中国科学院软件研究所
Institute of Software Chinese Academy of Sciences

智能软件研究中心
Intelligent Software Research Center

软件所智能软件中心PLCT实验室 王鹏 实习生

2020//02/19

目 录

• 01 添加llvm-mc命令行选项

llvm-mc 的主要任务是汇编一个.s文件（通过-assemble命令），反汇编字节串（-disassemble）以及显示指令及其内部表示的编码（-show-encoding和-show -inst）。

-mcpu=<cpu-name>

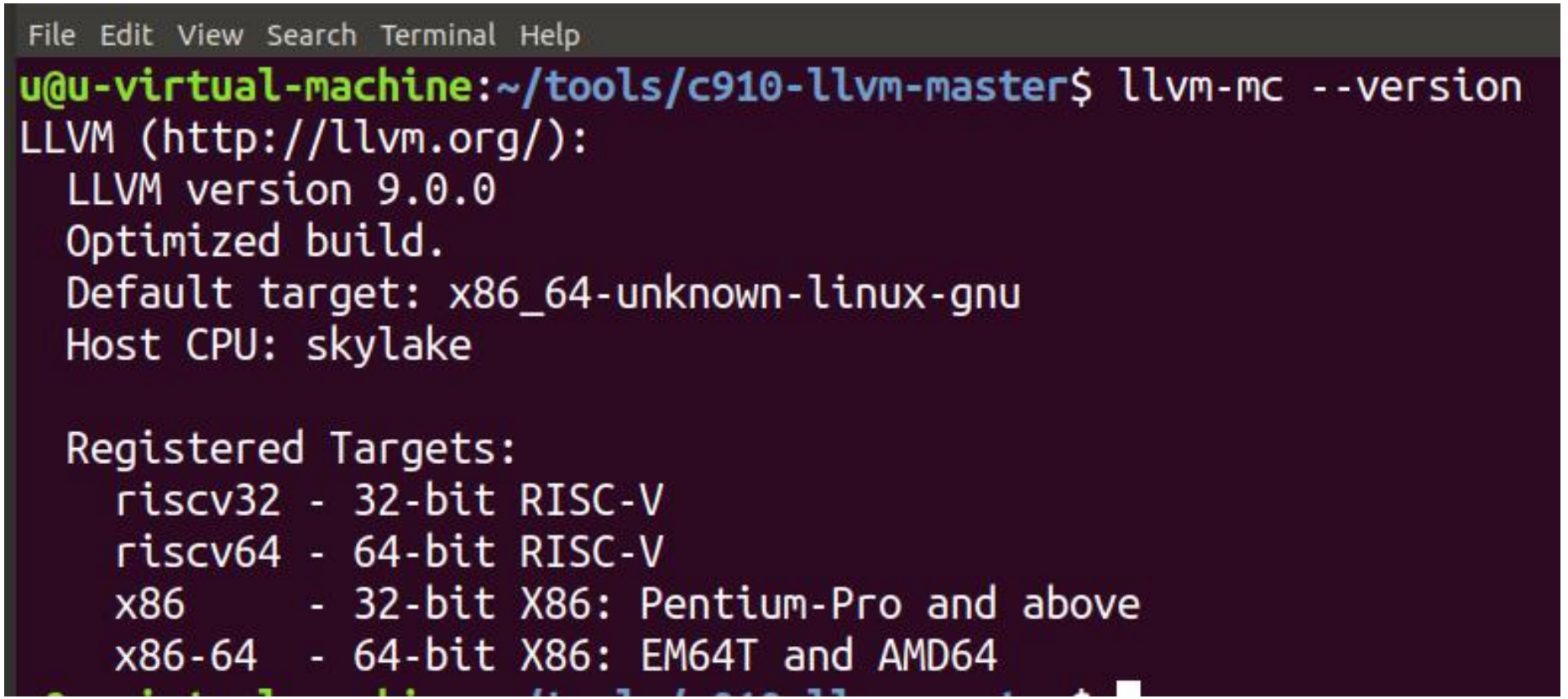
-triple=<string>　目标三重拆解　(e.g. i686-apple-darwin9)

