在LLVM中实现RISC-V用户自定义指令支持 ---以玄铁C910为例

报告人：陈影
中科院软件所智能软件中心 PLCT实验室 邢明杰，王鹏，张尹

2020/7/5

CONTENT

目录

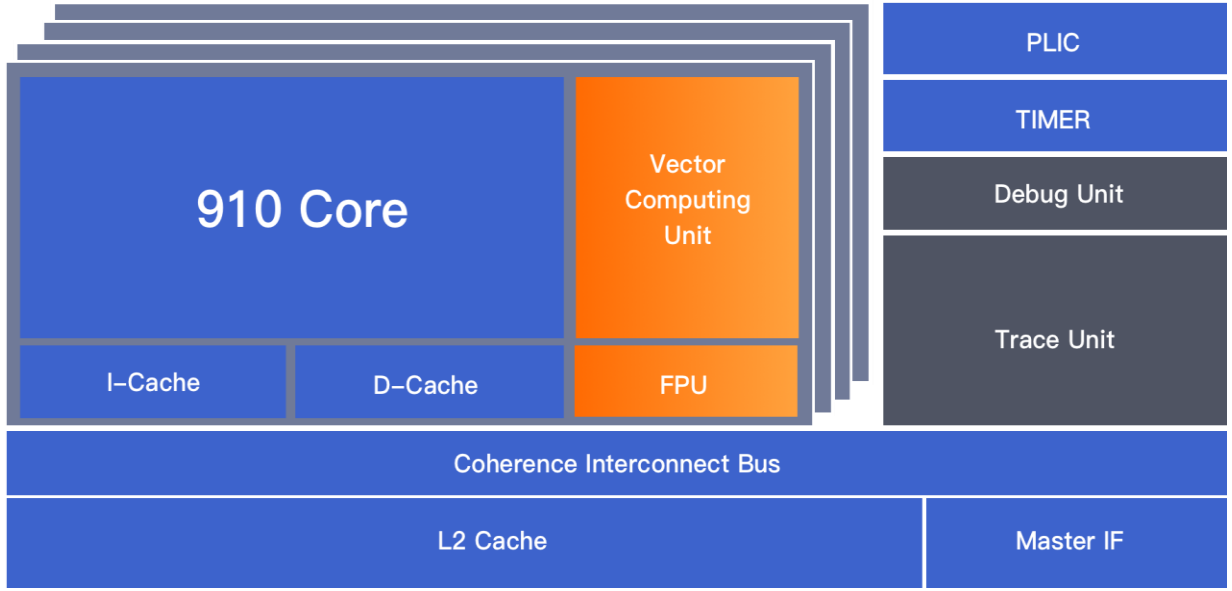
1. 背景介绍

项目整体概述

- 以玄铁C910的扩展指令集为例说明在RISC-V后端添加新的扩展指令的方法。
- 已经完成了对玄铁C910扩展指令的汇编器支持，接下来可以进一步考虑以LLVM为框架实现对更加复杂的指令集架构的支持。
- 项目开源地址：https://github.com/isrc-cas/c910-llvm

1. 背景介绍

玄铁C910简介



- 高性能64位RISC-V架构多核处理器
- 指令集：RISC-V RV64GC/RV 64GCV
- AI增强的向量计算引擎

玄铁C910官网介绍：https://www.t-head.cn/product/c910?spm=a2ouz.12987056.0.0.2cb96245CRx1Tp&lang=zh

1. 背景介绍

玄铁C910指令集架构

基本模块

RV64GCV指令
- RV64I 整型指令集
- RV64M 乘除法指令集
- RV64A 原子指令集
- RV64F 单精度浮点指令集
- RV64D 双精度浮点指令集
- RVC 压缩指令集
- RVV矢量指令集

扩展模块

扩展指令集
- 位操作指令子集
- 同步指令子集
- 算术运算指令子集
- CACHE 指令子集
- 存储指令子集

1. 背景介绍

玄铁C910指令集架构

基本模块

RV64GCV指令
- RV64I 整型指令集
- RV64M 乘除法指令集
- RV64A 原子指令集
- RV64F 单精度浮点指令集
- RV64D 双精度浮点指令集
- RVC 压缩指令集
- RVV矢量指令集

