegisterInfo.td**中生成寄存器定义信息、同名信息、寄存器组信息。同时还需要编写TargetRegisterInfo子类来说明寄存器如何分派和描述寄存器之间的关系。

3、描述目标机器的指令集。同样使用TableGen从目标机器的**TargetInstrFormats.td**和TargetInstrInfo.td中生成指令信息。同时还需要编写TargetInstrInfo子类来说明目标机器所支持的机器指令。

4、描述如何选择和转化LLVM IR，从一个DAG表示的指令到目标机器的本地指令。 基于**TargetInstrInfo.td**提供的信息，使用TableGen生成模式匹配成功的指令。编写代码**XXXISelDAGToDAG.cpp**来说明如何匹配模式和DAG-to-DAG指令选择等。同时还要编写**XXXISelLowering.cpp**来替换或者移除目标机器不支持的类型操作。

5、编写汇编语言打印代码，来转换LLVM IR到GAS格式的汇编。

6、（可选）支持subtargets（***）。

7、（可选）增加JIT支持和建立一个机器码生成器，用于直接生成二进制码到内存。

03 使用TableGen语言进行目标描述


```
llvm/lib/Target/RISCV/RISCVInstrFormatsC.td
llvm/lib/Target/RISCV/RISCVSystemOperands.td
llvm/lib/Target/RISCV/RISCVInstrInfoM.td
llvm/lib/Target/RISCV/RISCVInstrInfoD.td
llvm/lib/Target/RISCV/RISCVInstrInfoC.td
llvm/lib/Target/RISCV/RISCVRegisterInfo.td
llvm/lib/Target/RISCV/RISCVInstrFormats.td
llvm/lib/Target/RISCV/RISCVInstrInfoA.td
llvm/lib/Target/RISCV/RISCV.td
llvm/lib/Target/RISCV/RISCVInstrInfoF.td
llvm/lib/Target/RISCV/RISCVCallingConv.td
llvm/lib/Target/RISCV/RISCVInstrInfo.td
```

