如何给 llvm ir 语言的 instructions 增加个一个新 flag

郑俊杰
zhengjunjie@iscas.ac.cn

```
define i32 @test_and(i32 %a,i32 %b) {
  %res = and plct_openday i32 %a, %b
  ret i32 %res
}
```

目录

- 解析
- llvm ir
- Bitcode
- 测试

解析

llvm/include/llvm/AsmParser/LLToken.h

```
16 namespace llvm {
17 namespace lltok {
18 enum Kind {
19   // Markers
20   Eof,
21   Error,
22
23   // Tokens with no info.
24   dotdotdot, // ...
25   equal,
26   comma, // =  ,
27   star,  // *
28   lsquare,
29   rsquare, // [  ]
30   lbrace,
31   rbrace, // {  }
32   less,
33   greater, // <  >
34   lparen,
35   rparen,  // (  )
36   exclaim, // !
37   bar,     // |
38   colon,   // :
39
40   kw_plct_openday,
41   kw_vscale,
42   kw_x,
43   kw_true,
44   kw_false,
45   kw_declare,
46   kw_define,
47   kw_global,
48   kw_constant,
```

此文件中定义了所有 .ll 文件词法分析器所需的枚举

在此增加了 plct_openday 的枚举

llvm/lib/AsmParser/LLLexer.cpp

lltok::Kind LLLexer::LexIdentifier()

此函数读取并返回相应字符串为枚举，增加了 KEYWORD(plct_openday);
对应上一页的我们新增加的枚举。

```
#define KEYWORD(STR)
  do {
    if (Keyword == #STR)
      return lltok::kw_##STR;
  } while (false)
```

```
lltok::Kind LLLexer::LexIdentifier() {
  const char *StartChar = CurPtr;
  const char *IntEnd = CurPtr[-1] == 'i' ? nullptr : StartChar;
  const char *KeywordEnd = nullptr;

  for (; isLabelChar(c: *CurPtr); ++CurPtr) {
    // If we decide this is an integer, remember the end of the sequence.
    if (!IntEnd && !isdigit(static_cast<unsigned char>(*CurPtr)))
      IntEnd = CurPtr;
    if (!KeywordEnd && !isalnum(static_cast<unsigned char>(*CurPtr)) &&
        *CurPtr != '_')
      KeywordEnd = CurPtr;
  }
```

```
KEYWORD(nnan);
KEYWORD(ninf);
KEYWORD(nsz);
KEYWORD(plct_openday);
KEYWORD(arcp);
KEYWORD(contract);
```

llvm/include/llvm/IR/InstrTypes.h

增加一个 PossiblyPLCTOpenDay 类继承于 BinaryOperator ，我们可以由此转换到对应类，并且增加相应函数

```
class PossiblyPLCTOpenDay : public BinaryOperator {
public:
  enum { IsPLCTOpenDay = (1 << 0)};
  void setIsPLCTOpenDay(bool B) {
    SubclassOptionalData = (SubclassOptionalData & ~IsPLCTOpenDay) | (B * IsPLCTOpenDay);
  }
  bool isPLCTOpenDay() const { return SubclassOptionalData & IsPLCTOpenDay; }
  static bool classof(const Instruction *I) {
    return I->getOpcode() == Instruction::And;
  }
  static bool classof(const Value *V) {
    return isa<Instruction>(Val: V) && classof(I: cast<Instruction>(Val: V));
  }
};
```

llvm ir

llvm/lib/AsmParser/LLParser.cpp

int LLParser::parseInstruction(Instruction *&Inst, BasicBlock *BB, PerFunctionState &PFS)

```
6411   case lltok::kw_and: {
6412     bool PLCTOpenDay = EatIfPresent(T: lltok::kw_plct_openday);
6413     if (parseLogical(&: Inst, &: PFS, Opc: KeywordVal))
6414       return true;
6415     if (PLCTOpenDay)
6416       cast<PossiblyPLCTOpenDay>(Val: Inst)->setIsPLCTOpenDay(B: true);
6417     return false;
6418   }
```