扩展模块

扩展指令集
- 位操作指令子集
- 同步指令子集
- 算术运算指令子集
- CACHE 指令子集
- 存储指令子集

1. 背景介绍

玄铁C910扩展指令

位操作指令子集：EXT EXTU FF0 FF1 REV REVW TST

同步指令子集：SYNC SYNC.I SYNC.IS SYNC.S

算术运算指令子集：MULA MULAH MULAW ADDSL MULS MULSH MULSW MVEQZ MVNEZ SRRI SRRIW

CACHE 指令子集：DCACHE.CIALL DCACHE.CIPA DCACHE.CISW DCACHE.CIVA DCACHE.CPA DCACHE.CPAL1 DCACHE.CVA DCACHE.CVAL1 DCACHE.IPA DCHCHE.ISW DCACHE.IVA DCACHE.IALL ICACHE.IALL ICACHE.IALLS ICACHE.IPA ICACHE.IVA L2CACHE.CALL L2CACHE.CIALL L2CACHE.IALL

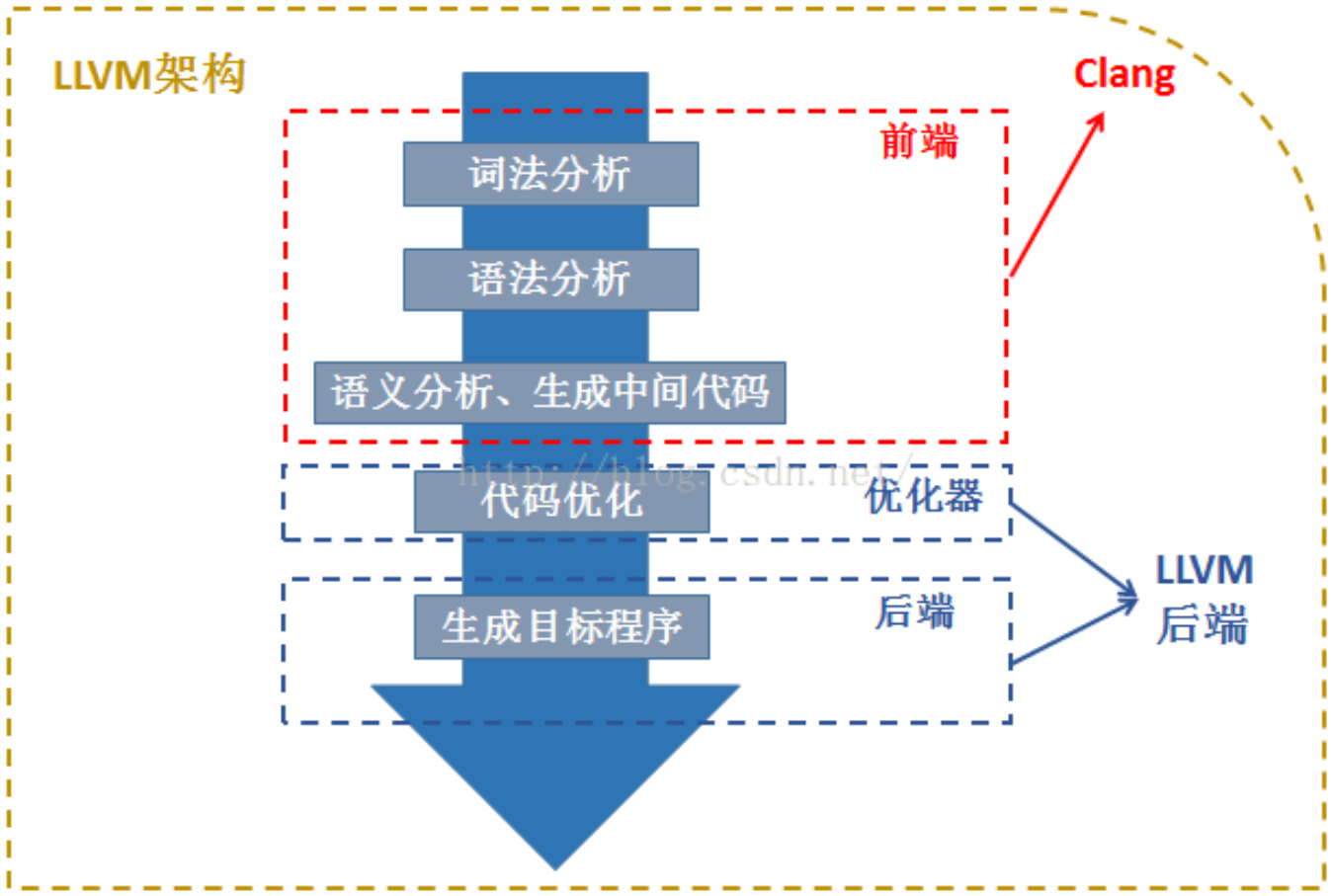
存储指令子集：LRB LRH LRW LRD LRBU LRHU LRWU LURB LURH LURW LURD LURBU LURHU LURWU LDIA LDIB LDD LWD LWUD
SRB SRH SRW SRD SURB SURH SURW SURD
FSRD FSRW FSUSR FSURW FLRD FLRW FLURD FLURW
LBIA LBIB LHIA LHIB LWIA LWIB LBUIA LBUIB LHUIA LHUIB LWUIA LWUIB
SBIA SBIB SHIA SHIB SWIA SWIB SDIA SDIB SDD SWD

