基于rvv-intrinsic 分析llvm中指令选择

廖春玉

2021/1/24

目录

简介

- cat test.ll

```
declare <vscale x 1 x i8> @llvm.riscv.vadd.nxv1i8.nxv1i8(
  <vscale x 1 x i8>,
  <vscale x 1 x i8>,
  i64);

define <vscale x 1 x i8> @intrinsic_vadd_vv_nxv1i8_nxv1i8_nxv1i8(<vscale x 1 x i8> %0, <vscale x 1 x
i8> %1, i64 %2) nounwind {

  %a = call <vscale x 1 x i8> @llvm.riscv.vadd.nxv1i8.nxv1i8(
    <vscale x 1 x i8> %0,
    <vscale x 1 x i8> %1,
    i64 %2)

  ret <vscale x 1 x i8> %a
}
```

来自：llvm/test/CodeGen/RISCV/rvv/vadd-rv64.ll

目录

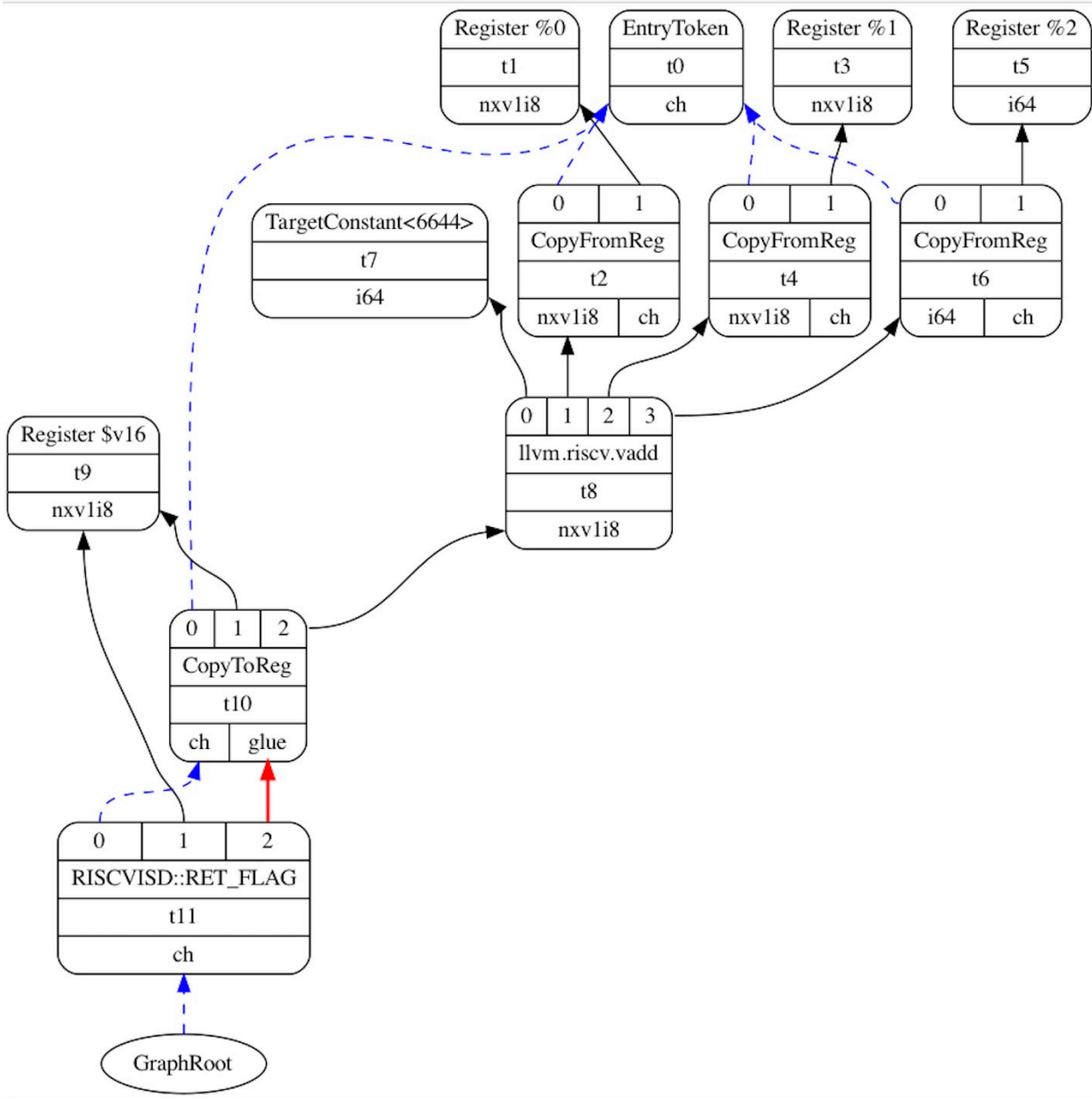
什么是指令选择呢

- 代码生成的一部分
- LLVM IR -> MachineInstr

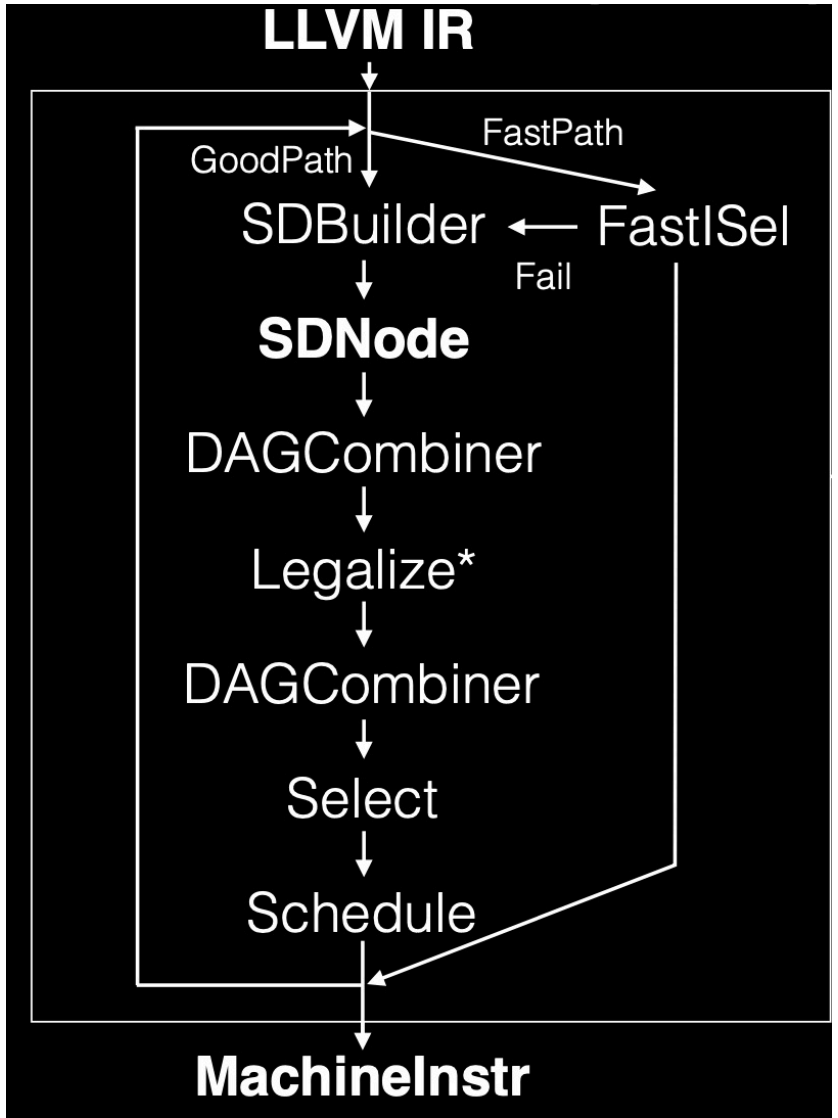
- Target-independent code generation algorithms
  - Instruction Selection
    - Introduction to SelectionDAGs
    - SelectionDAG Instruction Selection Process
    - Initial SelectionDAG Construction
    - SelectionDAG LegalizeTypes Phase
    - SelectionDAG Legalize Phase
    - SelectionDAG Optimization Phase: the DAG Combiner
    - SelectionDAG Select Phase
    - SelectionDAG Scheduling and Formation Phase
    - Future directions for the SelectionDAG
  - SSA-based Machine Code Optimizations
  - Live Intervals
    - Live Variable Analysis
    - Live Intervals Analysis
  - Register Allocation
    - How registers are represented in LLVM
    - Mapping virtual registers to physical registers
    - Handling two address instructions
    - The SSA deconstruction phase
    - Instruction folding
    - Built in register allocators
  - Prolog/Epilog Code Insertion
  - Late Machine Code Optimizations
  - Code Emission
    - Emitting function stack size information
  - VLIW Packetizer
    - Mapping from instructions to functional units
    - How the packetization tables are generated and used