基类 llvm/include/llvm/Target/Target.td

可执行文件：llvm-tblgen

输入：.td文件

输出：.inc文件（c++文件）

03 使用TableGen语言进行目标描述

使用llvm-tblgen工具：

```
chenying@cc:~/llvm-project-release-9.x$ ./mybuild/bin/llvm-tblgen -I ./llvm/includ
e/ -I ./llvm/lib/Target/RISCV/ ./llvm/lib/Target/RISCV/RISCV.td -print-enums -clas
s=Register
F0_32, F0_64, F10_32, F10_64, F11_32, F11_64, F12_32, F12_64, F13_32, F13_64, F14_
32, F14_64, F15_32, F15_64, F16_32, F16_64, F17_32, F17_64, F18_32, F18_64, F19_32
, F19_64, F1_32, F1_64, F20_32, F20_64, F21_32, F21_64, F22_32, F22_64, F23_32, F2
3_64, F24_32, F24_64, F25_32, F25_64, F26_32, F26_64, F27_32, F27_64, F28_32, F28_
64, F29_32, F29_64, F2_32, F2_64, F30_32, F30_64, F31_32, F31_64, F3_32, F3_64, F4
_32, F4_64, F5_32, F5_64, F6_32, F6_64, F7_32, F7_64, F8_32, F8_64, F9_32, F9_64,
X0, X1, X10, X11, X12, X13, X14, X15, X16, X17, X18, X19, X2, X20, X21, X22, X23,
X24, X25, X26, X27, X28, X29, X3, X30, X31, X4, X5, X6, X7, X8, X9,
```

```
chenying@cc:~/llvm-project-release-9.x$ ./mybuild/bin/llvm-tblgen -I ./llvm/includ
e/ -I ./llvm/lib/Target/RISCV/ ./llvm/lib/Target/RISCV/RISCV.td -print-enums -clas
s=Instruction
ADD, ADDI, ADDIW, ADDW, ADJCALLSTACKDOWN, ADJCALLSTACKUP, AMOADD_D, AMOADD_D_AQ, A
MOADD_D_AQ_RL, AMOADD_D_RL, AMOADD_W, AMOADD_W_AQ, AMOADD_W_AQ_RL, AMOADD_W_RL, AM
OAND_D, AMOAND_D_AQ, AMOAND_D_AQ_RL, AMOAND_D_RL, AMOAND_W, AMOAND_W_AQ, AMOAND_W_
AQ_RL, AMOAND_W_RL, AMOMAXU_D, AMOMAXU_D_AQ, AMOMAXU_D_AQ_RL, AMOMAXU_D_RL, AMOMAX
U_W, AMOMAXU_W_AQ, AMOMAXU_W_AQ_RL, AMOMAXU_W_RL, AMOMAX_D, AMOMAX_D_AQ, AMOMAX_D_
AQ_RL, AMOMAX_D_RL, AMOMAX_W, AMOMAX_W_AQ, AMOMAX_W_AQ_RL, AMOMAX_W_RL, AMOMINU_D,
 AMOMINU_D_AQ, AMOMINU_D_AQ_RL, AMOMINU_D_RL, AMOMINU_W, AMOMINU_W_AQ, AMOMINU_W_A
Q_RL, AMOMINU_W_RL, AMOMIN_D, AMOMIN_D_AQ, AMOMIN_D_AQ_RL, AMOMIN_D_RL, AMOMIN_W,
AMOMIN_W_AQ, AMOMIN_W_AQ_RL, AMOMIN_W_RL, AMOOR_D, AMOOR_D_AQ, AMOOR_D_AQ_RL, AMOO
R_D_RL, AMOOR_W, AMOOR_W_AQ, AMOOR_W_AQ_RL, AMOOR_W_RL, AMOSWAP_D, AMOSWAP_D_AQ, A
MOSWAP_D_AQ_RL, AMOSWAP_D_RL, AMOSWAP_W, AMOSWAP_W_AQ, AMOSWAP_W_AQ_RL, AMOSWAP_W_
RL, AMOXOR_D, AMOXOR_D_AQ, AMOXOR_D_AQ_RL, AMOXOR_D_RL, AMOXOR_W, AMOXOR_W_AQ, AMO
XOR_W_AQ_RL, AMOXOR_W_RL, AND, ANDI, ANNOTATION_LABEL, AUIPC, BEQ, BGE, BGEU, BLT,
 BLTU, BNE, BUNDLE, BuildPairF64Pseudo, CFI_INSTRUCTION, COPY, COPY_TO_REGCLASS, C
SRRC, CSRRCI, CSRRS, CSRRSI, CSRRW, CSRRWI, C_ADD, C_ADDI, C_ADDI16SP, C_ADDI4SPN,
```

通过使用llvm-tblgen工具，处理.td文件并检查结果记录：

$ llvm-tblgen –help //tablegen选项

$ llvm-tblgen RISCV.td -print-enums -class=Register

$ llvm-tblgen RISCV.td -print-enums -class=Instruction

03 使用TableGen语言进行目标描述

lib/Target/RISCV/RISCV.td

```
89 //===----------------------------------------------------------
90 // Define the RISC-V target.
91 //===----------------------------------------------------------
92
93 def RISCVInstrInfo : InstrInfo {
94   let guessInstructionProperties = 0;
95 }
96
97 def RISCVAsmParser : AsmParser {
98   let ShouldEmitMatchRegisterAltName = 1;
99   let AllowDuplicateRegisterNames = 1;
100 }
101
102 def RISCVAsmWriter : AsmWriter {
103   int PassSubtarget = 1;
104 }
105
106 def RISCV : Target {
107   let InstructionSet = RISCVInstrInfo;
108   let AssemblyParsers = [RISCVAsmParser];
109   let AssemblyWriters = [RISCVAsmWriter];
110   let AllowRegisterRenaming = 1;
111 }
```

谢 谢

欢迎交流合作

2020/02/12