CONTENT

目录

2. 汇编实现

Clang/LLVM编译器



.td文件

在LLVM的RISCV后端目录下新增了相应的目标描述文件，并在汇编/反汇编相关子目录AsmParser/Disassembler下面实现对特定操作数的处理。

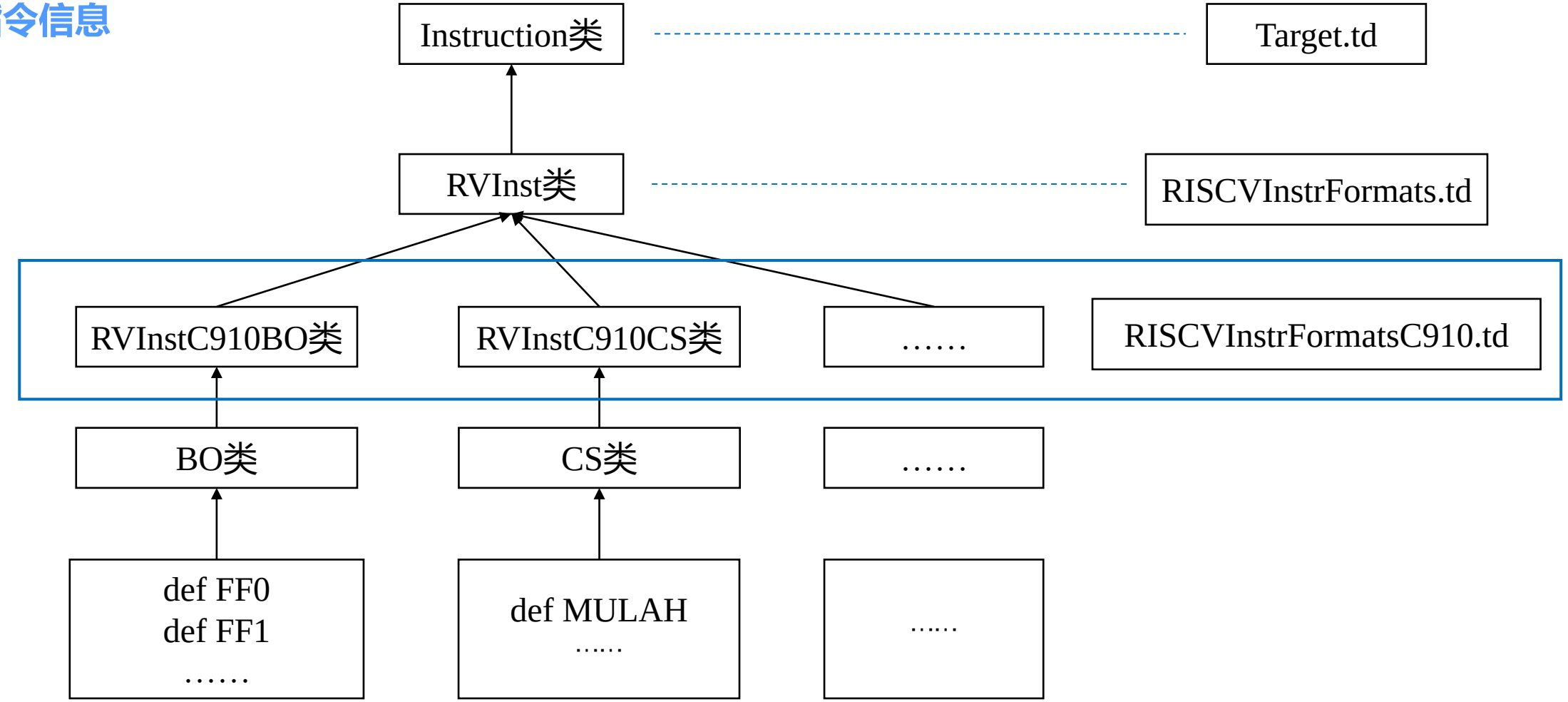
2. 汇编实现

TableGen语言（.td）

- 实现RISC-V用户自定义指令支持，主要是通过TableGen语言描述后端的寄存器信息和指令信息来实现的。
- LLVM使用TableGen来描述特定目标的信息记录，包括编译器后端的很多特征。
- TableGen语法类似于C++的template，用classes和definitions描述后端特征。

