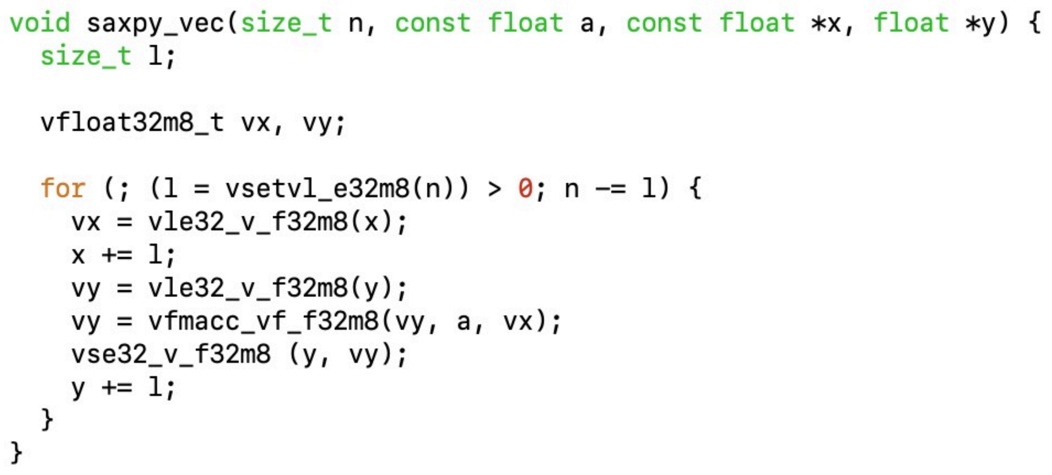
如何给 rvv-llvm 添加一个 rvv intrinsic

1 rvv-llvm 简介

- PLCT 实验室对 rvv-instrinsc 的支持
- https://github.com/isrc-cas/rvv-llvm

2 intrinsic 简介

- Intrinsic 一般是指高级编程语言中的低级汇编语言接口
- 让 C/C++可以访问所有 RISCV-V “V”扩展的指令
- https://github.com/riscv/rvv-intrinsic-doc
- Rvv-saxpy.c 例子：



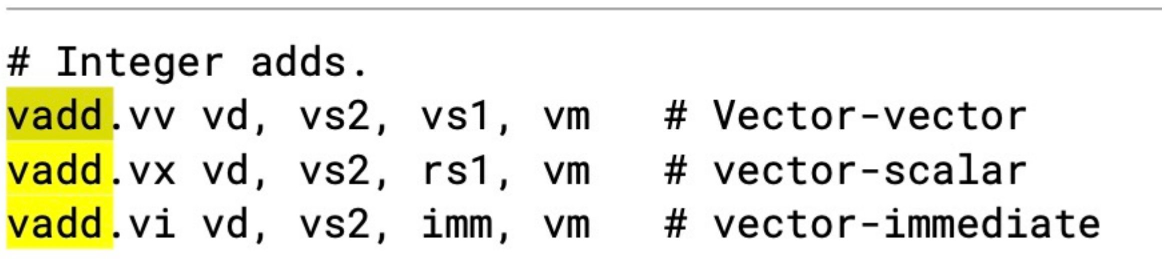
```
void saxpy_vec(size_t n, const float a, const float *x, float *y) {
  size_t l;

  vfloat32m8_t vx, vy;

  for (; (l = vsetvl_e32m8(n)) > 0; n -= l) {
    vx = vle32_v_f32m8(x);
    x += l;
    vy = vle32_v_f32m8(y);
    vy = vfmacc_vf_f32m8(vy, a, vx);
    vse32_v_f32m8 (y, vy);
    y += l;
  }
}
```

