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

RISCV后端和llvm-mc介绍

软件所智能软件中心PLCT实验室 王鹏 实习生

目 录

• 01 llvm-mc介绍

llvm-mc 一个通用的可定制的编译器驱动

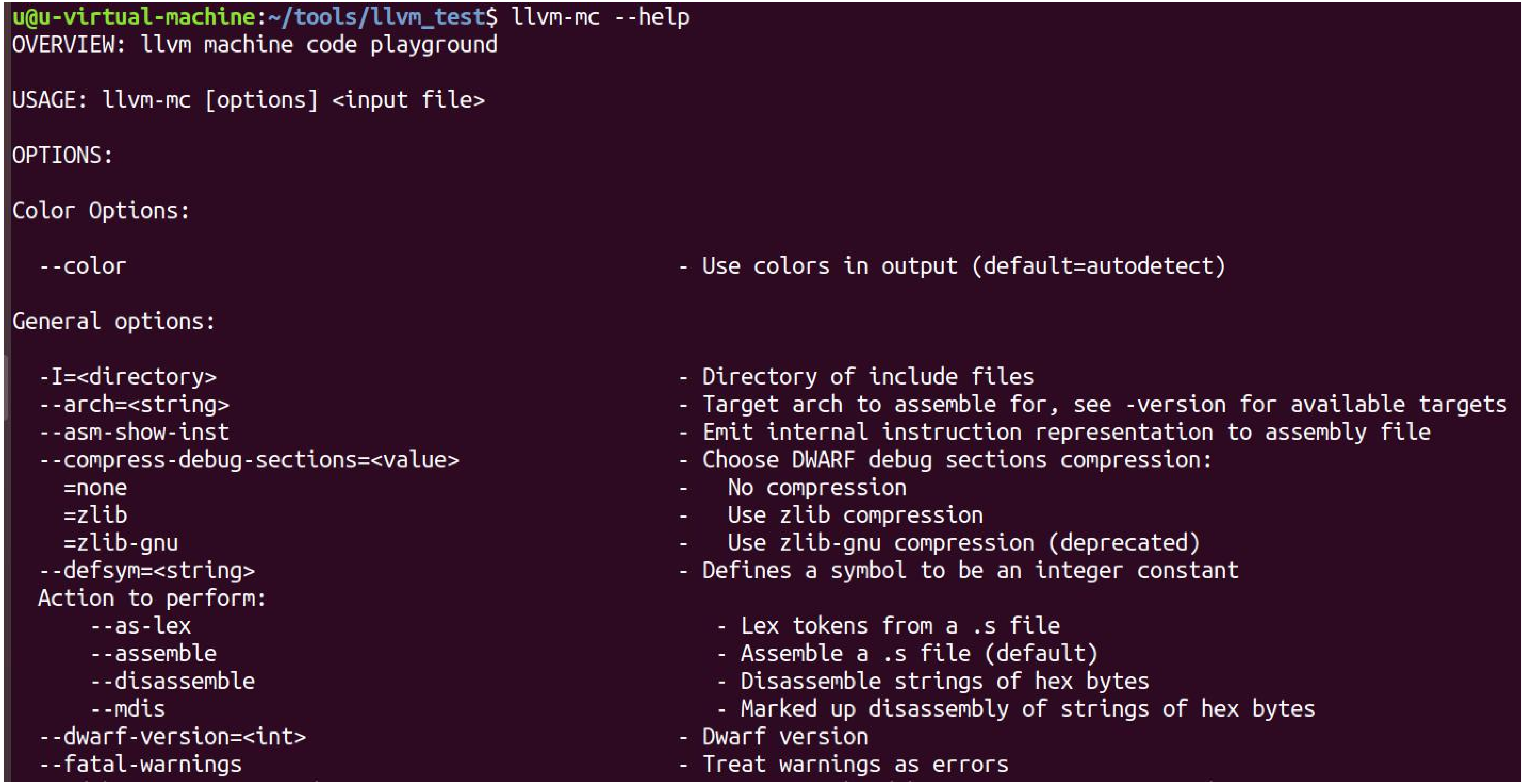
The llvm-mc tool, 称为 "LLVM machine code playground", 用于测试目标MC实施的组件。 该工具完成的主要任务是汇编一个.s文件（通过-assemble命令），反汇编字节串（-disassemble）以及显示指令及其内部表示的编码（-show-encoding和-show -inst）。

llvm-mc 能够支持大多数的机器架构，包括 x86, x86-64, ARM, ARM64，RISCV 以及大部分的 MIPS 架构等。

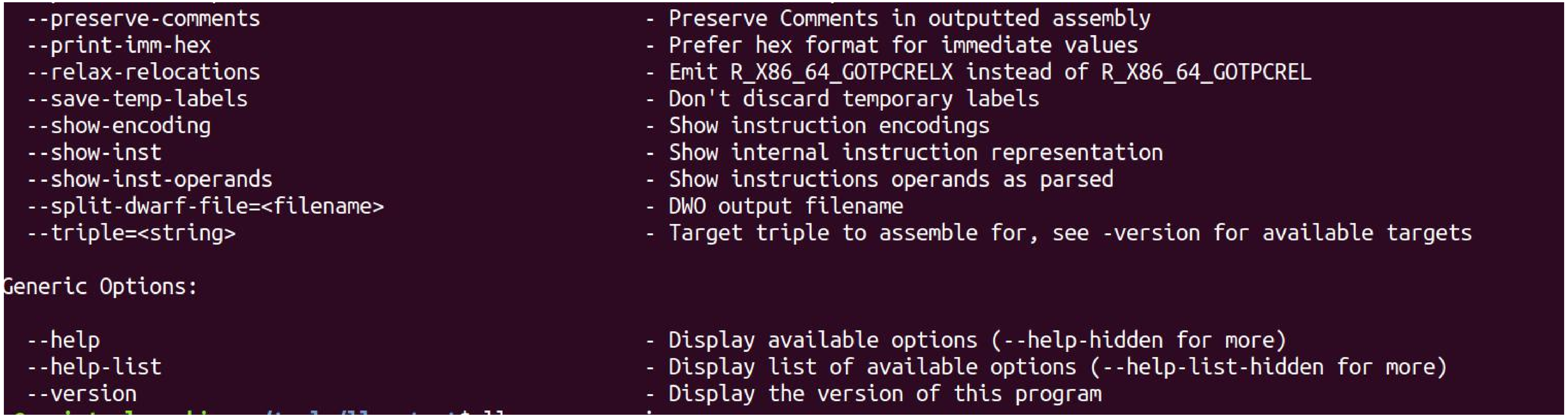
为了确定体系结构，使用了参数-arch = arch，并且使用了-mattr = a1，+ a2，-a3来启用和禁用有效指令所需的处理器功能。

下面针对OpenRISC 1000的架构，示例是对乘法和除法指令的可选支持。要启用这些功能，将使用以下命令。

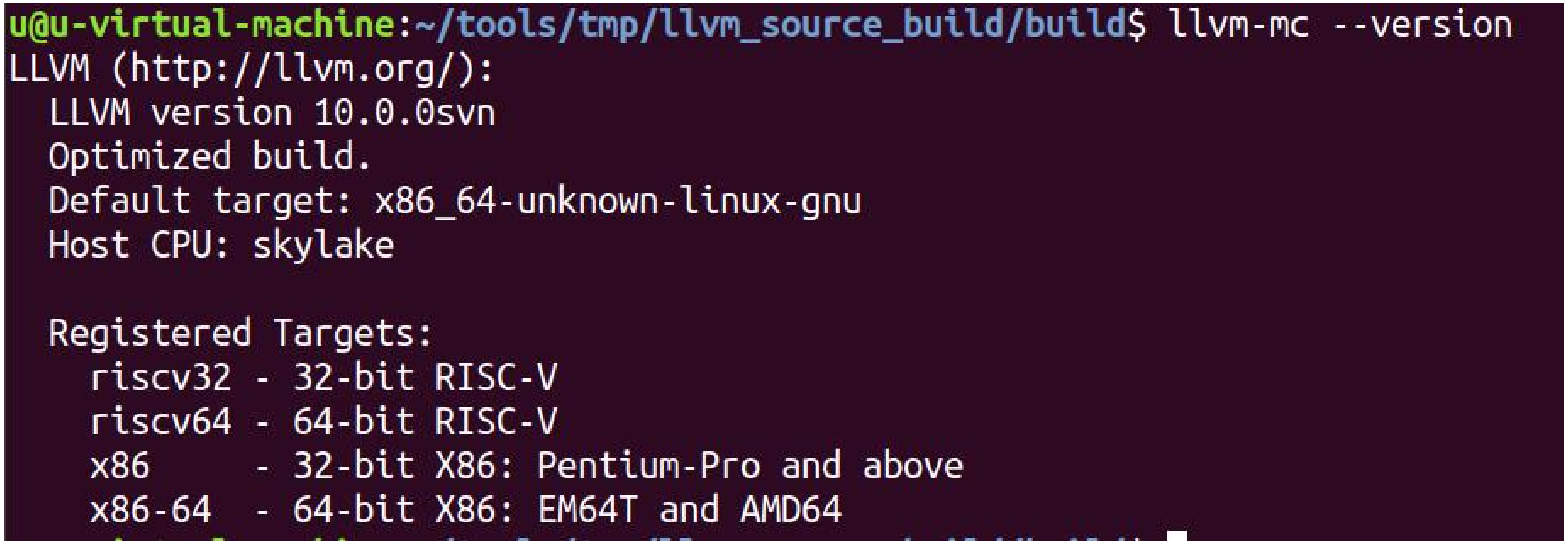
```
llvm-mc -assemble -show-encoding -arch=or1k -mattr=+mul,+div
input.s
```