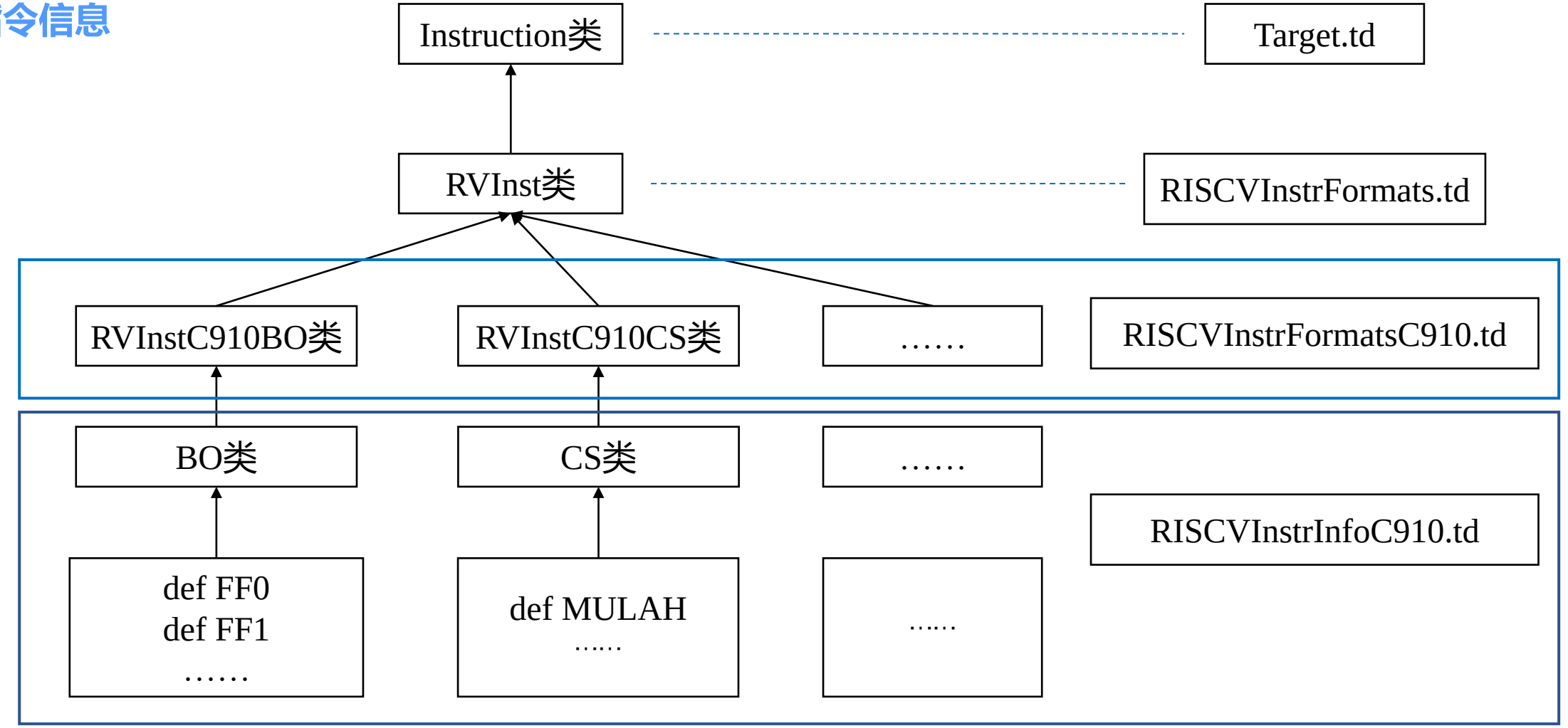
2. 汇编实现

描述指令信息



2. 汇编实现

描述指令信息



2. 汇编实现

描述指令信息---一个例子

共100条扩展指令，其具体的指令形式描述在[玄铁C910指令集手册]中，比如

6.3.2 MULA——乘累加指令

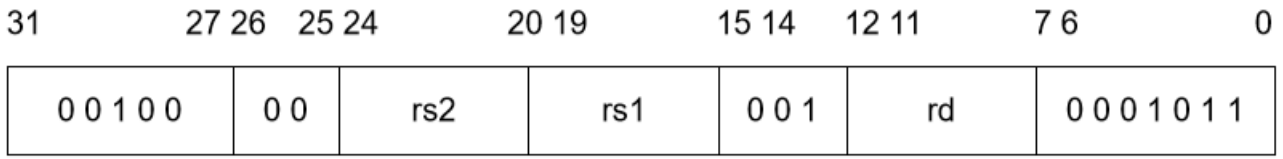
语法: mula rd, rs1, rs2

操作: rd ← rd+ (rs1 * rs2)[63:0]

执行权限: M mode/S mode/U mode

异常: 非法指令异常

指令格式:



2. 汇编实现

描述指令信息---一个例子

lib/Target/RISCV/RISCVInstrInfoC910.td

- **描述指令信息：**
  - 定义指令形式（Format）
  - 定义指令模板（Template）
  - 定义指令（Instruction）

```
44  def MULA : Instruction {
45      bits<32> Inst;
46      bits<32> SoftFail = 0;
47      bits<5> rs2;
48      bits<5> rs1;
49      bits<5> rd;
50      let Namespace = "RISCV";
51      let hasSideEffects = 0;
52      let mayLoad = 0;
53      let mayStore = 0;
54      let Size = 4;
55      let Inst{31-27} = 0b00100; //funct5
56      let Inst{26-25} = 0b00; //funct2
57      let Inst{24-20} = rs2;
58      let Inst{19-15} = rs1;
59      let Inst{14-12} = 0b001; //funct3
60      let Inst{11-7} = rd;
61      let Inst{6-0} = 0b0001011; //opcode
62      dag OutOperandList = (outs GPR:$rd);
63      dag InOperandList = (ins GPR:$rs1, GPR:$rs2);
64      let AsmString = "mula\t$rd, $rs1, $rs2";
65  }
```