3 添加 vadd

- **vadd 8bit 相关的 intrinsic 函数，函数在 `rvv_intrinsic_funcs.md` 文件中查看**

```
vint8mf8_t vadd_vv_i8mf8 (vint8mf8_t op1, vint8mf8_t op2);
vint8mf8_t vadd_vx_i8mf8 (vint8mf8_t op1, int8_t op2);
vint8mf4_t vadd_vv_i8mf4 (vint8mf4_t op1, vint8mf4_t op2);
vint8mf4_t vadd_vx_i8mf4 (vint8mf4_t op1, int8_t op2);
vint8mf2_t vadd_vv_i8mf2 (vint8mf2_t op1, vint8mf2_t op2);
vint8mf2_t vadd_vx_i8mf2 (vint8mf2_t op1, int8_t op2);
vint8m1_t vadd_vv_i8m1 (vint8m1_t op1, vint8m1_t op2);
vint8m1_t vadd_vx_i8m1 (vint8m1_t op1, int8_t op2);
vint8m2_t vadd_vv_i8m2 (vint8m2_t op1, vint8m2_t op2);
vint8m2_t vadd_vx_i8m2 (vint8m2_t op1, int8_t op2);
vint8m4_t vadd_vv_i8m4 (vint8m4_t op1, vint8m4_t op2);
vint8m4_t vadd_vx_i8m4 (vint8m4_t op1, int8_t op2);
vint8m8_t vadd_vv_i8m8 (vint8m8_t op1, vint8m8_t op2);
vint8m8_t vadd_vx_i8m8 (vint8m8_t op1, int8_t op2);
// masked functions
vint8mf8_t vadd_vv_i8mf8_m (vbool64_t mask, vint8mf8_t maskedoff, vint8mf8_t op1,
vint8mf8_t op2);
vint8mf8_t vadd_vx_i8mf8_m (vbool64_t mask, vint8mf8_t maskedoff, vint8mf8_t op1,
int8_t op2);
vint8mf4_t vadd_vv_i8mf4_m (vbool32_t mask, vint8mf4_t maskedoff, vint8mf4_t op1,
vint8mf4_t op2);
vint8mf4_t vadd_vx_i8mf4_m (vbool32_t mask, vint8mf4_t maskedoff, vint8mf4_t op1,
int8_t op2);
vint8mf2_t vadd_vv_i8mf2_m (vbool16_t mask, vint8mf2_t maskedoff, vint8mf2_t op1,
vint8mf2_t op2);
vint8mf2_t vadd_vx_i8mf2_m (vbool16_t mask, vint8mf2_t maskedoff, vint8mf2_t op1,
int8_t op2);
vint8m1_t vadd_vv_i8m1_m (vbool8_t mask, vint8m1_t maskedoff, vint8m1_t op1, vint8m1_t
op2);
vint8m1_t vadd_vx_i8m1_m (vbool8_t mask, vint8m1_t maskedoff, vint8m1_t op1, int8_t
op2);
vint8m2_t vadd_vv_i8m2_m (vbool4_t mask, vint8m2_t maskedoff, vint8m2_t op1, vint8m2_t
op2);
vint8m2_t vadd_vx_i8m2_m (vbool4_t mask, vint8m2_t maskedoff, vint8m2_t op1, int8_t
op2);
vint8m4_t vadd_vv_i8m4_m (vbool2_t mask, vint8m4_t maskedoff, vint8m4_t op1, vint8m4_t
op2);
vint8m4_t vadd_vx_i8m4_m (vbool2_t mask, vint8m4_t maskedoff, vint8m4_t op1, int8_t
op2);
vint8m8_t vadd_vv_i8m8_m (vbool1_t mask, vint8m8_t maskedoff, vint8m8_t op1, vint8m8_t
op2);
vint8m8_t vadd_vx_i8m8_m (vbool1_t mask, vint8m8_t maskedoff, vint8m8_t op1, int8_t
op2);
```

rvv vadd汇编指令：



```
# Integer adds.
vadd.vv vd, vs2, vs1, vm   # Vector-vector
vadd.vx vd, vs2, rs1, vm   # vector-scalar
vadd.vi vd, vs2, imm, vm   # vector-immediate
```