SelectionDAG简介

- LLVM使用基于SelectionDAG的指令选择器
- 有向无环图
- 节点SDNode：Opcode
- 边SDValue



SelectionDAG过程



来自：https://llvm.org/devmtg/2015-10/slides/Colombet-GlobalInstructionSelection.pdf

SelectionDAGBuilder

- LLVM IR 翻译为 SelectionDAG
- 以BasicBlock为单位，对每条指令调用SelectionDAGBuilder::visit()

```
void SelectionDAGISel::SelectBasicBlock(BasicBlock::const_iterator Begin,
                                        BasicBlock::const_iterator End,
                                        bool &HadTailCall) {
  // Allow creating illegal types during DAG building for the basic block.
  CurDAG->NewNodesMustHaveLegalTypes = false;
  // Lower the instructions. If a call is emitted as a tail call, cease
emitting
  // nodes for this block.
  for (BasicBlock::const_iterator I = Begin; I != End && !SDB->HasTailCall;
++I) {
    if (!ElidedArgCopyInstrs.count(&*I))
      SDB->visit(*I);
  }
  ......

  CodeGenAndEmitDAG();
}
```

SelectionDAGBuilder

- 根据opcode不同访问不同的函数，visit##OPCODE
- 针对intrinsic的实现
  - SelectionDAGBuilder::visitIntrinsicCall
  - SelectionDAGBuilder::visitTargetIntrinsic
- SelectionDAG::getNode()创建SDNode

```
void SelectionDAGBuilder::visitTargetIntrinsic(const CallInst &I,
                                               unsigned Intrinsic) {
......

  } else if (!HasChain) {
    Result = DAG.getNode(ISD::INTRINSIC_WO_CHAIN, getCurSDLoc(), VTs, Ops);
  } else if (!I.getType()->isVoidTy()) {
    Result = DAG.getNode(ISD::INTRINSIC_W_CHAIN, getCurSDLoc(), VTs, Ops)

......
```

Legalize

- 针对类型，将 DAG 中的类型转换为目标平台支持的类型
  - 标量
    - Promoting，小类型 ->大类型， 例如：f32 类型提升到 f64 类型
    - Expanding，大类型 ->小类型， 例如：i64 类型展开成 i32 类型的组合对
  - 向量
    - Splitting，拆分成可以支持的向量类型
    - Widening，在向量末端增加元素
    - Scalarizing，转换为标量

Legalize

- 针对操作，将 DAG 中的操作转换为目标平台支持的操作

There are cases when a vector type is directly supported by a backend, meaning that there is a register class for it, but a specific operation on a given vector type is not. For example, `x86` with `SSE2` supports the `v4i32` vector type. However, there is no `x86` instruction to support `ISD::OR` on `v4i32` types, but only on `v2i64`. Therefore, the vector legalizer handles those cases and promotes or expands the operations, using legal types for the instruction. The target can also handle the legalization in a custom manner. In the aforementioned `ISD::OR` case, the operation is promoted to use `v2i64` type. Have a look at the following code snippet of `lib/Target/X86/X86ISelLowering.cpp`:

```
setOperationAction(ISD::OR, v4i32, Promote);
AddPromotedToType (ISD::OR, v4i32, MVT::v2i64);
```