2. 汇编实现

描述指令信息---一个例子

lib/Target/RISCV/RISCVInstrInfoC910.td

```
70   let hasSideEffects = 0, mayLoad = 0, mayStore = 0 in
71   class CS_1<bits<5> funct5, bits<2> funct2, string opcodestr>
72       : RVInstC910CS_1<funct5, funct2, OPC_CUSTOM0, (outs GPR:$rd), (ins GPR:$rs1, GPR:$rs2),
73                 opcodestr, "$rd, $rs1, $rs2">;
74
75   def MULAH : CS_1<0b00101,00,"mulah">;
76   def MULAW : CS_1<0b00100,10,"mulaw">;
77   def MULS : CS_1<0b00100,01,"muls">;
```

```
4   //===----------------------------------------------------------------------===//
5   // Instruction Formats
6   //===----------------------------------------------------------------------===//
7   include "RISCVInstrFormatsC910.td"
```

lib/Target/RISCV/RISCVInstrFormatsC910.td

```
71  class RVInstC910CS_1<bits<5> funct5, bits<2> funct2, RISCVOpcode opcode,
72                 dag outs, dag ins, string opcodestr, string argstr>
73      : RVInst<outs, ins, opcodestr, argstr, [], InstFormatOther> {
74    bits<5> rs1;
75    bits<5> rs2;
76    bits<5> rd;
77
78    let Inst{31-27} = funct5;
79    let Inst{26-25} = funct2;
80    let Inst{24-20} = rs2;
81    let Inst{19-15} = rs1;
82    let Inst{14-12} = 0b001;
83    let Inst{11-7}  = rd;
84    let Opcode = opcode.Value;
85  }
```

2. 汇编实现

寄存器扩展

lib/Target/RISCV/RISCVSystemOperands.td

```
79  //===-------------------------
80  //  C910 User extended CSRs
81  //===-------------------------
82
83  def FXCR : SysReg<"fxcr", 0x800>;


168 //===-------------------------
169 //  C910 Supervisor extended CSRs
170 //===-------------------------
171 def SXSTATUS : SysReg<"sxstatus", 0x5C0>;
172 def SHCR : SysReg<"shcr", 0x5C1>;
173 def SCER2 : SysReg<"scer2", 0x5C2>;
```

```
350 //===-----------------------------------------------
351 // C910 Machine Extended CSRs
352 //===-----------------------------------------------
353 def MXSTATUS : SysReg<"mxstatus", 0x7C0>;
354 def MHCR : SysReg<"mhcr", 0x7C1>;
355 def MCOR : SysReg<"mcor", 0x7C2>;
356 def MCCR2 : SysReg<"mccr2", 0x7C3>;
357 def MCER2 : SysReg<"mcer2", 0x7C4>;
358 def MHINT : SysReg<"mhint", 0x7C5>;
359 def MRMR : SysReg<"mrmr", 0x7C6>;
360 def MRVBR : SysReg<"mrvbr", 0x7C7>;
362 //===-----------------------------------------------
363 // C910 Machine Cache Access Extended CSRs
364 //===-----------------------------------------------
365
366 def MCINS : SysReg<"mcins", 0x7D2>;
367 def MCINDEX : SysReg<"mcindex", 0x7D3>;
368 def MCDATA0 : SysReg<"mcdata0", 0x7D4>;
369 def MCDATA1 : SysReg<"mcdata1", 0x7D5>;
371 //===-----------------------------------------------
372 // C910 Machine Processor Extended CSRs
373 //===-----------------------------------------------
374 def MCPUID : SysReg<"mcupid", 0xFC0>;
```

对机器模式、超级用户模式、用户模式这三种模式下的CSRs扩展

CONTENT

目录

dcache_ciall
dcache_ciall
dcache_ciall
dcache_ciall
[0x0b,0x00,0xb0,0x00]

3. 添加一个汇编器选项

添加llvm-mc选项

- llvm-mc工具可以看作是通常意义下的汇编器和反汇编器，对标gcc下的as和dis;
- 使用--show-encoding 选项打印特定指令的汇编器代码;
  - $ echo “dcache_ciall” | llvm-mc –mcpu=c910 –show-encoding
  - # encoding: [0x0b,0x00,0xb0,0x00]
- 提供反汇编功能，使--show-inst 选项显示反汇编或汇编指令的MCInst实例。
  - $ echo “0x0b,0x00,0xb0,0x00” | llvm-mc –disassemble –mcpu=c910 –show-inst
  - dcache_ciall # <MCInst #350 DCACHE_CIALL> …