- **clang 中添加函数需要修改的文件 riscv_vector.td， 用于生成必要的头文件和代码片段，**
  a) **clang/include/clang/Basic/riscv_vector.td**，定义数据类型，intrinsic 函数：

```
1. defm vadd : int_binary_v_vv_vx;

2. multiclass int_binary_v_vv_vx : signed_binary_v_vv_vx, unsigned_binary_v_vv_vx;

3. multiclass signed_binary_v_vv_vx {
  foreach I = AllSignedVectorType in {
    defm NAME : Binary<NAME, I.Value, I.Value, I.Value, "_vv", I.Name, I.BoolValue, [-
1], [-1, 0]>;
    defm NAME : Binary<NAME, I.Value, I.Value, I.ElemType.Value, "_vx", I.Name,
I.BoolValue, [-1, 1], [-1, 0, 3]>;
  }
}
```

```
4. multiclass Binary<string name,/ string result_type, string arg1_type, string
   arg2_type, string infix, string suffix, string mask_type, list<int>
   anytype_operands,list<int> mask_anytype_operands>
  : BinaryMaskOff<name, result_type, arg1_type, arg2_type, infix, suffix, mask_type,
anytype_operands, mask_anytype_operands>,
    BinaryMaskOn<name, result_type, arg1_type, arg2_type, infix, suffix, mask_type,
anytype_operands, mask_anytype_operands>;
```

```
5. multiclass BinaryMaskOff<string name, string result_type, string arg1_type,
                         string arg2_type, string infix, string suffix,
                         string mask_type, list<int> anytype_operands,
                         list<int> mask_anytype_operands> {
  def NAME#infix#suffix : Inst<name, [result_type, arg1_type, arg2_type],
                               infix, suffix, 0, anytype_operands>;
}
```

```
6. class Inst<string name, list<string> types, string inf, string suf,
           int is_mask, list<int> anytype_operands, code definition = ""> {
  string IntrinsicName = name;
  list<string> BuiltinStr = types;
  string Infix = inf;
  string Suffix = suf;
  // The list of indexes for the overload type. Use '-1' to refer to result
  // type, '0' to refer to the first argument type and so on. This is used
  // to generate clang builtin codegen snippet. The given indexes must be
  // consisted with llvm/include/llvm/IR/IntrinsicsRISCV.td. For an example,
  // the index list of
  //   class V_VX : Intrinsic<[ llvm_anyvector_ty ],
  //                          [ LLVMMatchType<0>, llvm_anyint_ty ],
  //                          [ IntrNoMem ]>;
  // should be:
  //   [-1, 1]
  list<int> AnyTypeOperands = anytype_operands;
  int Mask = is_mask;
  // If not empty, the emitter will use it to define the intrinsic function.
  // Otherwise, the emitter will generate intrinsic in the default way.
  code CustomDef = definition;
  // Should emit BUILTIN macro. Set to false if the intrinsic function can
  // be implemented by other existing builtin functions.
  bit ShouldEmitBuiltin = 1;
}
```

b) **clang/utils/TableGen/RISCVVectorEmitter.cpp**， 根据 riscv_vector.td 生成必要的代码片段，主要生成到 build 目录的三个文件中。

c) clang 中其他被修改的文件

1) clang/include/clang/AST/Type.h
2) clang/include/clang/Basic/Attr.td
3) clang/include/clang/Basic/DiagnosticSemaKinds.td
4) clang/lib/CodeGen/CodeGenTypes.cpp
5) clang/lib/CodeGen/TargetInfo.cpp
6) clang/lib/Sema/SemaType.cpp
7) clang/lib/Basic/Targets/RISCV.cpp
8) clang/lib/Basic/Targets/RISCV.h
9) clang/lib/CodeGen/CGBuiltin.cpp
10) clang/lib/CodeGen/CodeGenFunction.h
11) ……