-arch=<string>　- Target arch to assemble for

llvm-mc hello.S -filetype=obj -o hello.o 汇编到目标文件转化

llvm-mc -assemble -show-encoding -arch=riscv64 -mattr=+f hello_riscv.s 显示汇编的编码

echo "0x10 0xf1 0x10 0xe7" | llvm-mc -disassemble -arch=arm -mattr=+hwdiv-arm 反汇编ARM的sdiv指令


File Edit View Search Terminal Help
u@u-virtual-machine:~/tools/c910-llvm-master$ llvm-mc --version
LLVM (http://llvm.org/):
LLVM version 9.0.0
Optimized build.
Default target: x86_64-unknown-linux-gnu
Host CPU: skylake
Registered Targets:
riscv32 - 32-bit RISC-V
riscv64 - 64-bit RISC-V
x86 - 32-bit X86: Pentium-Pro and above
x86-64 - 64-bit X86: EM64T and AMD64

```
  --preserve-comments                                - Preserve Comments in outputted assembly
  --print-imm-hex                                    - Prefer hex format for immediate values
  --relax-relocations                                - Emit R_X86_64_GOTPCRELX instead of R_X86_64_GOTPCREL
  --save-temp-labels                                 - Don't discard temporary labels
  --show-encoding                                    - Show instruction encodings
  --show-inst                                        - Show internal instruction representation
  --show-inst-operands                               - Show instructions operands as parsed
  --split-dwarf-file=<filename>                      - DWO output filename
  --triple=<string>                                  - Target triple to assemble for, see -version for available targets

Generic Options:

  --help                                             - Display available options (--help-hidden for more)
  --help-list                                        - Display list of available options (--help-list-hidden for more)
  --version                                          - Display the version of this program
```

在llvm-mc目录下grep -r split-dwarf-file

在llvm-mc目录下出现了关键字split-dwarf-file

c910-llvm-master/tmp/llvm/tools/llvm-mc目录下

CMakeLists.txt

Disassembler.cpp

Disassembler.h

LLVMBuild.txt

llvm-mc.cpp描述了split-dwarf-file

等llvm-mc --help中general option的信息

```
u@u-virtual-machine:~$ echo "0xCD 0x21" | llvm-mc --disassemble -triple=riscv32 -mattr=help
Available CPUs for this target:

  generic-rv32 - Select the generic-rv32 processor.
  generic-rv64 - Select the generic-rv64 processor.

Available features for this target:

  64bit - Implements RV64.
  a     - 'A' (Atomic Instructions).
  c     - 'C' (Compressed Instructions).
  d     - 'D' (Double-Precision Floating-Point).
  e     - Implements RV32E (provides 16 rather than 32 GPRs).
  f     - 'F' (Single-Precision Floating-Point).
  m     - 'M' (Integer Multiplication and Division).
  relax - Enable Linker relaxation..

Use +feature to enable a feature, or -feature to disable it.
For example, llc -mcpu=mycpu -mattr=+feature1,-feature2
        .text
<stdin>:1:1: warning: invalid instruction encoding
0xCD 0x21
^
```

C910的test/MC/RISCV下面有106个文件。

u@u-virtual-machine:~/tools/c910-llvm-master/test/MC/RISCV$ ls | wc -w

106

u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/test/MC/RISCV$ ls | wc -w

115

肯定有C910没有的测试文件，下面找到linux命令

```
diff -r directory1 directoru2
    RISCV/numeric-reg-names-d.s
    RISCV/rv64c-hints-valid.s
:   RISCV/rvc-hints-valid.s
    RISCV/rva-aliases-valid.s
    RISCV/rva-aliases-invalid.s
    RISCV/numeric-reg-names.s
    RISCV/rv64a-aliases-valid.s
    RISCV/rvc-hints-invalid.s
    RISCV/numeric-reg-names-f.s
```

• 1.1 为llvm-mc添加rvc-hints命令行选项

对C910和llvm官方源代码进行对比阅读lib/Target/RISCV/RISCV.td，查找对比。

在lib/Target/RISCV/RISCV.td(C910有)中主要描述:

1 RISC-V subtarget features and instruction predicates.

其中包括FeatureStdExtM，FeatureStdExtA，FeatureStdExtF，FeatureStdExtD，FeatureStdExtC，FeatureRV32E和FeatureRelax（Enable Linker relaxation)