3. 添加一个汇编器选项

添加llvm-mc选项

lib/Target/RISCV/RISCV.td：对 -mcpu=c910 在RISCV目录下进行定义

```
86  //===----------------------------------------------------------------------===//
87  // RISC-V processors supported.
88  //===----------------------------------------------------------------------===//
89
90  def : ProcessorModel<"c910", NoSchedModel, [Feature64Bit,FeatureStdExtA,FeatureStdExtC,
91                                               FeatureStdExtM,FeatureStdExtF,FeatureStdExtD,FeatureC910]>;
92
93  def : ProcessorModel<"generic-rv32", NoSchedModel, []>;
94
95  def : ProcessorModel<"generic-rv64", NoSchedModel, [Feature64Bit]>;
```

```
46  def FeatureC910
47      : SubtargetFeature<"c910", "HasC910", "true",
48                         "Implement C910">;
49  def HasC910 : Predicate<"Subtarget->hasC910()">,
50                           AssemblerPredicate<"FeatureC910">;
```

3. 添加一个汇编器选项

添加llvm-mc选项

lib/Target/RISCV/RISCVSubtarget.h

```
31  class RISCVSubtarget : public RISCVGenSubtargetInfo {
32    virtual void anchor();
33    bool HasStdExtM = false;
34    bool HasStdExtA = false;
35    bool HasStdExtF = false;
36    bool HasStdExtD = false;
37    bool HasStdExtC = false;
38    bool HasC910 = false;
39    bool HasRV64 = false;
40    bool IsRV32E = false;
41    bool EnableLinkerRelax = false;
42    unsigned XLen = 32;
43    MVT XLenVT = MVT::i32;
44    RISCVABI::ABI TargetABI = RISCVABI::ABI_Unknown;
45    RISCVFrameLowering FrameLowering;
```

```
82    bool hasStdExtD() const { return HasStdExtD; }
83    bool hasStdExtC() const { return HasStdExtC; }
84    bool hasC910() const { return HasC910; }
85    bool is64Bit() const { return HasRV64; }
86    bool isRV32E() const { return IsRV32E; }
87    bool enableLinkerRelax() const { return EnableLinkerRelax; }
```

CONTENT

目录

4. 汇编器测试

LLVM 测试框架

LLVM测试框架主要包括两类：**回归测试**和**整体程序测试。**

- 回归测试
  - 这些用例应在每次提交前运行通过。
  - 回归测试是测试LLVM特定功能或触发LLVM特定bug的一小段代码。
  - 这些测试用例由LLVM lit测试工具驱动，用例代码位于llvm/test路径下。
- 整体程序测试/测试套件（LLVM test suite）
  - 测试套件包含完整程序，程序代码通常用高级语言如C/C++，可以编译链接进某个可执行程序。
  - 这些程序由用户特定编译器编译，然后执行捕获程序输出和时序信息。
  - 这些输出和参考输出比较以确保程序编译正确。

4. 汇编器测试

LLVM 测试用例

test/MC/RISCV/c910-valid.s

```
# RUN: llvm-mc %s -triple=riscv64 -mcpu=c910 -riscv-no-aliases -show-encoding \
# RUN:     | FileCheck -check-prefixes=CHECK-ASM,CHECK-ASM-AND-OBJ %s


# CHECK-ASM-AND-OBJ: mula a1, a2, a3
# CHECK-ASM: encoding: [0x8b,0x15,0xd6,0x20]
mula a1, a2, a3
```

```
$ ./bin/llvm-lit -v ../llvm/test/MC/RISCV/c910-valid.s
```

5. 参考文献

http://llvm.org/docs/ProgrammersManual.html

https://www.design-reuse.com/articles/46237/extending-risc-v-isa-with-a-custom-instruction-set-extension.html

https://github.com/isrc-cas/rvv-llvm/tree/rvv-iscas

https://blog.csdn.net/wuhui_gdnt/article/details/62218211

https://zhuanlan.zhihu.com/p/54536799

谢 谢

欢迎加入我们，一起做些有意思的事情☺

https://github.com/isrc-cas/PLCT-Weekly/blob/master/Jobs.md

2020/07/05