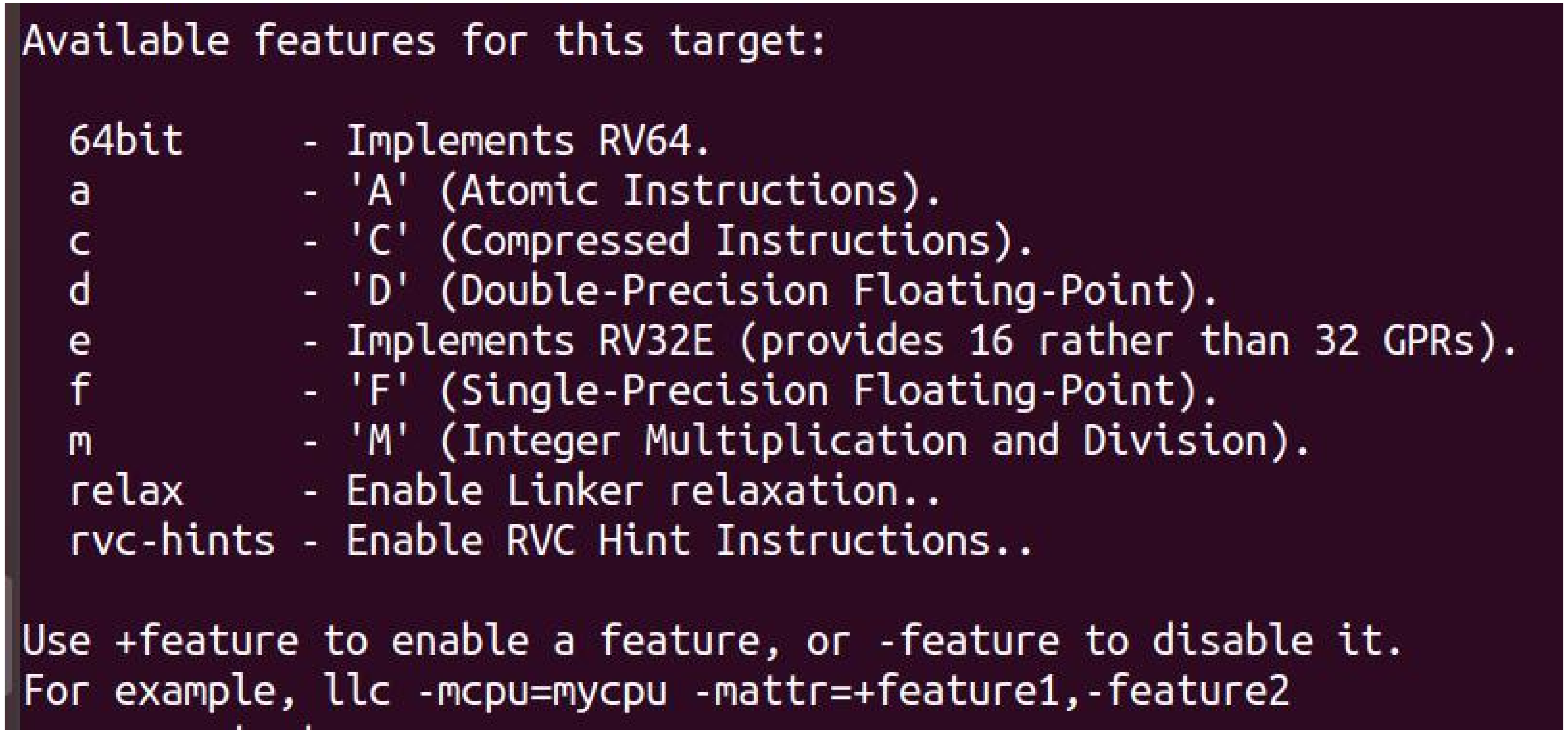
```
u@u-virtual-machine:~/tools/llvm_test$ llvm-mc --help
OVERVIEW: llvm machine code playground

USAGE: llvm-mc [options] <input file>

OPTIONS:

Color Options:

  --color                                             - Use colors in output (default=autodetect)

General options:

  -I=<directory>                                      - Directory of include files
  --arch=<string>                                     - Target arch to assemble for, see -version for available targets
  --asm-show-inst                                     - Emit internal instruction representation to assembly file
  --compress-debug-sections=<value>                   - Choose DWARF debug sections compression:
    =none                                             -   No compression
    =zlib                                             -   Use zlib compression
    =zlib-gnu                                         -   Use zlib-gnu compression (deprecated)
  --defsym=<string>                                   - Defines a symbol to be an integer constant
  Action to perform:
      --as-lex                                           - Lex tokens from a .s file
      --assemble                                         - Assemble a .s file (default)
      --disassemble                                      - Disassemble strings of hex bytes
      --mdis                                             - Marked up disassembly of strings of hex bytes
  --dwarf-version=<int>                               - Dwarf version
  --fatal-warnings                                    - Treat warnings as errors
```



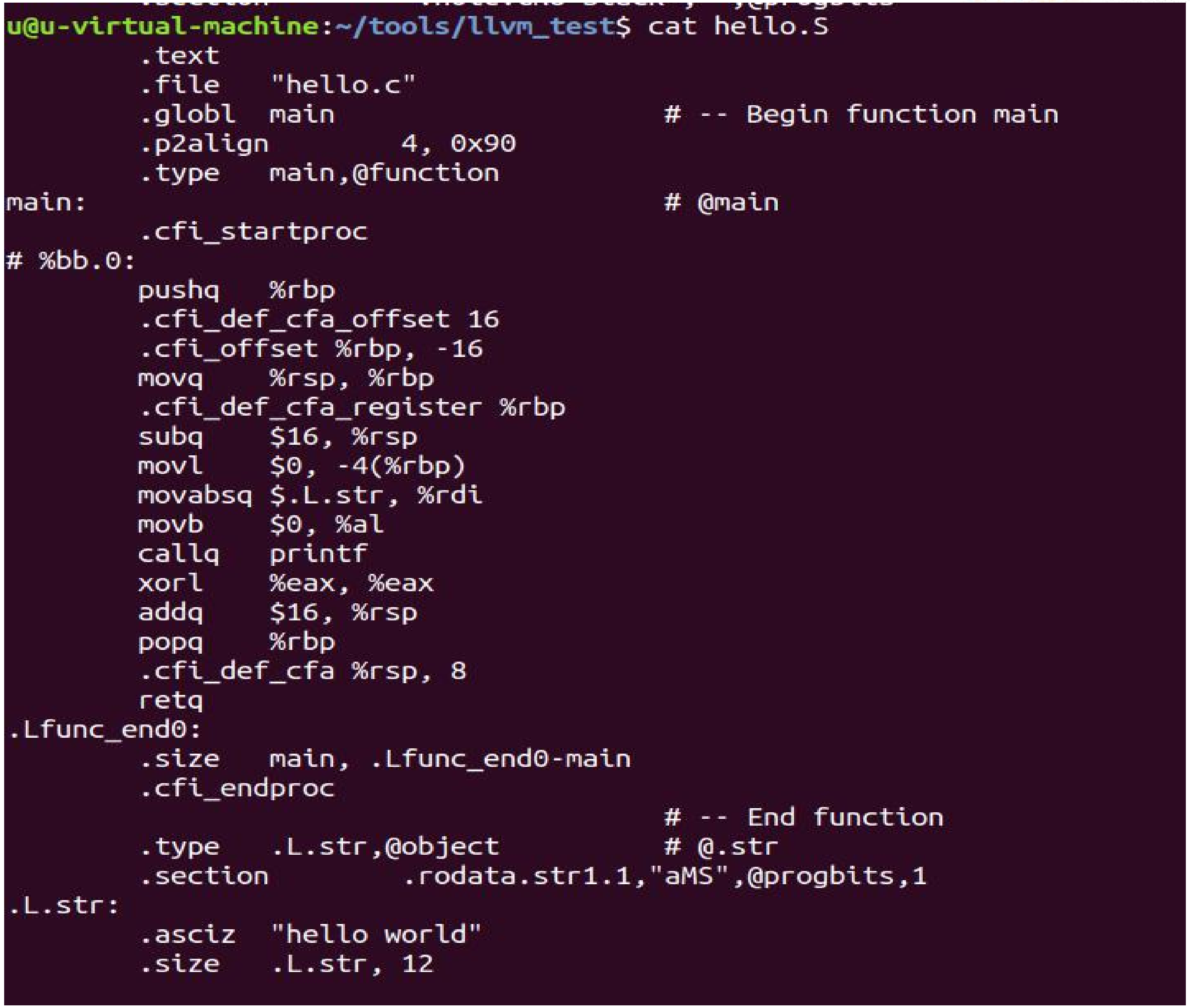
```
  --preserve-comments                                  - Preserve Comments in outputted assembly
  --print-imm-hex                                      - Prefer hex format for immediate values
  --relax-relocations                                  - Emit R_X86_64_GOTPCRELX instead of R_X86_64_GOTPCREL
  --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
```


```
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ llvm-mc --version
LLVM (http://llvm.org/):
  LLVM version 10.0.0svn
  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
```

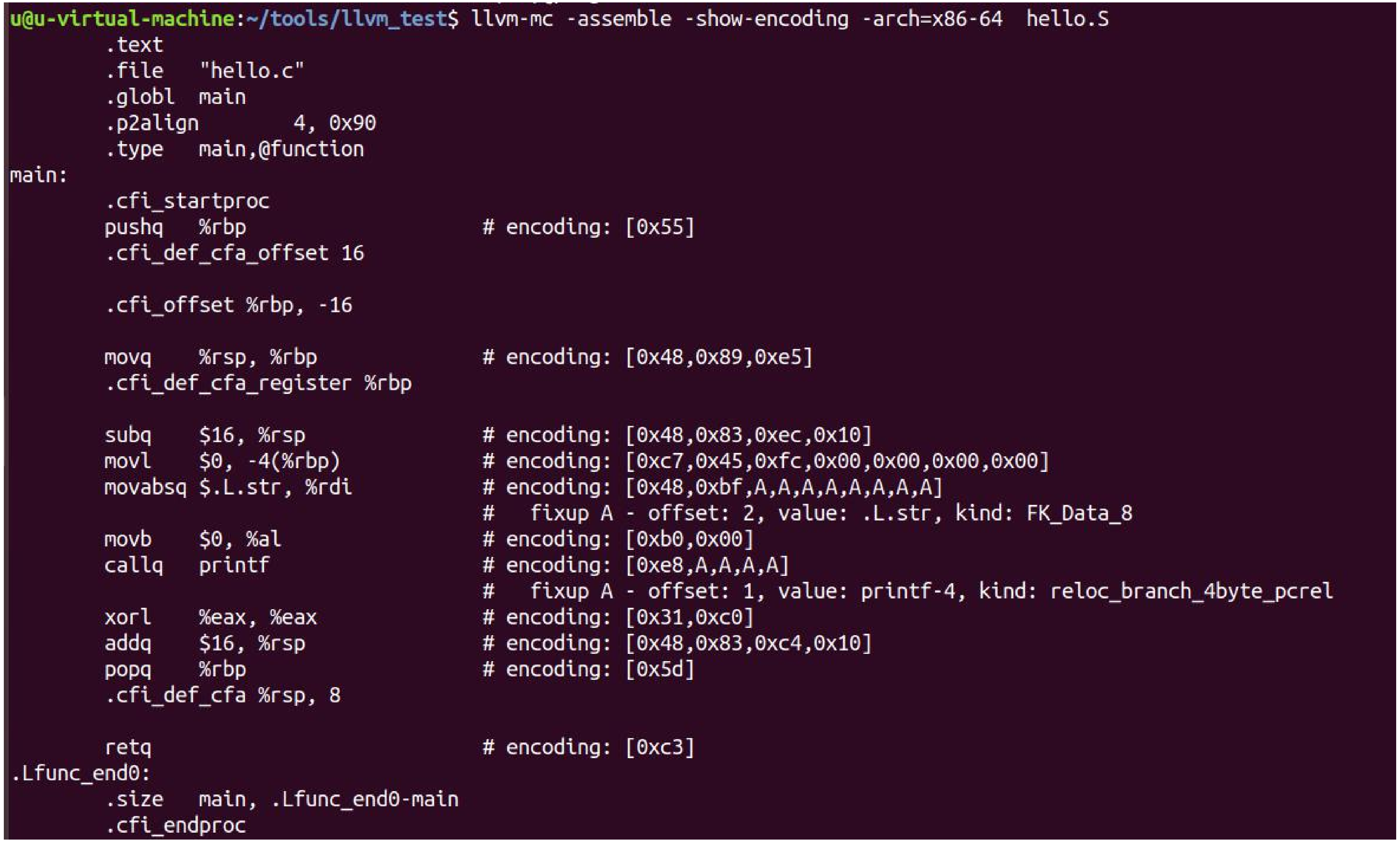
ISCAS 中国科学院软件研究所 Institute of Software Chinese Academy of Sciences | 智能软件研究中心 Intelligent Software Research Center



```
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..
  rvc-hints - Enable RVC Hint Instructions..

Use +feature to enable a feature, or -feature to disable it.
For example, llc -mcpu=mycpu -mattr=+feature1,-feature2
```