来自： 《Getting Started with LLVM Core Libraries》

Legalize

- 针对intrinsic的实现

```
setOperationAction(ISD::INTRINSIC_WO_CHAIN, MVT::i32, Custom);
setOperationAction(ISD::INTRINSIC_W_CHAIN, MVT::i32, Custom);
......


SDValue RISCVTargetLowering::LowerINTRINSIC_WO_CHAIN(SDValue Op,
                                                     SelectionDAG &DAG) const {
  ......
     if (OpVT == MVT::i8 || OpVT == MVT::i16 ||
           (OpVT == MVT::i32 && Subtarget.is64Bit())) {
         // If the operand is a constant, sign extend to increase our chances
         // of being able to use a .vi instruction. ANY_EXTEND would become a
         // a zero extend and the simm5 check in isel would fail.
         // FIXME: Should we ignore the upper bits in isel instead?
         unsigned ExtOpc = isa<ConstantSDNode>(ScalarOp) ? ISD::SIGN_EXTEND
                                                         : ISD::ANY_EXTEND;
         ScalarOp = DAG.getNode(ExtOpc, DL, Subtarget.getXLenVT(), ScalarOp);
         return DAG.getNode(ISD::INTRINSIC_WO_CHAIN, DL, Op.getValueType(),
                            Operands);
       }

 }
```

llvm-project/llvm/lib/Target/RISCV/RISCVISelLowering.cpp

Combiner

- 简化DAG，合并/消除冗余节点
- 被多次执行
  - DAG被创建创建后
  - 每次合法化后

- 目前针对intrinisic没有添加特定优化
  - 参考sve，预测vcreate/vset/vget相关的intrinsic可能会有修改

Select

- SelectionDAG -> MachineDAG
- 节点：指令

- llvm中intrinsic实现

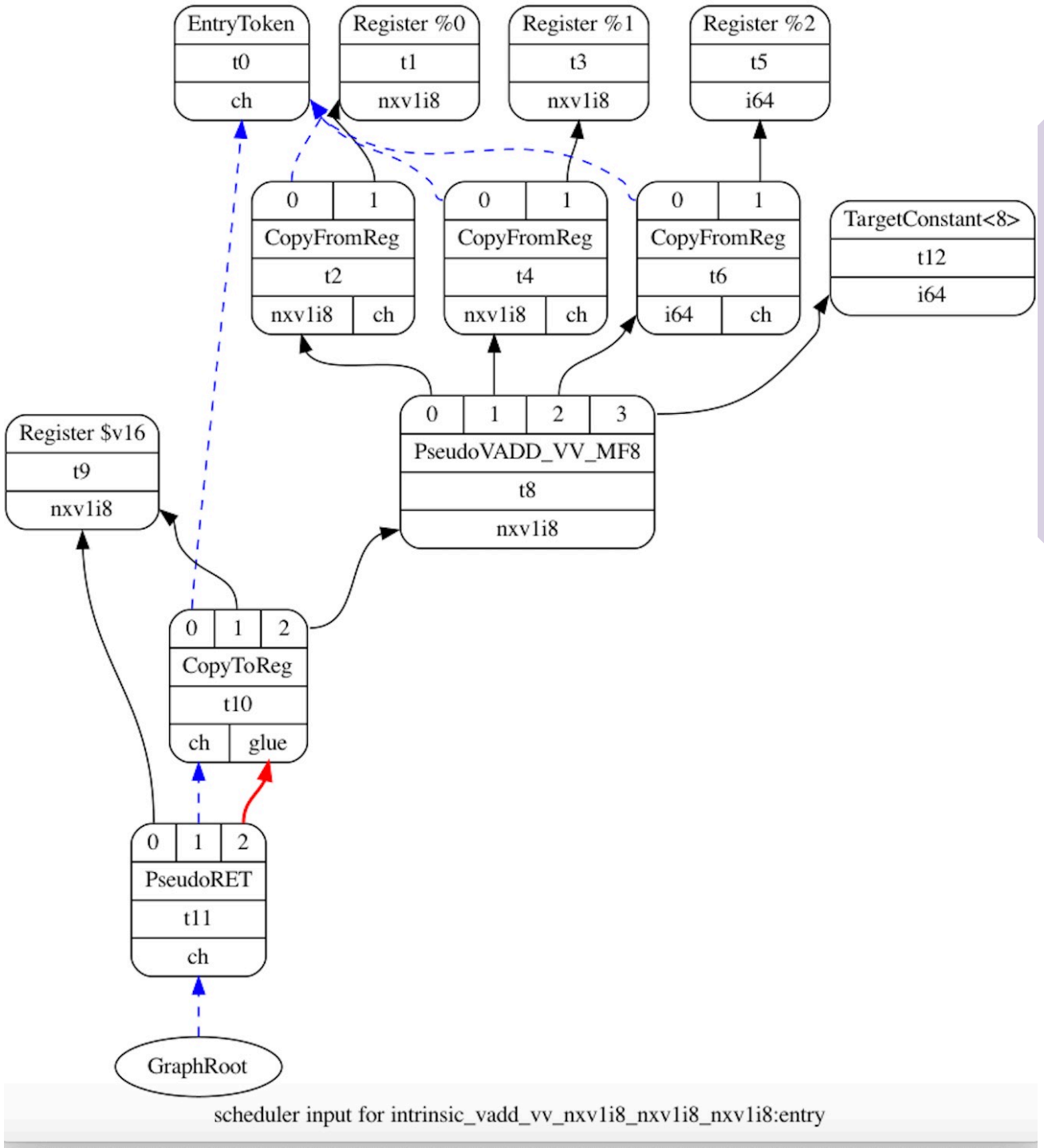
```
SelectBasicBlock

CodeGenAndEmitDAG

DoInstructionSelection

Select ( C++ )

SelectCode ( td )
```