2 Named operands for CSR instructions

3 Registers, calling conventions, instruction descriptions.

4 RISC-V processors supported.(ProcessorModel包括rv32,rv64)

5 Define the RISC-V target.包括 def

RISCVInstrInfo : InstrInfo

RISCVAsmParser : AsmParser（汇编程序解析由 AsmParser 类提供）

RISCVAsmWriter : AsmWriter（功能Printing LLVM IR as an assembly file）

RISCV : Target

C910没有且为了支持rvc-hints的属性需要添加

```
def FeatureRVCHints
    : SubtargetFeature<"rvc-hints", "EnableRVCHintInstrs",
"true",
                       "Enable RVC Hint Instructions.">;
def HasRVCHints : Predicate<"Subtarget-
>enableRVCHintInstrs()">,
                            AssemblerPredicate<"FeatureRVCHints">;
```

Predicate的意思是谓词

predicate是一种布尔值函数，所谓布尔值函数，即自变量x映射到因变量y，y的取值是true或者false

https://blog.csdn.net/wuhui_gdnt/article/details/68923901

值不为0的汇编程序谓词（AssemblerPredicate）将接收在调用它时使用的参数，并且可能传递出与这些参数匹配的参数

```
include "RISCVRegisterBanks.td"


def : ProcessorModel<"generic-rv32", NoSchedModel,
[FeatureRVCHints]>;
def : ProcessorModel<"generic-rv64", NoSchedModel,
[Feature64Bit,
                     FeatureRVCHints]>;
```

```
//=-- RISCVRegisterBank.td - Describe the RISCV Banks --------*- tablegen -*-=//
//
// Part of the LLVM Project, under the Apache License v2.0 with LLVM Exceptions.
// See https://llvm.org/LICENSE.txt for license information.
// SPDX-License-Identifier: Apache-2.0 WITH LLVM-exception
//
//===----------------------------------------------------------------------===//
//
//
//===----------------------------------------------------------------------===//

/// General Purpose Registers: X.
def GPRRegBank : RegisterBank<"GPRB", [GPR]>;
```

RegisterBank（模块）

1 寄存器库用于汇编语言程序员使用的可编程寄存器。

给定索引的情况下读写数据的端口

主存储器和缓存具有相似的接口

用于读取和写入数据

2 寄存器号用作寄存器组中的索引。存储器地址用作主存储器中的索引。内存地址的一部分用作高速缓存中的索引。

3 通过一组寄存器以及用于读和写端口的额外电路实现的寄存器组。（硬件?）

lib/Target/RISCV/AsmParser/RISCVAsmParser.cpp汇编解析器中定义了rvc-hints的立即数相关信息

lib/Target/RISCV/Disassembler/RISCVDisassembler.cpp定义了rvc-hints的解码信息（DecodeStatus)，其中包括rd，rs1，rs2和imm的一些关系，它们关系的定义主要在

The RISC-V Instruction Set Manual Volume I: Unprivileged ISA Document Version 20191213

最初在"llvm/include/llvm/Target/TargetSchedule.td"中定义

```
def NoSchedModel : SchedMachineModel {
  let NoModel = 1;
}
```

对所有的目标机器家族，描述的第一个处理器总是NoSchedModel，NoSchedModel等价于空指针

lib/Target/RISCV/RISCVSubtarget.h中

```
bool HasStdExtM = false;
bool HasStdExtA = false;
bool HasStdExtF = false;
bool HasStdExtD = false;
bool HasStdExtC = false;
bool HasRV64 = false;
bool IsRV32E = false;
bool EnableLinkerRelax = false;
bool EnableRVCHintInstrs = false;
```

```
bool hasStdExtM() const { return HasStdExtM; }
bool hasStdExtA() const { return HasStdExtA; }
bool hasStdExtF() const { return HasStdExtF; }
bool hasStdExtD() const { return HasStdExtD; }
bool hasStdExtC() const { return HasStdExtC; }
bool is64Bit() const { return HasRV64; }
bool isRV32E() const { return IsRV32E; }
bool enableLinkerRelax() const { return EnableLinkerRelax; }
bool enableRVCHintInstrs() const { return
EnableRVCHintInstrs; }
```