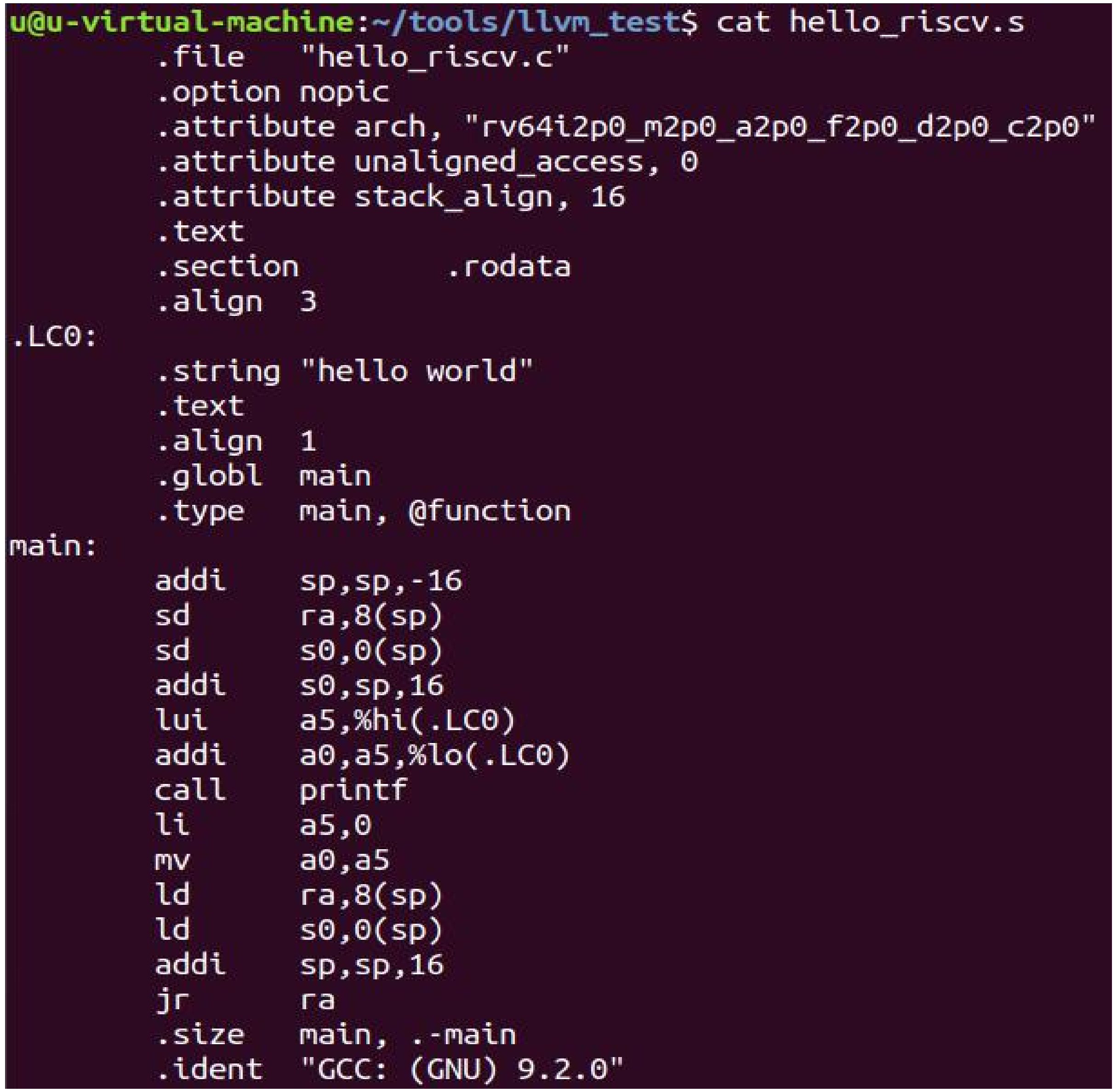


```
        .section        ".note.GNU-stack","",@progbits
u@u-virtual-machine:~/tools/llvm_test$ cat hello.S
        .text
        .file   "hello.c"
        .globl  main                    # -- Begin function main
        .p2align        4, 0x90
        .type   main,@function
main:                                   # @main
        .cfi_startproc
# %bb.0:
        pushq   %rbp
        .cfi_def_cfa_offset 16
        .cfi_offset %rbp, -16
        movq    %rsp, %rbp
        .cfi_def_cfa_register %rbp
        subq    $16, %rsp
        movl    $0, -4(%rbp)
        movabsq $.L.str, %rdi
        movb    $0, %al
        callq   printf
        xorl    %eax, %eax
        addq    $16, %rsp
        popq    %rbp
        .cfi_def_cfa %rsp, 8
        retq
.Lfunc_end0:
        .size   main, .Lfunc_end0-main
        .cfi_endproc
                                        # -- End function
        .type   .L.str,@object          # @.str
        .section        .rodata.str1.1,"aMS",@progbits,1
.L.str:
        .asciz  "hello world"
        .size   .L.str, 12
```



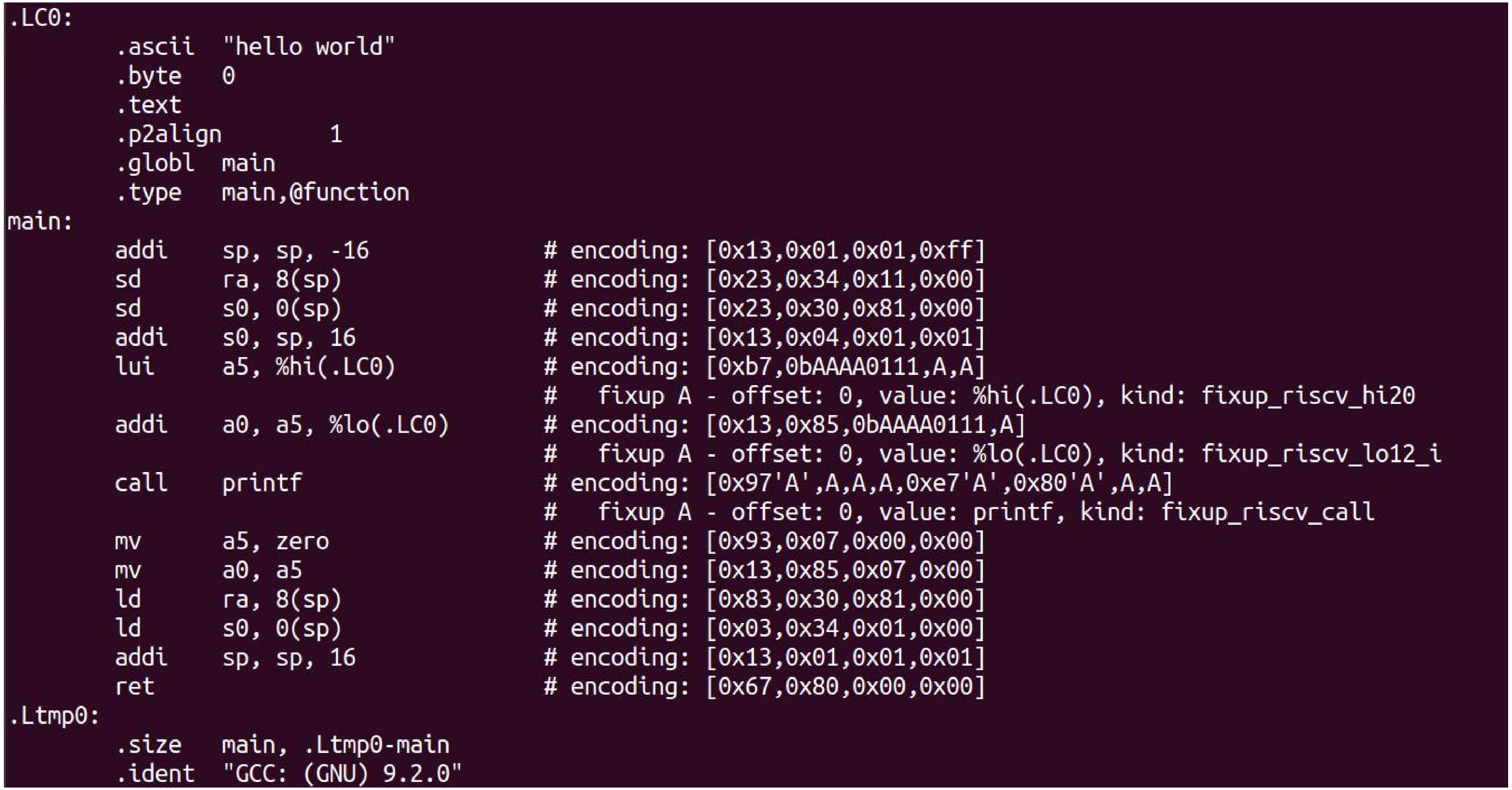
```
u@u-virtual-machine:~/tools/llvm_test$ llvm-mc -assemble -show-encoding -arch=x86-64  hello.S
        .text
        .file   "hello.c"
        .globl  main
        .p2align        4, 0x90
        .type   main,@function
main:
        .cfi_startproc
        pushq   %rbp                    # encoding: [0x55]
        .cfi_def_cfa_offset 16

        .cfi_offset %rbp, -16

        movq    %rsp, %rbp              # encoding: [0x48,0x89,0xe5]
        .cfi_def_cfa_register %rbp

        subq    $16, %rsp               # encoding: [0x48,0x83,0xec,0x10]
        movl    $0, -4(%rbp)            # encoding: [0xc7,0x45,0xfc,0x00,0x00,0x00,0x00]
        movabsq $.L.str, %rdi           # encoding: [0x48,0xbf,A,A,A,A,A,A,A,A]
                                        #   fixup A - offset: 2, value: .L.str, kind: FK_Data_8
        movb    $0, %al                 # encoding: [0xb0,0x00]
        callq   printf                  # encoding: [0xe8,A,A,A,A]
                                        #   fixup A - offset: 1, value: printf-4, kind: reloc_branch_4byte_pcrel
        xorl    %eax, %eax              # encoding: [0x31,0xc0]
        addq    $16, %rsp               # encoding: [0x48,0x83,0xc4,0x10]
        popq    %rbp                    # encoding: [0x5d]
        .cfi_def_cfa %rsp, 8

        retq                            # encoding: [0xc3]
.Lfunc_end0:
        .size   main, .Lfunc_end0-main
        .cfi_endproc
```