在此函数下匹配 kw_and，尝试获取 plct_openday 的 flag，检查 Instruction，再尝试在 Instruction 上设置 PLCTOpenDay 的 flag

```
// Instruction Parsing.  Each instruction parsing routine can return with a
// normal result, an error result, or return having eaten an extra comma.
enum InstResult { InstNormal = 0, InstError = 1, InstExtraComma = 2 };
```

llvm/lib/IR/AsmWriter.cpp

```
static void WriteOptimizationInfo(raw_ostream &Out, const User *U) {
  if (const FPMathOperator *FPO = dyn_cast<const FPMathOperator>(U))
    Out << FPO->getFastMathFlags();

  if (const OverflowingBinaryOperator *OBO =
        dyn_cast<OverflowingBinaryOperator>(U)) {
    if (OBO->hasNoUnsignedWrap())
      Out << " nuw";
    if (OBO->hasNoSignedWrap())
      Out << " nsw";
  } else if (const PossiblyPLCTOpenDay *PLCTOpenDay =
             dyn_cast<PossiblyPLCTOpenDay>(U)){
    if (PLCTOpenDay->isPLCTOpenDay())
      Out << " plct_openday";
  } else if (const PossiblyExactOperator *Div =
               dyn_cast<PossiblyExactOperator>(U)) {
    if (Div->isExact())
      Out << " exact";
  } else if (const PossiblyDisjointInst *PDI =
                dyn_cast<PossiblyDisjointInst>(U)) {
    if (PDI->isDisjoint())
      Out << " disjoint";
  } else if (const GEPOperator *GEP = dyn_cast<GEPOperator>(U)) {
    if (GEP->isInBounds())
      Out << " inbounds";
  } else if (const auto *NNI = dyn_cast<PossiblyNonNegInst>(U)) {
    if (NNI->hasNonNeg())
      Out << " nneg";
  }
}
```

在此文件中增加相应逻辑，用来打印新增加的 plct_openday flag

bitcode

llvm/include/llvm/Bitcode/LLVMBitCodes.h

加入一个新枚举用来标识我们的 PLCT_OPENDAY flag

```
enum PossiblyPLCTOpenDayFlags { PLCT_OPENDAY = 0 };
```

llvm/lib/Bitcode/Reader/BitcodeReader.cpp
Error BitcodeReader::parseFunctionBody(Function *F)

```
case bitc::FUNC_CODE_INST_BINOP: {    // BINOP: [opval, ty, opval, opcode]
  unsigned OpNum = 0;
  Value *LHS, *RHS;
  unsigned TypeID;
  if (getValueTypePair(Record, &Slot: OpNum, InstNum: NextValueNo, &ResVal: LHS, &: T
      popValue(Record, &Slot: OpNum, InstNum: NextValueNo, Ty: LHS->getType(), TyID:
                ConstExprInsertBB: CurBB) ||
      OpNum+1 > Record.size())
    return error(Message: "Invalid record");

  int Opc = getDecodedBinaryOpcode(Val: Record[OpNum++], Ty: LHS->getType());
  if (Opc == -1)
    return error(Message: "Invalid record");
  I = BinaryOperator::Create(Op: (Instruction::BinaryOps)Opc, S1: LHS, S2: RHS);
  ResTypeID = TypeID;
  InstructionList.push_back(Elt: I);
  if (OpNum < Record.size()) {
    if (Opc == Instruction::Add ||
        Opc == Instruction::Sub ||
        Opc == Instruction::Mul ||
        Opc == Instruction::Shl) {
      if (Record[OpNum] & (1 << bitc::OBO_NO_SIGNED_WRAP))
        cast<BinaryOperator>(Val: I)->setHasNoSignedWrap(b: true);
      if (Record[OpNum] & (1 << bitc::OBO_NO_UNSIGNED_WRAP))
        cast<BinaryOperator>(Val: I)->setHasNoUnsignedWrap(b: true);
    } else if (Opc == Instruction::And) {
      if (Record[OpNum] & (1 << bitc::PLCT_OPENDAY))
        cast<PossiblyPLCTOpenDay>(Val: I)->setIsPLCTOpenDay(B: true);
    } else if (Opc == Instruction::SDiv ||
               Opc == Instruction::UDiv ||
               Opc == Instruction::LShr ||
               Opc == Instruction::AShr) {
      if (Record[OpNum] & (1 << bitc::PEO_EXACT))
        cast<BinaryOperator>(Val: I)->setIsExact(b: true);
    } else if (Opc == Instruction::Or) {
      if (Record[OpNum] & (1 << bitc::PDI_DISJOINT))
        cast<PossiblyDisjointInst>(Val: I)->setIsDisjoint(B: true);
    } else if (isa<FPMathOperator>(Val: I)) {
      FastMathFlags FMF = getDecodedFastMathFlags(Val: Record[OpNum]);
      if (FMF.any())
        I->setFastMathFlags(FMF);
    }
```