Select

- 处理vsetvli指令
  - <Target>Node
  - getMachineNode
  - 依赖

```
void RISCVDAGToDAGISel::Select(SDNode *Node) {

   case ISD::INTRINSIC_W_CHAIN: {
    unsigned IntNo = cast<ConstantSDNode>(Node->getOperand(1))->getZExtValue()
    switch (IntNo) {

    case Intrinsic::riscv_vsetvli: {
     ......

      ReplaceNode(Node,
                  CurDAG->getMachineNode(RISCV::PseudoVSETVLI, DL, XLenVT,
                                         MVT::Other, VLOperand, VTypeIOp,
                                         /* Chain */ Node->getOperand(0)));
      return;
    }

    SelectCode(Node);
  }
```

llvm-project/llvm/lib/Target/RISCV/RISCVISelDAGToDAG.cpp

Select

- RISCVInstrInfoVPseudos.td中定义的

```
defm "" : VPatBinaryV_VV_VX_VI<"int_riscv_vadd", "PseudoVADD", AllIntegerVectors>;
......
class  VPatBinaryNoMask<string intrinsic_name,
                        string inst,
                        string kind,
                        ValueType result_type,
                        ValueType op1_type,
                        ValueType op2_type,
                        int sew,
                        LMULInfo vlmul,
                        VReg op1_reg_class,
                        DAGOperand op2_kind> :
  Pat<(result_type (!cast<Intrinsic>(intrinsic_name)
                   (op1_type op1_reg_class:$rs1),
                   (op2_type op2_kind:$rs2),
                   (XLenVT GPR:$vl))),
                   (!cast<Instruction>(inst#"_"#kind#"_"#vlmul.MX)
                   (op1_type op1_reg_class:$rs1),
                   ToFPR32<op2_type, op2_kind, "rs2">.ret,
                   (NoX0 GPR:$vl), sew)>;
```

Select

- Tablegen生成的build/lib/Target/RISCV/RISCVGenDAGISel.inc文件
  - RSICVDAGToDAGISel::SelectCode
  - SelectionDAGISel::SelectCodeCommon
    - 匹配成功则替换node
    - 否则SelectionDAGISel::CannotYetSelect

目录

指令选择log分析

- 命令
  - **llc** -mtriple=riscv64 -mattr=+experimental-v -verify-machineinstrs --debug-only=isel test.ll

参考

- https://llvm.org/docs/CodeGenerator.html#instruction-selection-section
- https://llvm.org/devmtg/2015-10/slides/Colombet-GlobalInstructionSelection.pdf

谢谢