test/MC/RISCV/rv64c-hints-valid.s 支持

llvm-mc -triple=riscv64 -mattr=+c -riscv-no-aliases -show-encoding

llvm-mc -filetype=obj -triple=riscv64 -mattr=+c

test/MC/RISCV/rvc-hints-invalid.s 支持

llvm-mc -triple=riscv32 -mattr=+c

llvm-mc -triple=riscv64 -mattr=+c

test/MC/RISCV/rvc-hints-valid.s 支持

llvm-mc -triple=riscv32 -mattr=+c -riscv-no-aliases -show-encoding

llvm-mc -triple riscv64 -mattr=+c -riscv-no-aliases -show-encoding

llvm-mc -filetype=obj -triple=riscv32 -mattr=+c

llvm-mc -filetype=obj -triple=riscv64 -mattr=+c

- 02 RISCV手册中rvc-hints

The RISC-V Instruction Set Manual
Volume I: Unprivileged ISA
Document Version 20191213

Editors: Andrew Waterman[1], Krste Asanović[1,2]
[1]SiFive Inc.,
[2]CS Division, EECS Department, University of California, Berkeley
andrew@sifive.com, krste@berkeley.edu
December 13, 2019

Chapter 2 RV32I

2.9 HINT Instructions

RV32I为HINT指令保留了很大的编码空间，通常用于将性能提示传达给微体系结构。提示被编码为rd = x0的整数计算指令。因此，与NOP指令一样，HINT不会更改任何体系结构上可见的状态，除非会提高PC和任何适用的性能计数器。始终允许实现忽略编码的提示。

表2.3列出了所有RV32I HINT代码点。 91%的HINT空间是为标准HINT保留的，但目前尚未定义。其余的提示空间保留给自定义提示：在此子空间中将永远不会定义任何标准的提示。

Instruction	Constraints	Code Points	Purpose
LUI	*rd*=x0	2^{20}	*Reserved for future standard use*
AUIPC	*rd*=x0	2^{20}	
ADDI	*rd*=x0, and either *rs1*≠x0 or *imm*≠0	$2^{17}-1$	
ANDI	*rd*=x0	2^{17}	
ORI	*rd*=x0	2^{17}	
XORI	*rd*=x0	2^{17}	
ADD	*rd*=x0	2^{10}	
SUB	*rd*=x0	2^{10}	
AND	*rd*=x0	2^{10}	
OR	*rd*=x0	2^{10}	
XOR	*rd*=x0	2^{10}	
SLL	*rd*=x0	2^{10}	
SRL	*rd*=x0	2^{10}	
SRA	*rd*=x0	2^{10}	
FENCE	*pred*=0 or *succ*=0	$2^{5}-1$	

SLTI	*rd*=x0	2^{17}	*Reserved for custom use*
SLTIU	*rd*=x0	2^{17}	
SLLI	*rd*=x0	2^{10}	
SRLI	*rd*=x0	2^{10}	
SRAI	*rd*=x0	2^{10}	
SLT	*rd*=x0	2^{10}	
SLTU	*rd*=x0	2^{10}	

Table 2.3: RV32I HINT instructions.

代码点(code point)：是指编码字符集中，字符所对应的数字。有效范围从U+0000到U+10FFFF。其中U+0000到U+FFFF为基本字符，U+10000到U+10FFFF为增补字符。

Chapter 5 RV64I

5.4 HINT Instructions

RV32I中所有属于微体系结构HINT的指令（请参见2.9节）在RV64I中也均为HINT。 RV64I中的其他计算指令扩展了标准和自定义的HINT编码空间。

表5.1列出了所有RV64I HINT代码点。 91%的HINT空间是为标准HINT保留的，但目前尚未定义。其余的提示空间保留给自定义提示：在此子空间中将永远不会定义任何标准的提示。