增加逻辑获取并设置 PLCTOpenDay flag

llvm/lib/Bitcode/Writer/BitcodeWriter.cpp

```
static uint64_t getOptimizationFlags(const Value *V) {
  uint64_t Flags = 0;

  if (const auto *OBO = dyn_cast<OverflowingBinaryOperator>(Val: V)) {
    if (OBO->hasNoSignedWrap())
      Flags |= 1 << bitc::OBO_NO_SIGNED_WRAP;
    if (OBO->hasNoUnsignedWrap())
      Flags |= 1 << bitc::OBO_NO_UNSIGNED_WRAP;
  } else if (const auto *PlctOpenDay: PossiblyPLCTOpenDay const *=dyn_cast<PossiblyPLCTOpenDay>(Val: V)) {
    if (PlctOpenDay->isPLCTOpenDay())
      Flags |= 1 << bitc::PLCT_OPENDAY;
  } else if (const auto *PEO: PossiblyExactOperator const * = dyn_cast<PossiblyExactOperator>(Val: V)) {
    if (PEO->isExact())
      Flags |= 1 << bitc::PEO_EXACT;
  } else if (const auto *PDI: PossiblyDisjointInst const * = dyn_cast<PossiblyDisjointInst>(Val: V)) {
    if (PDI->isDisjoint())
      Flags |= 1 << bitc::PDI_DISJOINT;
  } else if (const auto *FPMO: FPMathOperator const * = dyn_cast<FPMathOperator>(Val: V)) {
    if (FPMO->hasAllowReassoc())
      Flags |= bitc::AllowReassoc;
    if (FPMO->hasNoNaNs())
      Flags |= bitc::NoNaNs;
    if (FPMO->hasNoInfs())
      Flags |= bitc::NoInfs;
    if (FPMO->hasNoSignedZeros())
      Flags |= bitc::NoSignedZeros;
    if (FPMO->hasAllowReciprocal())
      Flags |= bitc::AllowReciprocal;
    if (FPMO->hasAllowContract())
      Flags |= bitc::AllowContract;
    if (FPMO->hasApproxFunc())
      Flags |= bitc::ApproxFunc;
  } else if (const auto *NNI: PossiblyNonNegInst const * = dyn_cast<PossiblyNonNegInst>(Val: V)) {
    if (NNI->hasNonNeg())
      Flags |= 1 << bitc::PNNI_NON_NEG;
  }

  return Flags;
}
```