- **正确添加后在 build 目录下生成头文件和代码片段，以 i8m1 为例子进行说明，同时也是添加后需要检查是否正确：**

  a) riscv_vector.h

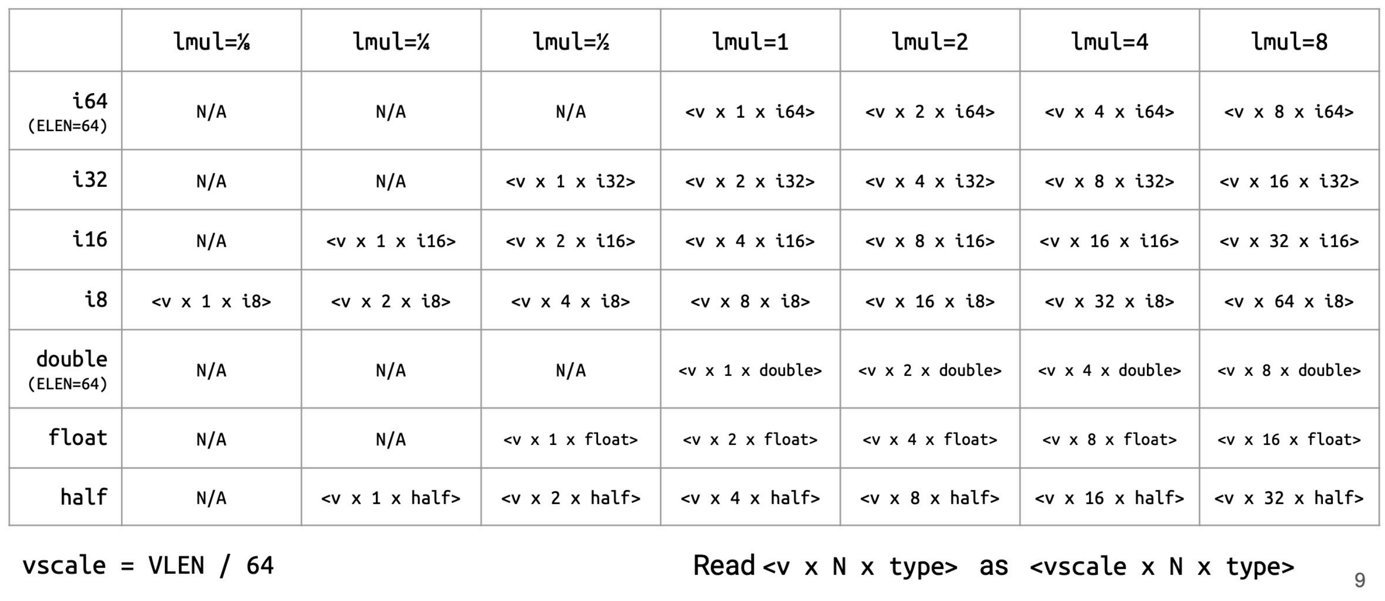
```
#define vadd_vv_i8m1(...) __builtin_riscv_vadd_vv_i8m1(__VA_ARGS__)
```

  a) riscv_vector_builtins.inc

```
BUILTIN(__builtin_riscv_vadd_vv_i8m1, "q8cq8cq8c", "")
```

其中 vadd_vv_i8m1 函数有一个返回和两个输入，均是 vint8m1_t 类型，也就是<v x 8 x i8>（具体类型见下图），其中 q8c

q=v, 8 是数量，i8 是 char 类型(见 Builtins.def)。



	lmul=⅛	lmul=¼	lmul=½	lmul=1	lmul=2	lmul=4	lmul=8
i64 (ELEN=64)	N/A	N/A	N/A	<v x 1 x i64>	<v x 2 x i64>	<v x 4 x i64>	<v x 8 x i64>
i32	N/A	N/A	<v x 1 x i32>	<v x 2 x i32>	<v x 4 x i32>	<v x 8 x i32>	<v x 16 x i32>
i16	N/A	<v x 1 x i16>	<v x 2 x i16>	<v x 4 x i16>	<v x 8 x i16>	<v x 16 x i16>	<v x 32 x i16>
i8	<v x 1 x i8>	<v x 2 x i8>	<v x 4 x i8>	<v x 8 x i8>	<v x 16 x i8>	<v x 32 x i8>	<v x 64 x i8>
double (ELEN=64)	N/A	N/A	N/A	<v x 1 x double>	<v x 2 x double>	<v x 4 x double>	<v x 8 x double>
float	N/A	N/A	<v x 1 x float>	<v x 2 x float>	<v x 4 x float>	<v x 8 x float>	<v x 16 x float>
half	N/A	<v x 1 x half>	<v x 2 x half>	<v x 4 x half>	<v x 8 x half>	<v x 16 x half>	<v x 32 x half>

vscale = VLEN / 64 Read <v x N x type> as <vscale x N x type> 9

  b) riscv_vector_builtin_cg.inc

……

```
case RISCV::BI__builtin_riscv_vadd_vv_i8m1:
case RISCV::BI__builtin_riscv_vadd_vv_i8m2:
……
{
  Function *F = CGM.getIntrinsic(Intrinsic::riscv_vadd_vv, {ResultType});
  return Builder.CreateCall(F, Ops);

}
```

1. 其中 riscv_vadd_vv 是 intrinsic 的名字，在 build/include/llvm/IR/IntrinsicsRISCV.h 中查看全部支持的 intrinsic;
2. ResultType 是返回的数据类型，如果打开 riscv_vector_builtins.inc 会发现，CGM.getIntrinsic 的第二个参数有如下情况：

```
Eg: Function *F = CGM.getIntrinsic(Intrinsic::riscv_vadd_vv_mask,
{ResultType, Ops[0]->getType()});

  defm NAME : Binary<NAME, I.Value, I.Value, I.Value, "_vv", I.Name, I.BoolValue, [-
1], [-1, 0]>;
```