Instruction	Constraints	Code Points	Purpose
LUI	*rd*=x0	2^{20}	*Reserved for future standard use*
AUIPC	*rd*=x0	2^{20}	
ADDI	*rd*=x0, and either *rs1*≠x0 or *imm*≠0	$2^{17}-1$	
ANDI	*rd*=x0	2^{17}	
ORI	*rd*=x0	2^{17}	
XORI	*rd*=x0	2^{17}	
ADDIW	*rd*=x0	2^{17}	
ADD	*rd*=x0	2^{10}	
SUB	*rd*=x0	2^{10}	
AND	*rd*=x0	2^{10}	
OR	*rd*=x0	2^{10}	
XOR	*rd*=x0	2^{10}	
SLL	*rd*=x0	2^{10}	
SRL	*rd*=x0	2^{10}	
SRA	*rd*=x0	2^{10}	
ADDW	*rd*=x0	2^{10}	
SUBW	*rd*=x0	2^{10}	
SLLW	*rd*=x0	2^{10}	
SRLW	*rd*=x0	2^{10}	
SRAW	*rd*=x0	2^{10}	
FENCE	*pred*=0 or *succ*=0	$2^{5}-1$	

SLTI	*rd*=x0	2^{17}	*Reserved for custom use*
SLTIU	*rd*=x0	2^{17}	
SLLI	*rd*=x0	2^{11}	
SRLI	*rd*=x0	2^{11}	
SRAI	*rd*=x0	2^{11}	
SLLIW	*rd*=x0	2^{10}	
SRLIW	*rd*=x0	2^{10}	
SRAIW	*rd*=x0	2^{10}	
SLT	*rd*=x0	2^{10}	
SLTU	*rd*=x0	2^{10}	

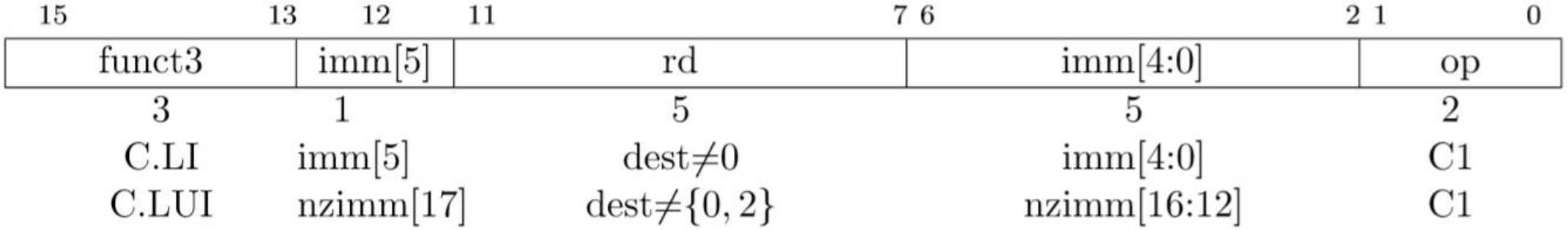
Table 5.1: RV64I HINT instructions.

Chapter 16 “C” Standard Extension for Compressed Instructions

16.5 Integer Computational Instructions

RVC提供了几种用于整数算术和常量生成的指令

下面这两个常量生成指令均使用CI（Computational Integer）指令格式，并且可以针对任何整数寄存器。



15 13	12	11 7	6 2	1 0
funct3	imm[5]	rd	imm[4:0]	op
3	1	5	5	2
C.LI	imm[5]	dest≠0	imm[4:0]	C1
C.LUI	nzimm[17]	dest≠{0, 2}	nzimm[16:12]	C1

C.LI将符号扩展的6位立即数imm加载到寄存器rd中。 C.LI扩展为addi rd，x0，imm [5： 0]。 C.LI仅在rd！ = x0时有效。具有 rd = x0的代码点被编码为HINTs（提示）。 nzimm(non-zero imm)

c.li x0, 0

15 13	12	11 7	6 2	1 0
funct3	imm[5]	rd	imm[4:0]	op
3	1	5	5	2
C.LI	imm[5]	dest≠0	imm[4:0]	C1
C.LUI	nzimm[17]	dest≠{0, 2}	nzimm[16:12]	C1