用riscv64-unknown-elf-gcc交叉编译出riscv的汇编语言



```
u@u-virtual-machine:~/tools/llvm_test$ cat hello_riscv.s
        .file   "hello_riscv.c"
        .option nopic
        .attribute arch, "rv64i2p0_m2p0_a2p0_f2p0_d2p0_c2p0"
        .attribute unaligned_access, 0
        .attribute stack_align, 16
        .text
        .section        .rodata
        .align  3
.LC0:
        .string "hello world"
        .text
        .align  1
        .globl  main
        .type   main, @function
main:
        addi    sp,sp,-16
        sd      ra,8(sp)
        sd      s0,0(sp)
        addi    s0,sp,16
        lui     a5,%hi(.LC0)
        addi    a0,a5,%lo(.LC0)
        call    printf
        li      a5,0
        mv      a0,a5
        ld      ra,8(sp)
        ld      s0,0(sp)
        addi    sp,sp,16
        jr      ra
        .size   main, .-main
        .ident  "GCC: (GNU) 9.2.0"
```

llvm-mc -assemble -show-encoding -arch=riscv64 -mattr=+f hello_riscv.s



```
.LC0:
        .ascii  "hello world"
        .byte   0
        .text
        .p2align        1
        .globl  main
        .type   main,@function
main:
        addi    sp, sp, -16             # encoding: [0x13,0x01,0x01,0xff]
        sd      ra, 8(sp)               # encoding: [0x23,0x34,0x11,0x00]
        sd      s0, 0(sp)               # encoding: [0x23,0x30,0x81,0x00]
        addi    s0, sp, 16              # encoding: [0x13,0x04,0x01,0x01]
        lui     a5, %hi(.LC0)           # encoding: [0xb7,0bAAAA0111,A,A]
                                        #   fixup A - offset: 0, value: %hi(.LC0), kind: fixup_riscv_hi20
        addi    a0, a5, %lo(.LC0)       # encoding: [0x13,0x85,0bAAAA0111,A]
                                        #   fixup A - offset: 0, value: %lo(.LC0), kind: fixup_riscv_lo12_i
        call    printf                  # encoding: [0x97'A',A,A,A,0xe7'A',0x80'A',A,A]
                                        #   fixup A - offset: 0, value: printf, kind: fixup_riscv_call
        mv      a5, zero                # encoding: [0x93,0x07,0x00,0x00]
        mv      a0, a5                  # encoding: [0x13,0x85,0x07,0x00]
        ld      ra, 8(sp)               # encoding: [0x83,0x30,0x81,0x00]
        ld      s0, 0(sp)               # encoding: [0x03,0x34,0x01,0x00]
        addi    sp, sp, 16              # encoding: [0x13,0x01,0x01,0x01]
        ret                             # encoding: [0x67,0x80,0x00,0x00]
.Ltmp0:
        .size   main, .Ltmp0-main
        .ident  "GCC: (GNU) 9.2.0"
```

要反汇编ARM的sdiv指令，我们需要像下面这样显式地打开hwdiv-arm功能，否则llvm-mc将报告无效的编码错误。

```
  echo "0x10 0xf1 0x10 0xe7" | llvm-mc -disassemble -arch=arm -
mattr=+hwdiv-arm
```

-assemble - Assemble a .s file (default)

-disassemble - Disassemble strings of hex bytes

-mattr=<a1,+a2,-a3,...>   - Target specific attributes (-mattr=help for details)

每一种triple的编码形式不同，互相不能识别

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

-mcpu=<cpu-name>   - Target a specific cpu type

(-mcpu=help for details)

-triple=<string>   - Target triple to assemble for, see -version for available targets 更加specific

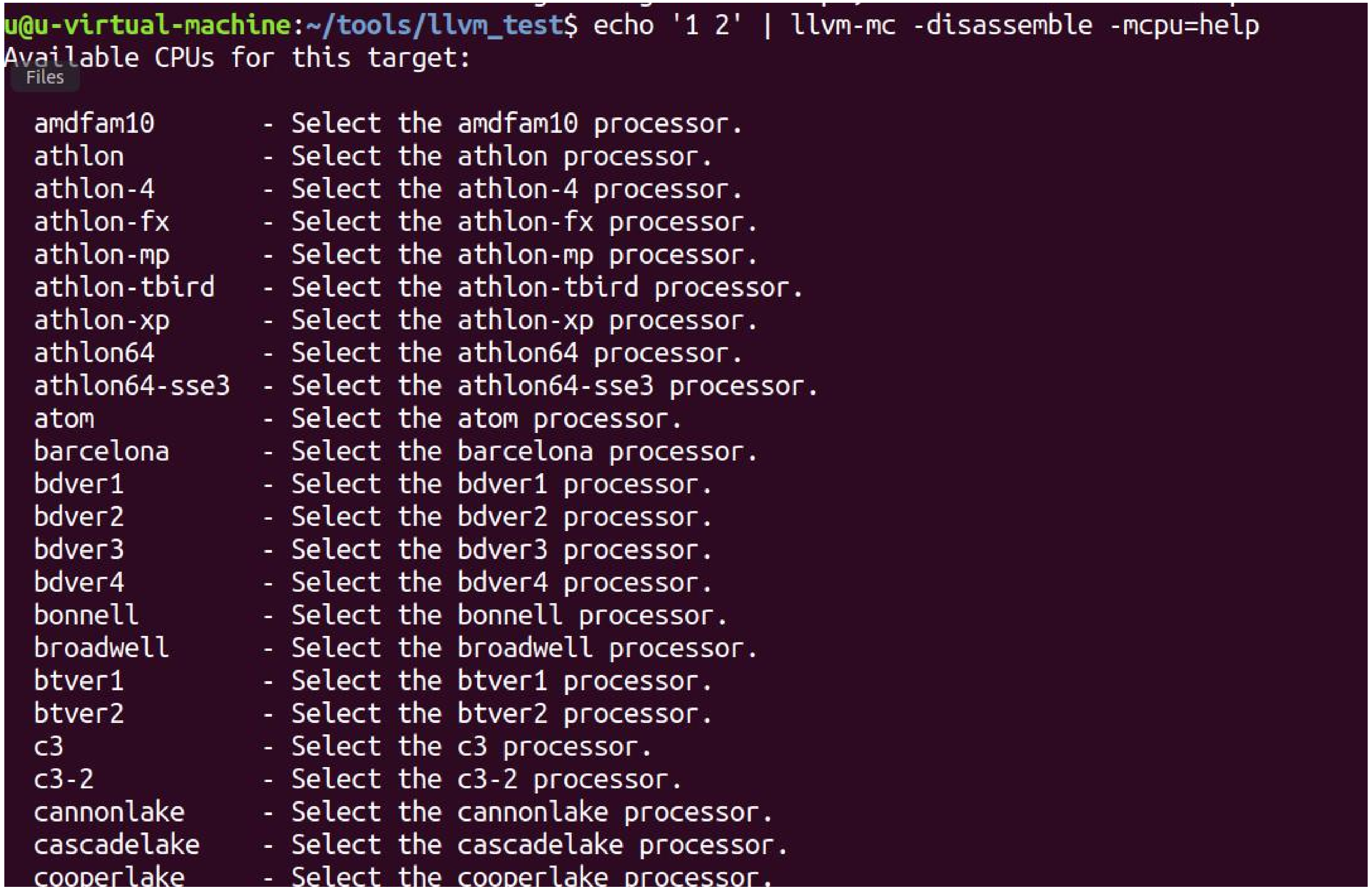
Specify target triple (e.g. i686-apple-darwin9)

-arch=<string>   - Target arch to assemble for, see -version for available targets pentium系列

i386系列

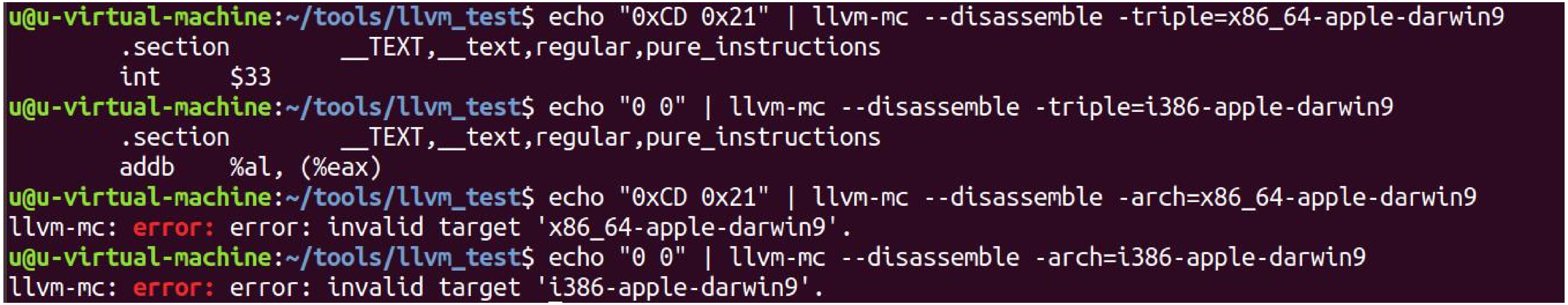
x86-64等

Available features for this target:...



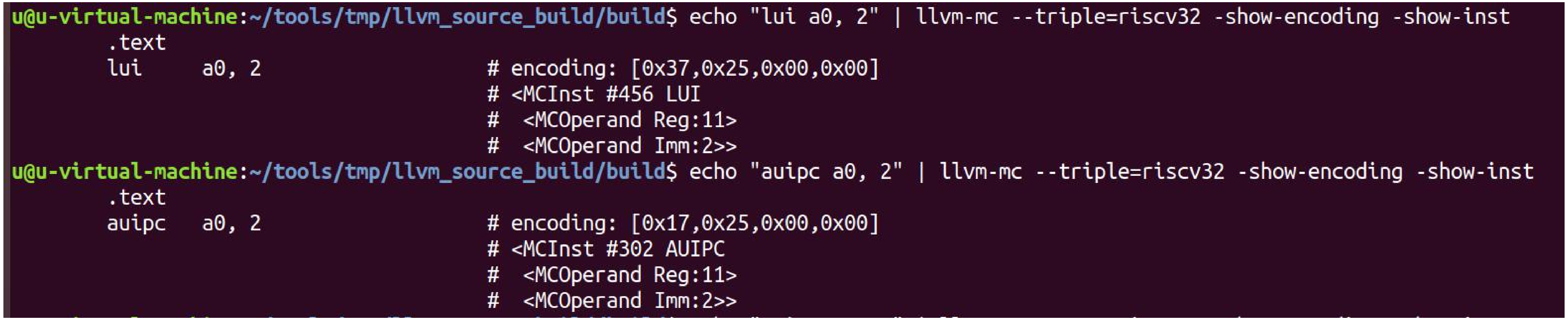
```
u@u-virtual-machine:~/tools/llvm_test$ echo '1 2' | llvm-mc -disassemble -mcpu=help
Available CPUs for this target:

  amdfam10         - Select the amdfam10 processor.
  athlon           - Select the athlon processor.
  athlon-4         - Select the athlon-4 processor.
  athlon-fx        - Select the athlon-fx processor.
  athlon-mp        - Select the athlon-mp processor.
  athlon-tbird     - Select the athlon-tbird processor.
  athlon-xp        - Select the athlon-xp processor.
  athlon64         - Select the athlon64 processor.
  athlon64-sse3    - Select the athlon64-sse3 processor.
  atom             - Select the atom processor.
  barcelona        - Select the barcelona processor.
  bdver1           - Select the bdver1 processor.
  bdver2           - Select the bdver2 processor.
  bdver3           - Select the bdver3 processor.
  bdver4           - Select the bdver4 processor.
  bonnell          - Select the bonnell processor.
  broadwell        - Select the broadwell processor.
  btver1           - Select the btver1 processor.
  btver2           - Select the btver2 processor.
  c3               - Select the c3 processor.
  c3-2             - Select the c3-2 processor.
  cannonlake       - Select the cannonlake processor.
  cascadelake      - Select the cascadelake processor.
  cooperlake       - Select the cooperlake processor.
```