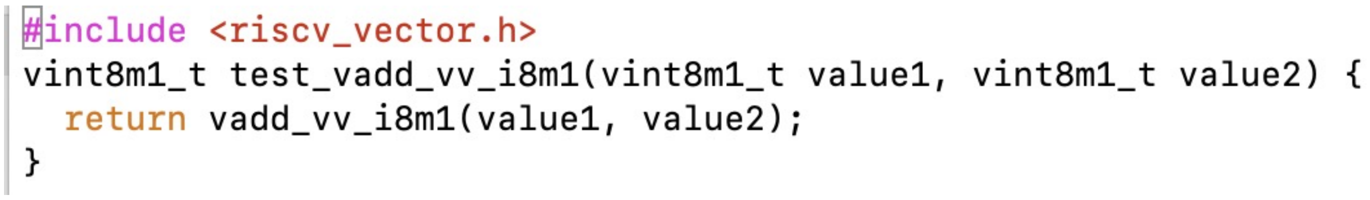
- **如果在 IntrinsicsRISCV.h 中找不到相关的函数名字,则在 IntrinsicsRISCV.td 中添加 intrinsic 的支持**

```
1. defm vadd : Binary_int_vv_vx;
2. multiclass Binary_int_vv_vx {
  defm NAME : Binary_int_vv;
  defm NAME : Binary_int_vx;
  }
3. multiclass Binary_int_vv {
  def "int_riscv_" # NAME # "_vv" : V_VV;
  def "int_riscv_" # NAME # "_vv_mask" : V_VV_mask;
}
4. class V_VV : Intrinsic<[llvm_anyvector_ty],
                          [LLVMMatchType<0>, LLVMMatchType<0>],
                          [IntrNoMem]>;

5. class V_VV_mask : Intrinsic<[llvm_anyvector_ty],
                               [llvm_anyvector_ty, LLVMMatchType<0>,
                                LLVMMatchType<0>, LLVMMatchType<0>],

                               [IntrNoMem]>;
```

其中值得注意的是 LLVMMatchType<0>, 是和第 0 个也就是返回值的类型是一样的，根据 intrinsic 定义中的参数形式，调整 CGM.getIntrinsic 中的传参。

4 测试 intrisic 函数生成 IR 的，参考 clang/test/CodeGen/riscv-vector-intrinsics 目录下的测试用例添加新的测试

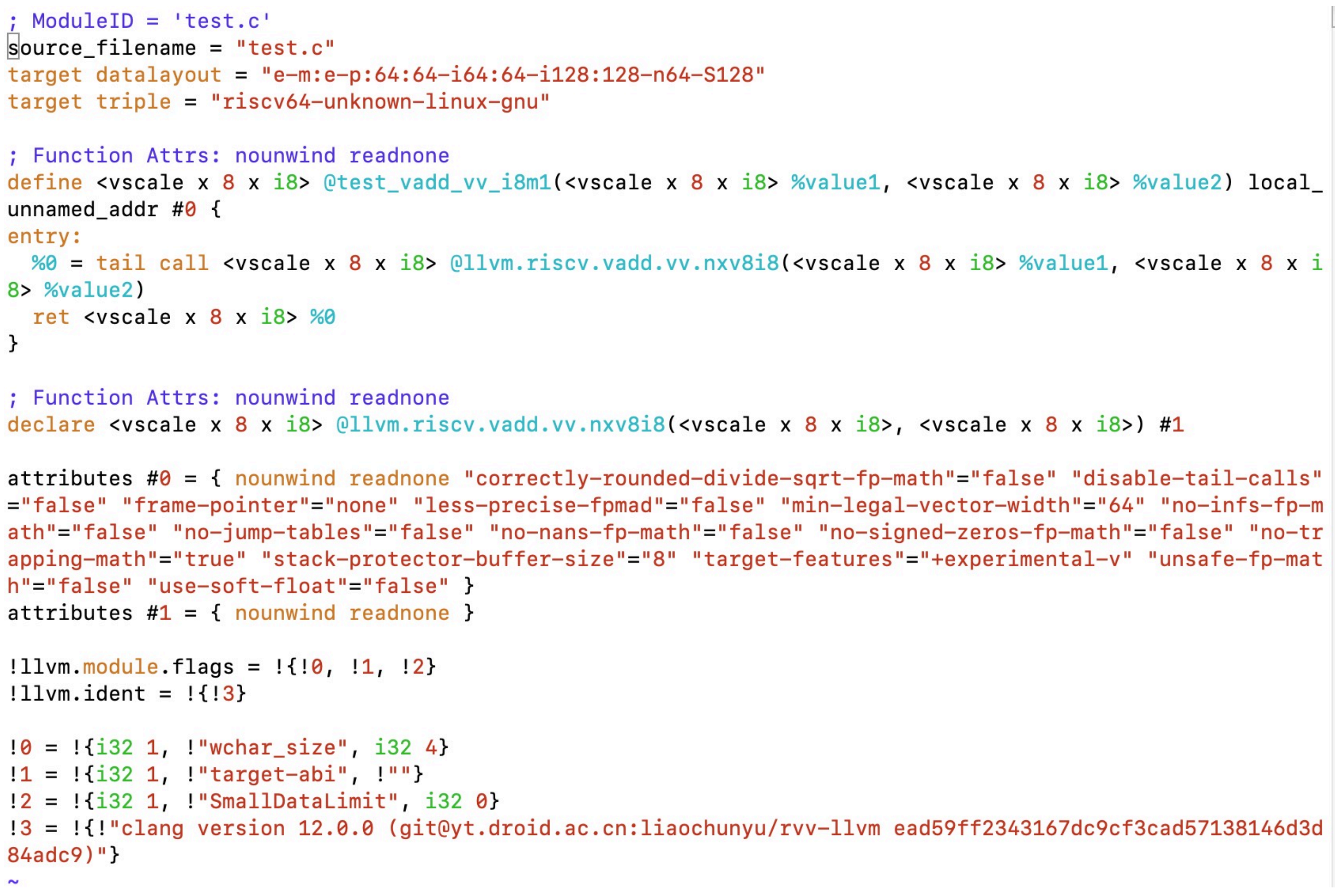
新的测试用例 test.c：



```c
#include <riscv_vector.h>
vint8m1_t test_vadd_vv_i8m1(vint8m1_t value1, vint8m1_t value2) {
  return vadd_vv_i8m1(value1, value2);
}
```