检查 PLCTOpenDay 的 flag

```cpp
void Instruction::copyIRFlags(const Value *V, bool IncludeWrapFlags) {
  // Copy the wrapping flags.
  if (IncludeWrapFlags && isa<OverflowingBinaryOperator>(Val: this)) {
    if (auto *OB = dyn_cast<OverflowingBinaryOperator>(Val: V)) {
      setHasNoSignedWrap(b: OB->hasNoSignedWrap());
      setHasNoUnsignedWrap(b: OB->hasNoUnsignedWrap());
    }
  }

  // Copy the exact flag.
  if (auto *PE: const PossiblyExactOperator * = dyn_cast<PossiblyExactOperator>(Val: V))
    if (isa<PossiblyExactOperator>(Val: this))
      setIsExact(b: PE->isExact());

  if (auto *SrcPD: const PossiblyDisjointInst * = dyn_cast<PossiblyDisjointInst>(Val: V))
    if (auto *DestPD: PossiblyDisjointInst * = dyn_cast<PossiblyDisjointInst>(Val: this))
      DestPD->setIsDisjoint(B: SrcPD->isDisjoint());

  // Copy the fast-math flags.
  if (auto *FP: const FPMathOperator * = dyn_cast<FPMathOperator>(Val: V))
    if (isa<FPMathOperator>(Val: this))
      copyFastMathFlags(FMF: FP->getFastMathFlags());

  if (auto *SrcPLCT: const PossiblyPLCTOpenDay * = dyn_cast<PossiblyPLCTOpenDay>(Val: V))
    if (auto *DestPLCT: PossiblyPLCTOpenDay * = dyn_cast<PossiblyPLCTOpenDay>(Val: this))
      DestPLCT->setIsPLCTOpenDay(B: SrcPLCT->isPLCTOpenDay());


  if (auto *SrcGEP: const GetElementPtrInst * = dyn_cast<GetElementPtrInst>(Val: V))
    if (auto *DestGEP: GetElementPtrInst * = dyn_cast<GetElementPtrInst>(Val: this))
      DestGEP->setIsInBounds(b: SrcGEP->isInBounds() || DestGEP->isInBounds());

  if (auto *NNI: const PossiblyNonNegInst * = dyn_cast<PossiblyNonNegInst>(Val: V))
    if (isa<PossiblyNonNegInst>(Val: this))
      setNonNeg(b: NNI->hasNonNeg());
}
```

```cpp
void Instruction::andIRFlags(const Value *V) {
  if (auto *OB = dyn_cast<OverflowingBinaryOperator>(Val: V)) {
    if (isa<OverflowingBinaryOperator>(Val: this)) {
      setHasNoSignedWrap(b: hasNoSignedWrap() && OB->hasNoSignedWrap());
      setHasNoUnsignedWrap(b: hasNoUnsignedWrap() && OB->hasNoUnsignedWrap());
    }
  }

  if (auto *PE: const PossiblyExactOperator * = dyn_cast<PossiblyExactOperator>(Val: V))
    if (isa<PossiblyExactOperator>(Val: this))
      setIsExact(b: isExact() && PE->isExact());

  if (auto *SrcPD: const PossiblyDisjointInst * = dyn_cast<PossiblyDisjointInst>(Val: V))
    if (auto *DestPD: PossiblyDisjointInst * = dyn_cast<PossiblyDisjointInst>(Val: this))
      DestPD->setIsDisjoint(B: DestPD->isDisjoint() && SrcPD->isDisjoint());


  if (auto *SrcPLCT: const PossiblyPLCTOpenDay * = dyn_cast<PossiblyPLCTOpenDay>(Val: V))
    if (auto *DestPLCT: PossiblyPLCTOpenDay * = dyn_cast<PossiblyPLCTOpenDay>(Val: this))
      DestPLCT->setIsPLCTOpenDay(B: DestPLCT->isPLCTOpenDay() && SrcPLCT->isPLCTOpenDay());


  if (auto *FP: const FPMathOperator * = dyn_cast<FPMathOperator>(Val: V)) {
    if (isa<FPMathOperator>(Val: this)) {
      FastMathFlags FM = getFastMathFlags();
      FM &= FP->getFastMathFlags();
      copyFastMathFlags(FMF: FM);
    }
  }

  if (auto *SrcGEP: const GetElementPtrInst * = dyn_cast<GetElementPtrInst>(Val: V))
    if (auto *DestGEP: GetElementPtrInst * = dyn_cast<GetElementPtrInst>(Val: this))
      DestGEP->setIsInBounds(b: SrcGEP->isInBounds() && DestGEP->isInBounds());

  if (auto *NNI: const PossiblyNonNegInst * = dyn_cast<PossiblyNonNegInst>(Val: V))
    if (isa<PossiblyNonNegInst>(Val: this))
      setNonNeg(b: hasNonNeg() && NNI->hasNonNeg());
}
```

测试

llvm/test/Assembler/flags.ll

; RUN: llvm-as < %s | llvm-dis | FileCheck %s

```
define i32 @test_and(i32 %a,i32 %b) {
; CHECK: %res = and plct_openday i32 %a, %b
  %res = and plct_openday i32 %a, %b
  ret i32 %res
}
```

llvm-as 转换文件到 llvm 字节码，再由 llvm-dis 转为 ir 以此来验证代码正确性

```
[z572@m build]$ cat /tmp/ba.ll
define i32 @test_and(i32 %a,i32 %b) {
; CHECK: %res = and plct_openday i32 %a, %b
  %res = and plct_openday i32 %a, %b
  ret i32 %res
}
[z572@m build]$ cat /tmp/ba.ll  | ./bin/opt -S
; ModuleID = '<stdin>'
source_filename = "<stdin>"

define i32 @test_and(i32 %a, i32 %b) {
  %res = and plct_openday i32 %a, %b
  ret i32 %res
}
```

参考

- https://github.com/llvm/llvm-project/pull/72583
- https://github.com/llvm/llvm-project/pull/67982
- https://discourse.llvm.org/t/rfc-add-or-disjoint-flag/75036/2
- https://discourse.llvm.org/t/rfc-add-zext-nneg-flag/73914

END