C.LUI将非零的6位立即数字段加载到目标寄存器的位17-12中，清除低12位，然后将位17符号扩展到目标的所有高位中。 C.LUI扩展为lui rd, nzimm [17:12]。 C.LUI仅在rd！ = {x0，x2}且立即数不等于0时有效。 rd = x0的其余代码点被编码为HINTs（提示）。

c.lui x0, 1

NOP Instruction

15 13	12	11 7	6 2	1 0
funct3	imm[5]	rd/rs1	imm[4:0]	op
3	1	5	5	2
C.NOP	0	0	0	C1

C.NOP是一种CI格式的指令，除了advanced PC和增加任何适用的性能计数器外，它不会更改任何用户可见的状态。 C.NOP扩展到nop。 C.NOP仅在imm = 0时有效。 imm!= 0的代码点被编码为HINTs(提示)

c.nop 8

c.nop 0（immediate must be non-zero in the range [-32, 31]）

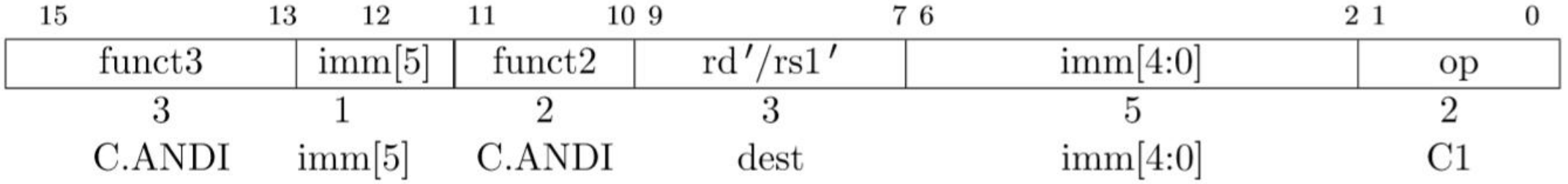
一个nop为1个指令周期,主要用于精确定时或者延时

nop指令的作用:

1 通过nop指令的填充(nop指令一个字节),使指令按字对齐,从而减少取指令时的内存访问次数。(一般用来内存地址偶数对齐,比如有一条指令,占3字节,这时候使用nop指令,cpu 就可以从第四个字节处读取指令了。

2 通过nop指令产生一定的延迟，但是对于快速的CPU来说效果不明显

3 i/o传输时，也会用一下 nop，等待缓冲区清空，总线恢复。



C.ANDI是一种CB格式的指令，它计算寄存器rd'中的值与符号扩展的6位立即数的按位与，然后将结果写入rd'。 C.ANDI扩展为andi rd', rd', imm [5：0]。代码点被编码为HINTs（提示）。

c.addi a0, 0

Integer Register-Register Operations

15 12	11 7	6 2	1 0
funct4	rd/rs1	rs2	op
4	5	5	2
C.MV	dest≠0	src≠0	C2
C.ADD	dest≠0	src≠0	C2

这些说明使用CR格式。

C.MV将寄存器rs2中的值复制到寄存器rd中。 C.MV扩展为加法rd，x0，rs2。 C.MV仅在rs2/= x0时有效； rs2/= x0和rd = x0的代码点被编码为HINTs（提示）。

c.mv x0, a0

Integer Register-Register Operations

15 12	11 7	6 2	1 0
funct4	rd/rs1	rs2	op
4	5	5	2
C.MV	dest≠0	src≠0	C2
C.ADD	dest≠0	src≠0	C2

C.ADD将值添加到寄存器rd和rs2中，并将结果写入寄存器rd。C.ADD扩展为添加rd，rd，rs2。 C.ADD仅在rs2/= x0时有效。 rs2 = x0的代码点对应于C.JALR和C.EBREAK指令。 rs2/= x0和rd = x0的代码点被编码为HINTs（提示）。

c.add x0, a0

- 问题

其他文件中对于添加rvc-hints属性的支持找的不顺利，关键是不知道对应属性的各种参数设置在哪个文件中定义，就需要先看一下RISCV/下每一个文件的作用，然后在一个文件一个文件看。然后在RISCVInstrInfo.td，

RISCVInstrInfoC.td

中对照官方llvm源代码，对C910缺少的部分进行diff ？ ？ ？

但是如果llvm源代码目前都不支持的mattr，添加起来困难就更多了。

- 参考资料

llvm学习笔记（43）

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

LLVM Programmer's Manual

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

Design of the RISC-V Instruction Set Architecture笔记（chapter5）

https://blog.csdn.net/shuiliusheng/article/details/82691128

《Tutorial: Creating an LLVM Backend for the Cpu0 Architecture Release 3.9.1》

谢 谢

欢迎交流合作

2019/02/25