测试生成的 IR，即 test.ll 文件

```
~/rvv-llvm/build/bin/clang -cc1 -internal-isystem ~/rvv-llvm/build/lib/clang/12.0.0/include  -triple riscv64-unknown-linux-gnu -target-feature +experimental-v -S -emit-llvm  -O3 test.c
```



```
; ModuleID = 'test.c'
source_filename = "test.c"
target datalayout = "e-m:e-p:64:64-i64:64-i128:128-n64-S128"
target triple = "riscv64-unknown-linux-gnu"

; Function Attrs: nounwind readnone
define <vscale x 8 x i8> @test_vadd_vv_i8m1(<vscale x 8 x i8> %value1, <vscale x 8 x i8> %value2) local_unnamed_addr #0 {
entry:
  %0 = tail call <vscale x 8 x i8> @llvm.riscv.vadd.vv.nxv8i8(<vscale x 8 x i8> %value1, <vscale x 8 x i8> %value2)
  ret <vscale x 8 x i8> %0
}

; Function Attrs: nounwind readnone
declare <vscale x 8 x i8> @llvm.riscv.vadd.vv.nxv8i8(<vscale x 8 x i8>, <vscale x 8 x i8>) #1

attributes #0 = { nounwind readnone "correctly-rounded-divide-sqrt-fp-math"="false" "disable-tail-calls"="false" "frame-pointer"="none" "less-precise-fpmad"="false" "min-legal-vector-width"="64" "no-infs-fp-math"="false" "no-jump-tables"="false" "no-nans-fp-math"="false" "no-signed-zeros-fp-math"="false" "no-trapping-math"="true" "stack-protector-buffer-size"="8" "target-features"="+experimental-v" "unsafe-fp-math"="false" "use-soft-float"="false" }
attributes #1 = { nounwind readnone }

!llvm.module.flags = !{!0, !1, !2}
!llvm.ident = !{!3}

!0 = !{i32 1, !"wchar_size", i32 4}
!1 = !{i32 1, !"target-abi", !""}
!2 = !{i32 1, !"SmallDataLimit", i32 0}
!3 = !{!"clang version 12.0.0 (git@yt.droid.ac.cn:liaochunyu/rvv-llvm ead59ff2343167dc9cf3cad57138146d3d84adc9)"}
~
```

Tablegen 的一些参考：

https://llvm.org/docs/TableGen/

https://llvm.org/docs/TableGen/ProgRef.html