-triple=<string> 和 -arch=<string>比较



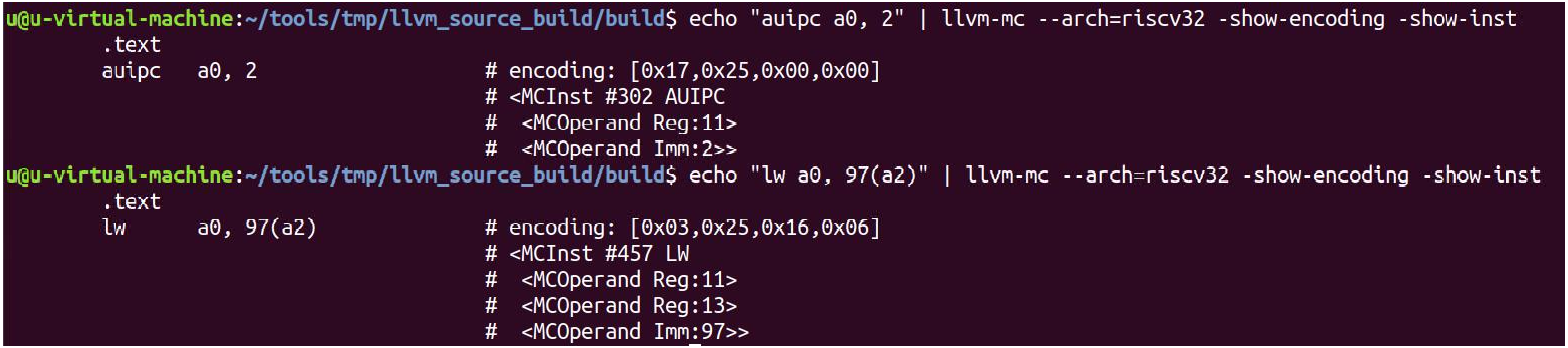
```
u@u-virtual-machine:~/tools/llvm_test$ echo "0xCD 0x21" | llvm-mc --disassemble -triple=x86_64-apple-darwin9
	.section	__TEXT,__text,regular,pure_instructions
	int	$33
u@u-virtual-machine:~/tools/llvm_test$ echo "0 0" | llvm-mc --disassemble -triple=i386-apple-darwin9
	.section	__TEXT,__text,regular,pure_instructions
	addb	%al, (%eax)
u@u-virtual-machine:~/tools/llvm_test$ echo "0xCD 0x21" | llvm-mc --disassemble -arch=x86_64-apple-darwin9
llvm-mc: error: error: invalid target 'x86_64-apple-darwin9'.
u@u-virtual-machine:~/tools/llvm_test$ echo "0 0" | llvm-mc --disassemble -arch=i386-apple-darwin9
llvm-mc: error: error: invalid target 'i386-apple-darwin9'.
```

llvm/test/MC/RISCV/rv32i-valid.s



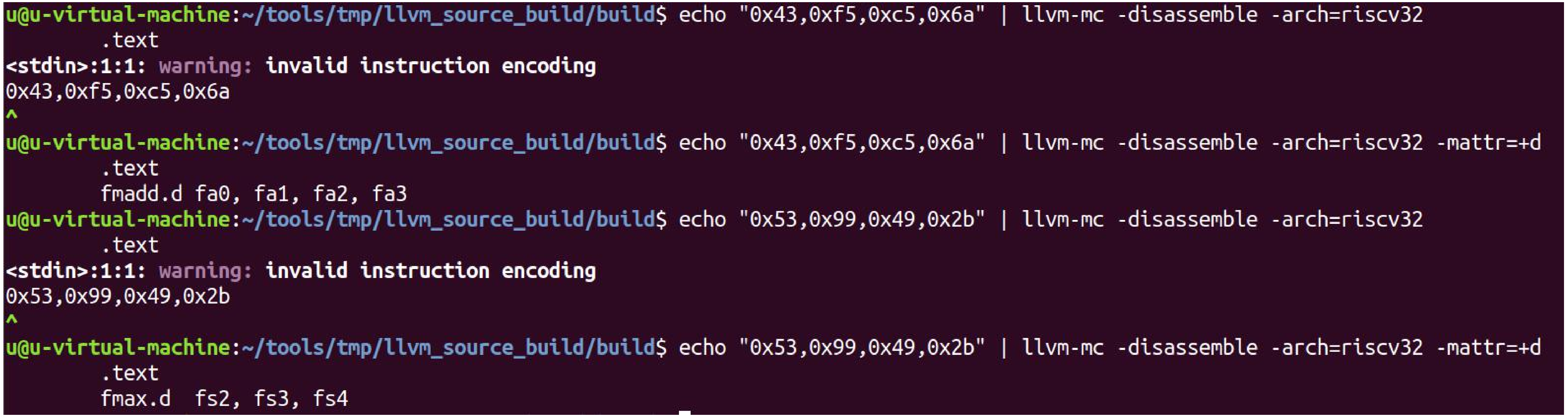
```
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "lui a0, 2" | llvm-mc --triple=riscv32 -show-encoding -show-inst
	.text
	lui	a0, 2                           # encoding: [0x37,0x25,0x00,0x00]
                                        # <MCInst #456 LUI
                                        #  <MCOperand Reg:11>
                                        #  <MCOperand Imm:2>>
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "auipc a0, 2" | llvm-mc --triple=riscv32 -show-encoding -show-inst
	.text
	auipc	a0, 2                           # encoding: [0x17,0x25,0x00,0x00]
                                        # <MCInst #302 AUIPC
                                        #  <MCOperand Reg:11>
                                        #  <MCOperand Imm:2>>
```

llvm/test/MC/RISCV/rv32i-valid.s



```
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "auipc a0, 2" | llvm-mc --arch=riscv32 -show-encoding -show-inst
        .text
        auipc   a0, 2                   # encoding: [0x17,0x25,0x00,0x00]
                                        # <MCInst #302 AUIPC
                                        #  <MCOperand Reg:11>
                                        #  <MCOperand Imm:2>>
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "lw a0, 97(a2)" | llvm-mc --arch=riscv32 -show-encoding -show-inst
        .text
        lw      a0, 97(a2)              # encoding: [0x03,0x25,0x16,0x06]
                                        # <MCInst #457 LW
                                        #  <MCOperand Reg:11>
                                        #  <MCOperand Reg:13>
                                        #  <MCOperand Imm:97>>
```

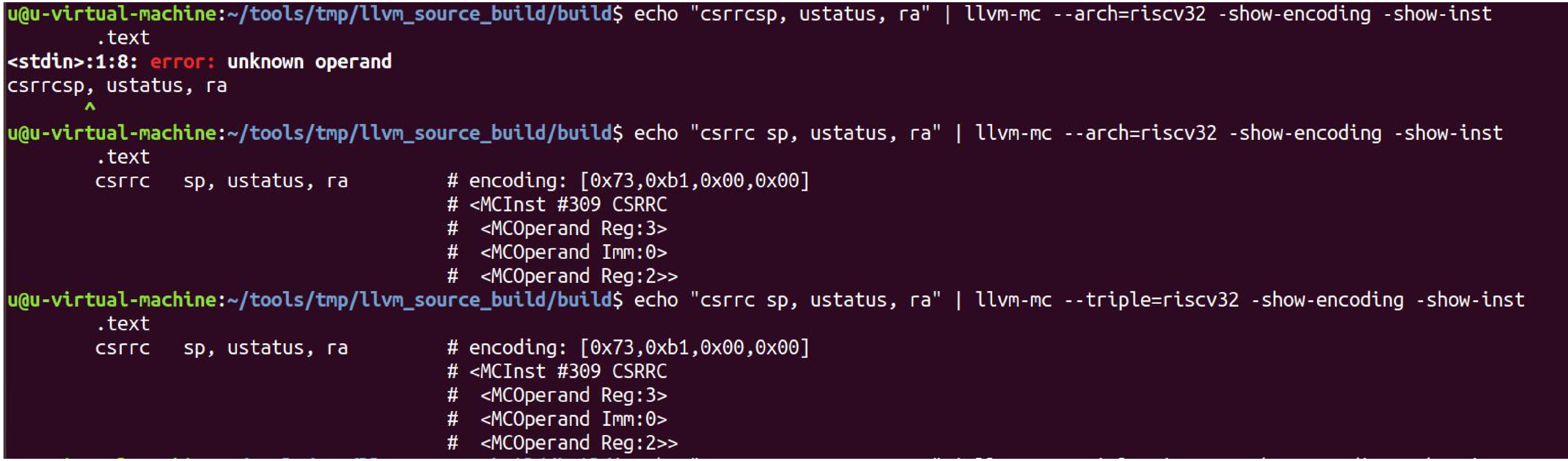
llvm/test/MC/RISCV/rv32d-valid.s



```
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "0x43,0xf5,0xc5,0x6a" | llvm-mc -disassemble -arch=riscv32
        .text
<stdin>:1:1: warning: invalid instruction encoding
0x43,0xf5,0xc5,0x6a
^
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "0x43,0xf5,0xc5,0x6a" | llvm-mc -disassemble -arch=riscv32 -mattr=+d
        .text
        fmadd.d fa0, fa1, fa2, fa3
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "0x53,0x99,0x49,0x2b" | llvm-mc -disassemble -arch=riscv32
        .text
<stdin>:1:1: warning: invalid instruction encoding
0x53,0x99,0x49,0x2b
^
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "0x53,0x99,0x49,0x2b" | llvm-mc -disassemble -arch=riscv32 -mattr=+d
        .text
        fmax.d  fs2, fs3, fs4
```

mattr的重要性

llvm/test/MC/RISCV/rv32d-valid.s

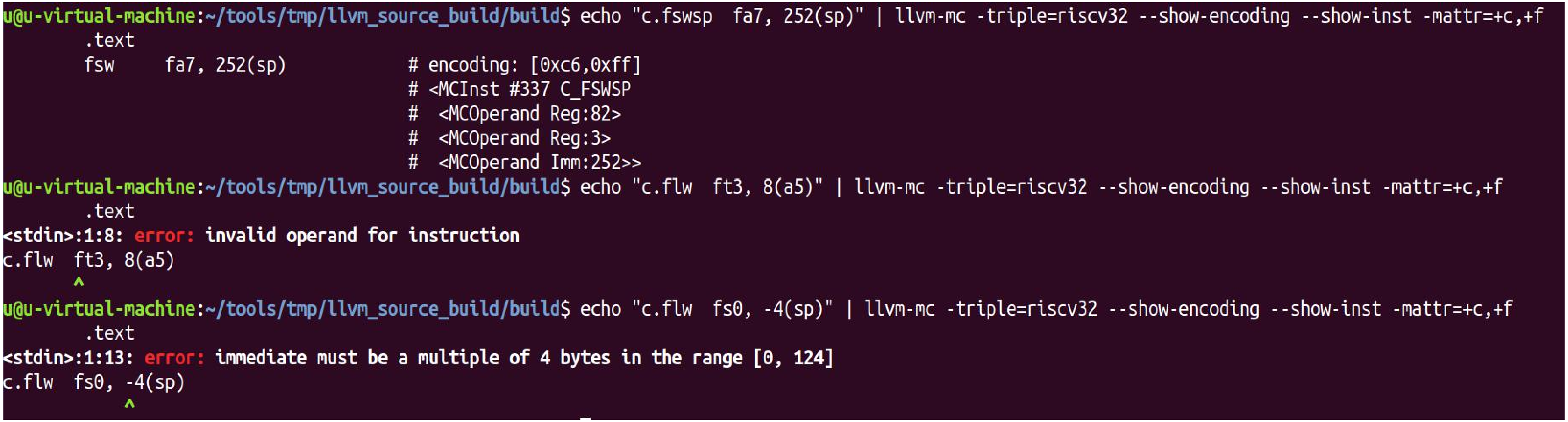


```
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "csrrcsp, ustatus, ra" | llvm-mc --arch=riscv32 -show-encoding -show-inst
        .text
<stdin>:1:8: error: unknown operand
csrrcsp, ustatus, ra
       ^
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "csrrc sp, ustatus, ra" | llvm-mc --arch=riscv32 -show-encoding -show-inst
        .text
        csrrc   sp, ustatus, ra         # encoding: [0x73,0xb1,0x00,0x00]
                                        # <MCInst #309 CSRRC
                                        #  <MCOperand Reg:3>
                                        #  <MCOperand Imm:0>
                                        #  <MCOperand Reg:2>>
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "csrrc sp, ustatus, ra" | llvm-mc --triple=riscv32 -show-encoding -show-inst
        .text
        csrrc   sp, ustatus, ra         # encoding: [0x73,0xb1,0x00,0x00]
                                        # <MCInst #309 CSRRC
                                        #  <MCOperand Reg:3>
                                        #  <MCOperand Imm:0>
                                        #  <MCOperand Reg:2>>
```

报错完善，“-”和“--”无差别，“---”报错提示 arch指的是本机特定的target（x86-64,x86,riscv32,riscv64）

triple

llvm/test/MC/RISCV/rv32fc-valid.s（1）和rv32fc-invalid.s（2&3）



```
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "c.fswsp  fa7, 252(sp)" | llvm-mc -triple=riscv32 --show-encoding --show-inst -mattr=+c,+f
        .text
        fsw     fa7, 252(sp)            # encoding: [0xc6,0xff]
                                        # <MCInst #337 C_FSWSP
                                        #  <MCOperand Reg:82>
                                        #  <MCOperand Reg:3>
                                        #  <MCOperand Imm:252>>
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "c.flw  ft3, 8(a5)" | llvm-mc -triple=riscv32 --show-encoding --show-inst -mattr=+c,+f
        .text
<stdin>:1:8: error: invalid operand for instruction
c.flw  ft3, 8(a5)
       ^
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo "c.flw  fs0, -4(sp)" | llvm-mc -triple=riscv32 --show-encoding --show-inst -mattr=+c,+f
        .text
<stdin>:1:13: error: immediate must be a multiple of 4 bytes in the range [0, 124]
c.flw  fs0, -4(sp)
            ^
```

• 2 RISCV后端介绍

编译器的最终目标是产生目标平台的代码，或者是产生汇编码，进而能够通过汇编器转为目标代码并能够在真实的硬件上执行。为了得到汇编码，编译器需要知道目标机器架构的各个方面——寄存器、指令集、调用约定、流水线等。

LLVM有自己的定义目标机器的方式——tablegen，通过它来指定目标的寄存器、指令集、调用约定等，并且tablegen函数以编程的方式缓解了描述一套架构属性所带来的困扰。

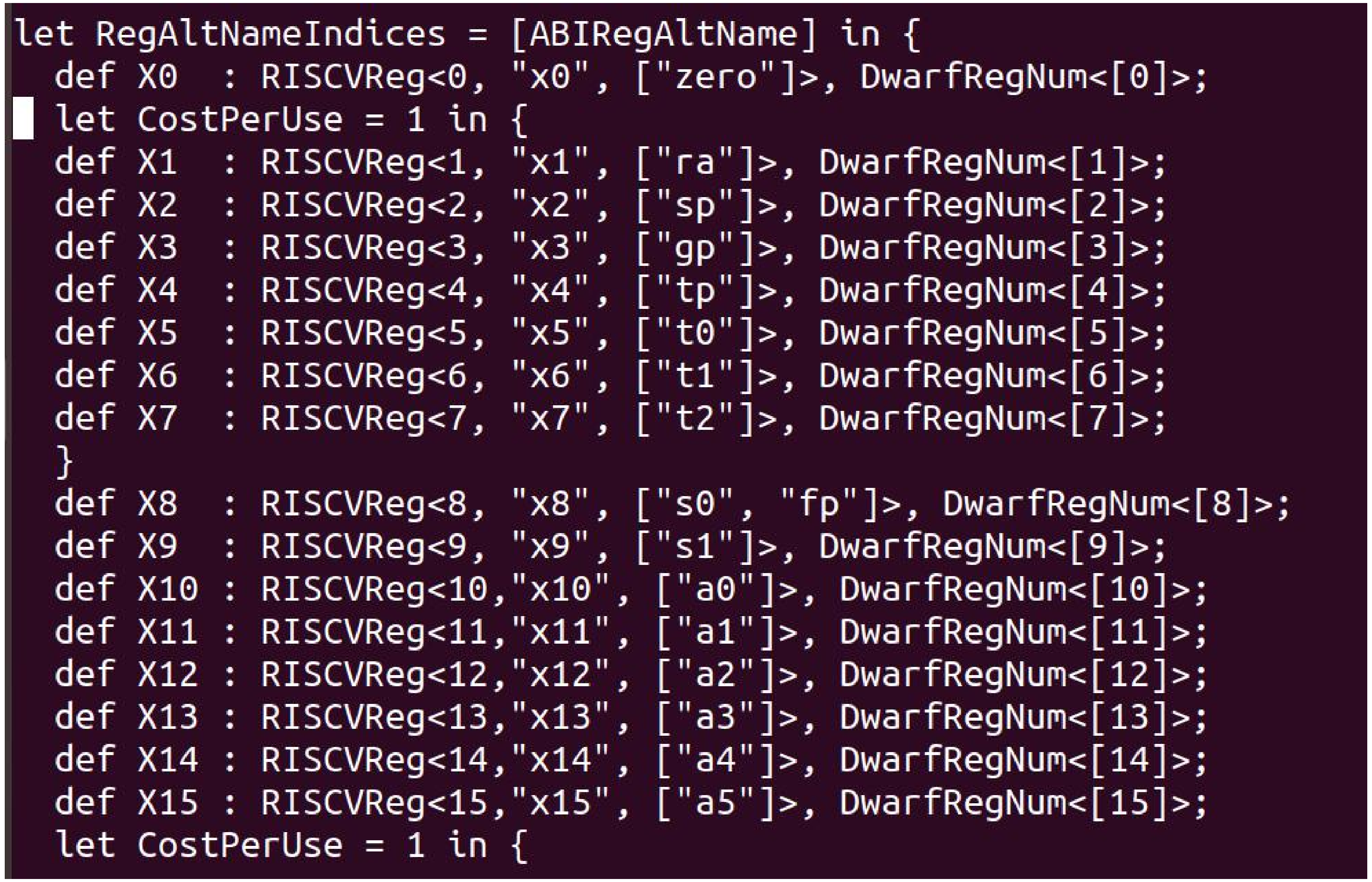
•2.1 定义寄存器和寄存器集合

RISCVRegisterInfo.td文件定义了寄存器和寄存器集合，之后通过tablegen函数可以把.td文件转为.inc文件，而这些文件可以在.cpp文件中通过#include声明引入，进而应用其中定义的寄存器

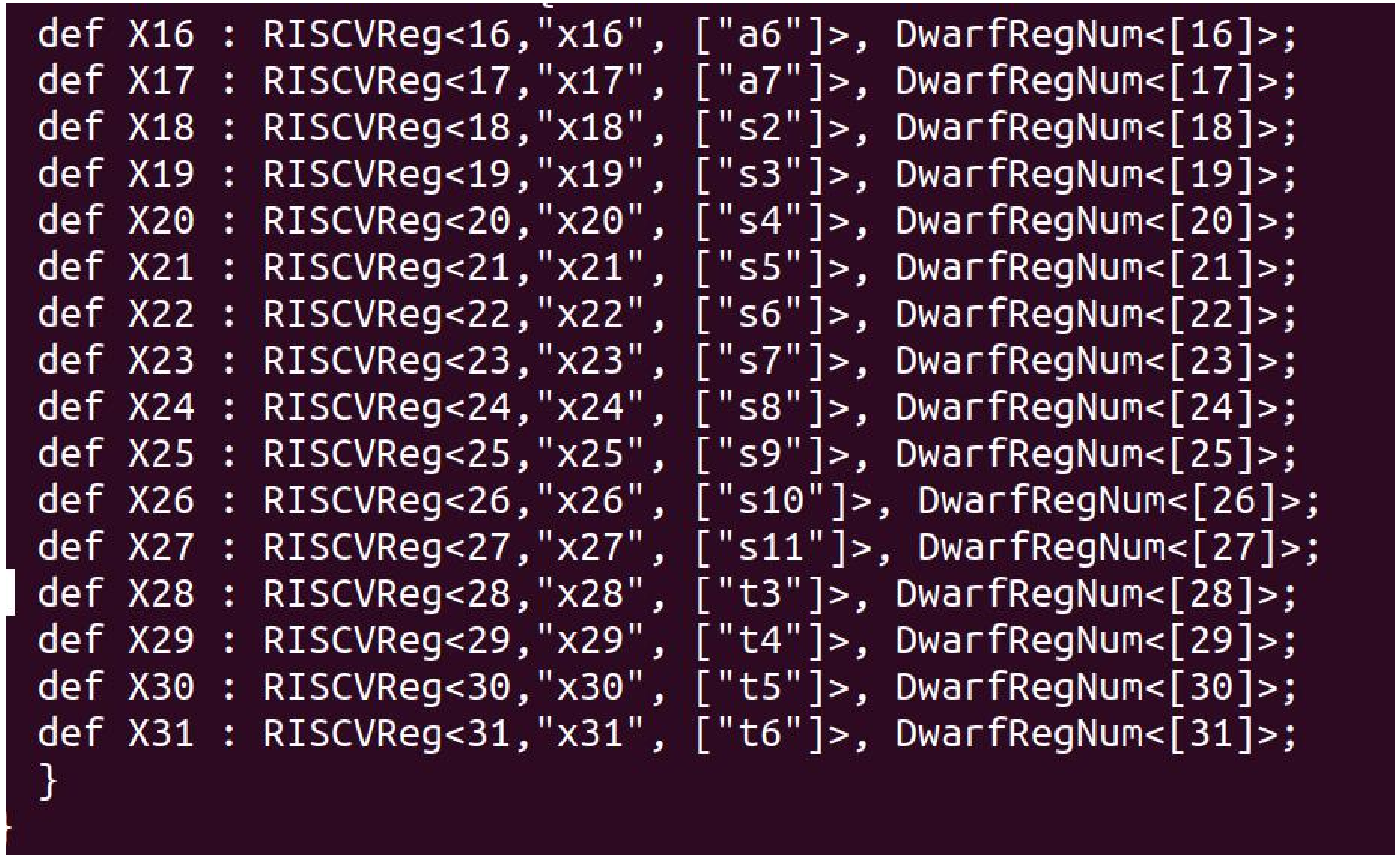
RISCV目标机器有32个普通寄存器及其别名，还有浮点寄存器。这些内容可以在 RISCVRegisterInfo.td文件中指定。tablegen函数提供了Register类，通过继承这个类，可以表示这些寄存器。

1. 定义硬件编码、命名空间、寄存器、寄存器类

```
let Namespace = "RISCV" in {
class RISCVReg<bits<5> Enc, string n, list<string> alt = []> :
Register<n> {
  let HWEncoding{4-0} = Enc;
  let AltNames = alt;
}
```



```
let RegAltNameIndices = [ABIRegAltName] in {
  def X0  : RISCVReg<0, "x0", ["zero"]>, DwarfRegNum<[0]>;
  let CostPerUse = 1 in {
  def X1  : RISCVReg<1, "x1", ["ra"]>, DwarfRegNum<[1]>;
  def X2  : RISCVReg<2, "x2", ["sp"]>, DwarfRegNum<[2]>;
  def X3  : RISCVReg<3, "x3", ["gp"]>, DwarfRegNum<[3]>;
  def X4  : RISCVReg<4, "x4", ["tp"]>, DwarfRegNum<[4]>;
  def X5  : RISCVReg<5, "x5", ["t0"]>, DwarfRegNum<[5]>;
  def X6  : RISCVReg<6, "x6", ["t1"]>, DwarfRegNum<[6]>;
  def X7  : RISCVReg<7, "x7", ["t2"]>, DwarfRegNum<[7]>;
  }
  def X8  : RISCVReg<8, "x8", ["s0", "fp"]>, DwarfRegNum<[8]>;
  def X9  : RISCVReg<9, "x9", ["s1"]>, DwarfRegNum<[9]>;
  def X10 : RISCVReg<10,"x10", ["a0"]>, DwarfRegNum<[10]>;
  def X11 : RISCVReg<11,"x11", ["a1"]>, DwarfRegNum<[11]>;
  def X12 : RISCVReg<12,"x12", ["a2"]>, DwarfRegNum<[12]>;
  def X13 : RISCVReg<13,"x13", ["a3"]>, DwarfRegNum<[13]>;
  def X14 : RISCVReg<14,"x14", ["a4"]>, DwarfRegNum<[14]>;
  def X15 : RISCVReg<15,"x15", ["a5"]>, DwarfRegNum<[15]>;
  let CostPerUse = 1 in {
```



```
  def X16 : RISCVReg<16,"x16", ["a6"]>, DwarfRegNum<[16]>;
  def X17 : RISCVReg<17,"x17", ["a7"]>, DwarfRegNum<[17]>;
  def X18 : RISCVReg<18,"x18", ["s2"]>, DwarfRegNum<[18]>;
  def X19 : RISCVReg<19,"x19", ["s3"]>, DwarfRegNum<[19]>;
  def X20 : RISCVReg<20,"x20", ["s4"]>, DwarfRegNum<[20]>;
  def X21 : RISCVReg<21,"x21", ["s5"]>, DwarfRegNum<[21]>;
  def X22 : RISCVReg<22,"x22", ["s6"]>, DwarfRegNum<[22]>;
  def X23 : RISCVReg<23,"x23", ["s7"]>, DwarfRegNum<[23]>;
  def X24 : RISCVReg<24,"x24", ["s8"]>, DwarfRegNum<[24]>;
  def X25 : RISCVReg<25,"x25", ["s9"]>, DwarfRegNum<[25]>;
  def X26 : RISCVReg<26,"x26", ["s10"]>, DwarfRegNum<[26]>;
  def X27 : RISCVReg<27,"x27", ["s11"]>, DwarfRegNum<[27]>;
  def X28 : RISCVReg<28,"x28", ["t3"]>, DwarfRegNum<[28]>;
  def X29 : RISCVReg<29,"x29", ["t4"]>, DwarfRegNum<[29]>;
  def X30 : RISCVReg<30,"x30", ["t5"]>, DwarfRegNum<[30]>;
  def X31 : RISCVReg<31,"x31", ["t6"]>, DwarfRegNum<[31]>;
  }
```

• 2.2 定义调用约定

调用约定指的是值如何传递给函数以及如何从函数返回。在RISCV架构中，参数通过寄存器传递，剩下的通过栈传递。

调用约定在RISCVCallingConv.td文件中定义，它主要包含两块内容： 返回值约定和参数传递约定。返回值约定指的是返回值会如何传递以及通过哪个寄存器传递；参数传递约定指的是参数通过栈还是寄存器传递，以及通过哪个寄存器传递。在定义RISCV平台的调用约定时，会继承 CallingConv类。

•2.3 定义指令集

指令目标描述文件定义了3个内容：操作数、汇编字符串、指令格式。具体包括定义或输出列表，以及使用或输入列表。其中也有不同的操作类，如Register类、立即数，以及更复杂的register+imm操作数。

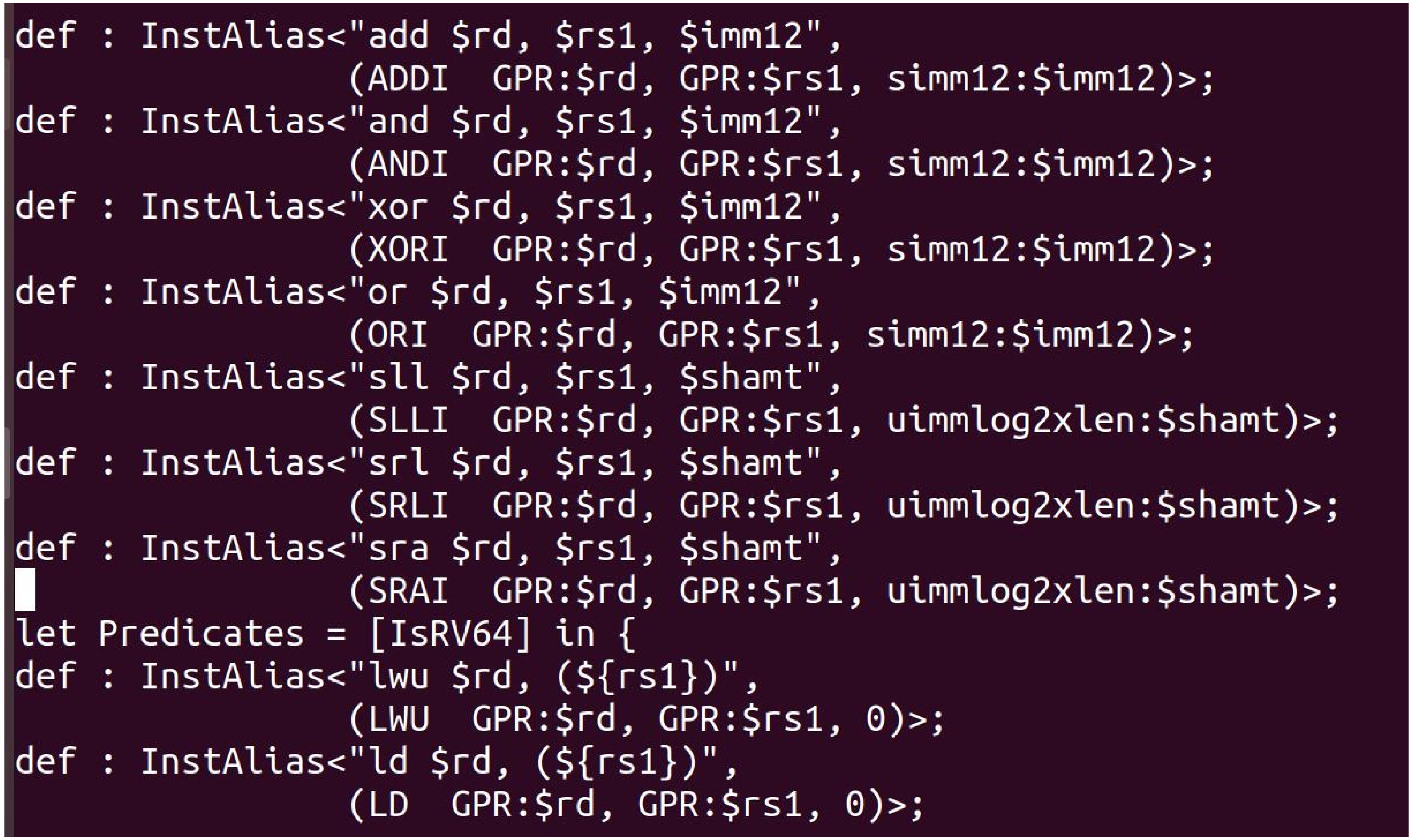
通过目标描述文件来定义指令集。

在lib/Target/RISCV/目录下创建RISCVInstrInfo.td文件：

以RISCVInstrInfo.td 中以TableGen的形式描述了RISC-V instructions

包括：RISCVInstrInfoA.td RISCVInstrInfoC.td RISCVInstrInfoD.td RISCVInstrInfoF.td RISCVInstrInfoM.td

Assembler Pseudo Instructions (User-Level ISA, Version 2.2)



```
def : InstAlias<"add $rd, $rs1, $imm12",
                (ADDI  GPR:$rd, GPR:$rs1, simm12:$imm12)>;
def : InstAlias<"and $rd, $rs1, $imm12",
                (ANDI  GPR:$rd, GPR:$rs1, simm12:$imm12)>;
def : InstAlias<"xor $rd, $rs1, $imm12",
                (XORI  GPR:$rd, GPR:$rs1, simm12:$imm12)>;
def : InstAlias<"or $rd, $rs1, $imm12",
                (ORI  GPR:$rd, GPR:$rs1, simm12:$imm12)>;
def : InstAlias<"sll $rd, $rs1, $shamt",
                (SLLI  GPR:$rd, GPR:$rs1, uimmlog2xlen:$shamt)>;
def : InstAlias<"srl $rd, $rs1, $shamt",
                (SRLI  GPR:$rd, GPR:$rs1, uimmlog2xlen:$shamt)>;
def : InstAlias<"sra $rd, $rs1, $shamt",
                (SRAI  GPR:$rd, GPR:$rs1, uimmlog2xlen:$shamt)>;
let Predicates = [IsRV64] in {
def : InstAlias<"lwu $rd, (${rs1})",
                (LWU  GPR:$rd, GPR:$rs1, 0)>;
def : InstAlias<"ld $rd, (${rs1})",
                (LD  GPR:$rd, GPR:$rs1, 0)>;
```

• 2.4 打印指令

在生成目标代码的过程中，打印汇编指令是很重要的步骤。定义很多类以管道的方式过滤，由之前定义的.td文件提供指令字符串。

打印指令最重要的一步是在.td文件中定义指令字符串，

定义const char *RISCVInstPrinter::getRegisterName函数来打印寄存器名称，定义RISCVMCAsmInfo.h和RISCVMCAsmInfo.cpp文件，指定 MCASMinfo来打印指令

• 参考资料

LLVM 后端移植 指令集代码学习笔记

https://blog.csdn.net/idevede/article/details/90453239

llvm学习笔记（2）

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

http://llvm.org中backend部分

LLVM TableGen 学习笔记

https://blog.csdn.net/idevede/article/details/90417529

以及PCLT实验室tablegen学习视频

《LLVM Codebook》

《Getting Started With LLVM》

LLVM MC项目简介

https://www.cnblogs.com/Proteas/p/3286221.html

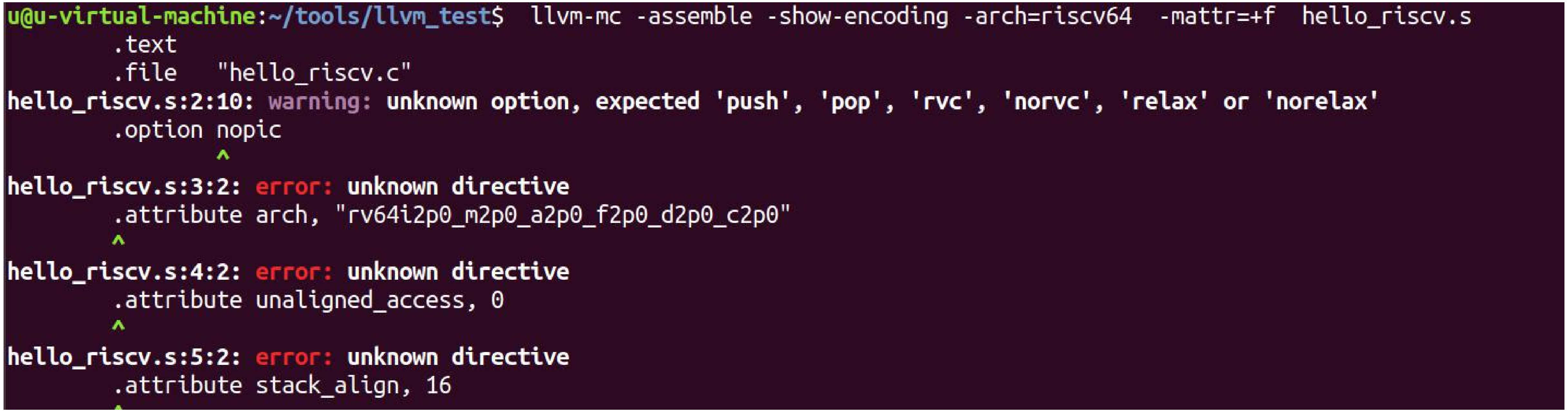
LLVM Programmer's Manual

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

LLVM Language Reference Manual

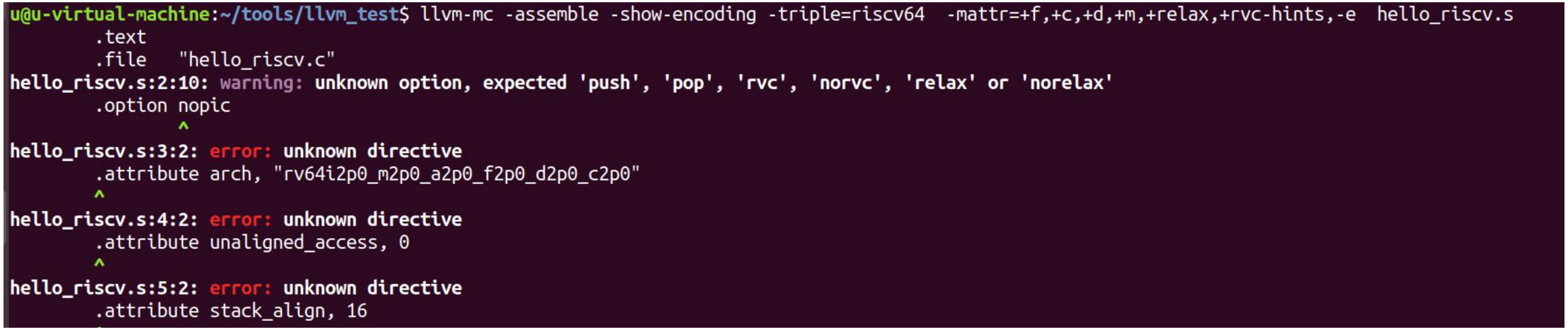
http://llvm.org/docs/LangRef.html

• 问题讨论



```
u@u-virtual-machine:~/tools/llvm_test$  llvm-mc -assemble -show-encoding -arch=riscv64  -mattr=+f  hello_riscv.s
        .text
        .file   "hello_riscv.c"
hello_riscv.s:2:10: warning: unknown option, expected 'push', 'pop', 'rvc', 'norvc', 'relax' or 'norelax'
        .option nopic
                ^
hello_riscv.s:3:2: error: unknown directive
        .attribute arch, "rv64i2p0_m2p0_a2p0_f2p0_d2p0_c2p0"
        ^
hello_riscv.s:4:2: error: unknown directive
        .attribute unaligned_access, 0
        ^
hello_riscv.s:5:2: error: unknown directive
        .attribute stack_align, 16
```

llvm-mc -assemble -show-encoding -arch=riscv64 -mattr=+f hello_riscv.s

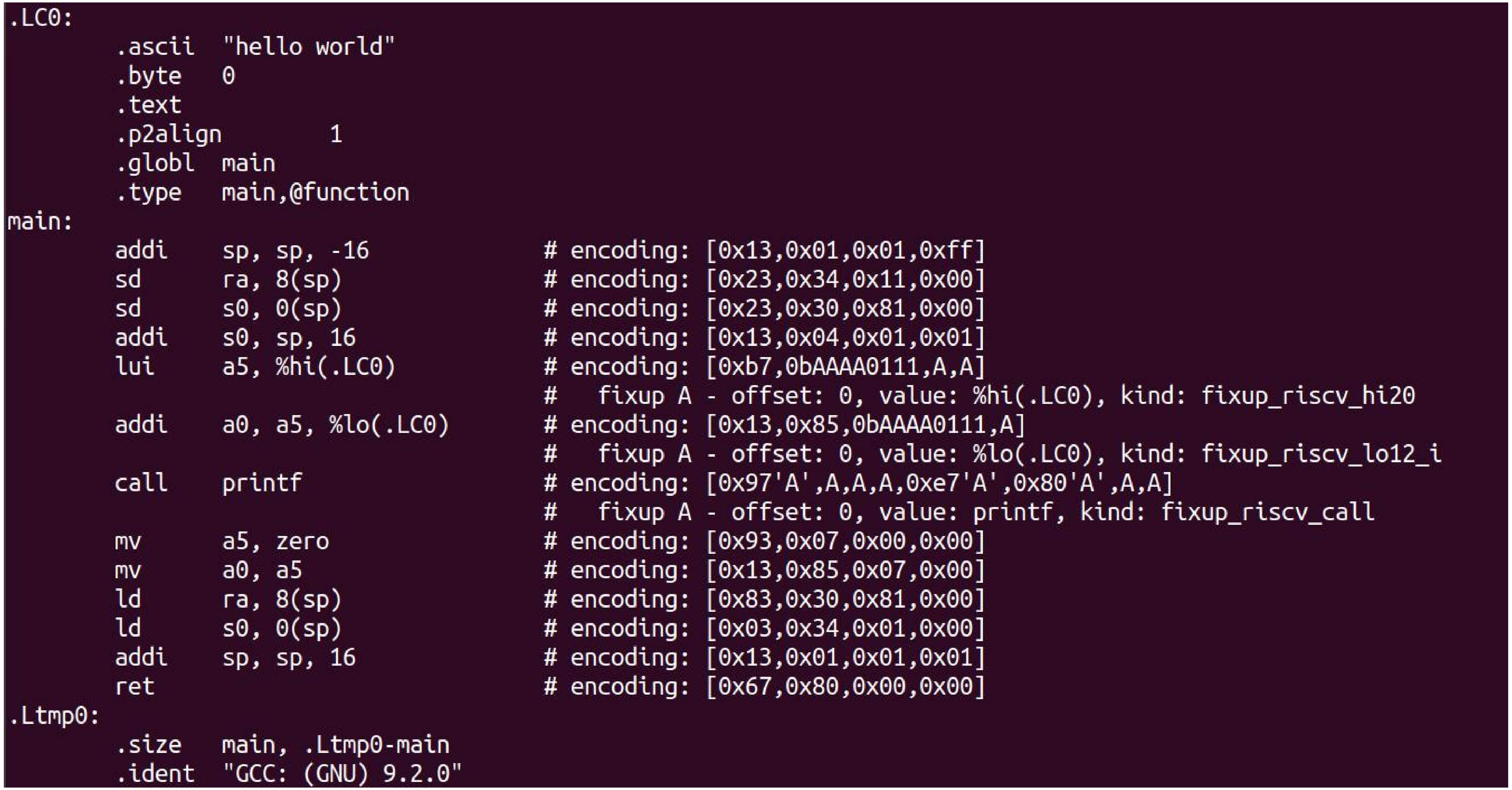


```
u@u-virtual-machine:~/tools/llvm_test$ llvm-mc -assemble -show-encoding -triple=riscv64  -mattr=+f,+c,+d,+m,+relax,+rvc-hints,-e  hello_riscv.s
        .text
        .file   "hello_riscv.c"
hello_riscv.s:2:10: warning: unknown option, expected 'push', 'pop', 'rvc', 'norvc', 'relax' or 'norelax'
        .option nopic
                ^
hello_riscv.s:3:2: error: unknown directive
        .attribute arch, "rv64i2p0_m2p0_a2p0_f2p0_d2p0_c2p0"
        ^
hello_riscv.s:4:2: error: unknown directive
        .attribute unaligned_access, 0
        ^
hello_riscv.s:5:2: error: unknown directive
        .attribute stack_align, 16
```

llvm-mc -assemble -show-encoding -arch=riscv64

-mattr=+f,+c,+d,+m,+relax,+rvc-hints hello_riscv.s

llvm-mc -assemble -show-encoding -arch=riscv64 -mattr=+f hello_riscv.s



```
.LC0:
        .ascii  "hello world"
        .byte   0
        .text
        .p2align        1
        .globl  main
        .type   main,@function
main:
        addi    sp, sp, -16             # encoding: [0x13,0x01,0x01,0xff]
        sd      ra, 8(sp)               # encoding: [0x23,0x34,0x11,0x00]
        sd      s0, 0(sp)               # encoding: [0x23,0x30,0x81,0x00]
        addi    s0, sp, 16              # encoding: [0x13,0x04,0x01,0x01]
        lui     a5, %hi(.LC0)           # encoding: [0xb7,0bAAAA0111,A,A]
                                        #   fixup A - offset: 0, value: %hi(.LC0), kind: fixup_riscv_hi20
        addi    a0, a5, %lo(.LC0)       # encoding: [0x13,0x85,0bAAAA0111,A]
                                        #   fixup A - offset: 0, value: %lo(.LC0), kind: fixup_riscv_lo12_i
        call    printf                  # encoding: [0x97'A',A,A,A,0xe7'A',0x80'A',A,A]
                                        #   fixup A - offset: 0, value: printf, kind: fixup_riscv_call
        mv      a5, zero                # encoding: [0x93,0x07,0x00,0x00]
        mv      a0, a5                  # encoding: [0x13,0x85,0x07,0x00]
        ld      ra, 8(sp)               # encoding: [0x83,0x30,0x81,0x00]
        ld      s0, 0(sp)               # encoding: [0x03,0x34,0x01,0x00]
        addi    sp, sp, 16              # encoding: [0x13,0x01,0x01,0x01]
        ret                             # encoding: [0x67,0x80,0x00,0x00]
.Ltmp0:
        .size   main, .Ltmp0-main
        .ident  "GCC: (GNU) 9.2.0"
```

• 问题讨论参考资料

https://github.com/riscv/riscv-asm-manual/issues/19

https://embarc.org/man-pages/as/RISC_002dV_002dDirectives.html

https://reviews.llvm.org/D46423

谢 谢

欢迎交流合作

2020/02/12