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

软件所智能软件中心PLCT实验室 王鹏 实习生

2020//02/25

目 录

- 01 添加mcpu=C910

在tmp/llvm_source_build/llvm/lib/Support$ vim Triple.cpp中

getArchName () const

getArchName - Get the architecture (first) component of the triple.

getVendorName () const

getVendorName - Get the vendor (second) component of the triple.

getOSName () const

getOSName - Get the operating system (third) component of the triple.

例子llvm-mc -triple=i686-apple-darwin9　　setObjectFormat

```
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/lib/MC$ vim MCSubtargetInfo.cpp
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/lib/MC$ grep -r i686
 u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/lib/MC$  cd ..
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/lib$ grep -r i686
Support/Triple.cpp:    .Cases("i386", "i486", "i586", "i686", Triple::x86)
Target/X86/X86ISelLowering.cpp:    // i686 and for current function we need a base pointer
Target/X86/X86ISelLowering.cpp:    // - which is ESI for i686 - register allocator would not be able to
Target/X86/X86ISelLowering.cpp:    // If it is not i686 or there is no base pointer - nothing to do here.
Target/X86/X86ISelLowering.cpp:                "LCMPXCHG8B custom insertion for i686 is written with X86::ESI as a "
Target/X86/X86ISelLowering.cpp:/// For i686, the best sequence is apparently storing the value and loading
Target/X86/X86.td:def : Proc<"i686", [FeatureX87, FeatureSlowUAMem16, FeatureCMPXCHG8B,
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/lib$ cd Support
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/lib/Support$ vim Triple.cpp
u@u-virtual-machine:~/tools/tmp/llvm_source_build/llvm/lib/Support$
```

Target/X86/X86.td中有基于X86的mcpu定义

```
tmp/llvm/lib/Target/X86$ echo '1 2' | llvm-mc -disassemble -mcpu=help
  (Available CPUs for this target:)
athlon-4        - Select the athlon-4 processor.
barcelona       - Select the barcelona processor.
cannonlake      - Select the cannonlake processor.
goldmont-plus   - Select the goldmont-plus processor.
pentium2        - Select the pentium2 processor.
winchip-c6      - Select the winchip-c6 processor.
```

```
u@u-virtual-machine:~/tools/c910-llvm-master/tmp/llvm$ grep -r --exclude-dir=test athlon-4
lib/Target/X86/X86PfmCounters.td:def : PfmCountersBinding<"athlon-4", DefaultAMDPfmCounters>;
lib/Target/X86/X86.td:foreach P = ["athlon-4", "athlon-xp", "athlon-mp"] in {
docs/tutorial/MyFirstLanguageFrontend/LangImpl08.rst:      athlon-4      - Select the athlon-4 processor.
docs/tutorial/LangImpl08.rst:      athlon-4      - Select the athlon-4 processor.
u@u-virtual-machine:~/tools/c910-llvm-master/tmp/llvm$ grep -r --exclude-dir=test barcelona
lib/Target/X86/X86PfmCounters.td:def : PfmCountersBinding<"barcelona", DefaultAMDPfmCounters>;
lib/Target/X86/X86.td:foreach P = ["amdfam10", "barcelona"] in {
include/llvm/Support/X86TargetParser.def:X86_CPU_SUBTYPE_COMPAT("amdfam10",      AMDFAM10H_BARCELONA,         "barcelona"
)
u@u-virtual-machine:~/tools/c910-llvm-master/tmp/llvm$ grep -r --exclude-dir=test cannonlake
lib/Support/Host.cpp:      *Subtype = X86::INTEL_COREI7_CANNONLAKE; // "cannonlake"
lib/Target/X86/X86PfmCounters.td:def : PfmCountersBinding<"cannonlake", SkylakeServerPfmCounters>;
lib/Target/X86/X86.td:def : ProcessorModel<"cannonlake", SkylakeServerModel,
include/llvm/Support/X86TargetParser.def:X86_CPU_SUBTYPE_COMPAT("cannonlake",     INTEL_COREI7_CANNONLAKE,     "cannonlake
")
```

```
u@u-virtual-machine:~/tools/c910-llvm-master/tmp/llvm$ grep -r --exclude-dir=test goldmont-plus
lib/Target/X86/X86PfmCounters.td:def : PfmCountersBinding<"goldmont-plus", SLMPfmCounters>;
lib/Target/X86/X86.td:def : ProcessorModel<"goldmont-plus", SLMModel, ProcessorFeatures.GLPFeatures>;
include/llvm/Support/X86TargetParser.def:X86_CPU_TYPE_COMPAT           ("goldmont-plus", INTEL_GOLDMONT_PLUS, "goldmont-pl
us")
u@u-virtual-machine:~/tools/c910-llvm-master/tmp/llvm$ grep -r --exclude-dir=test pentium2
lib/Target/X86/X86PfmCounters.td:def : PfmCountersBinding<"pentium2", PentiumPfmCounters>;
lib/Target/X86/X86.td:def : Proc<"pentium2",        [FeatureX87, FeatureSlowUAMem16, FeatureCMPXCHG8B,
lib/Target/X86/README.txt:gcc compiles this to the following when using march/mtune=pentium2/3/4/m/etc.:
include/llvm/Support/X86TargetParser.def:X86_CPU_TYPE                  ("pentium2",    INTEL_PENTIUM_II)
u@u-virtual-machine:~/tools/c910-llvm-master/tmp/llvm$ grep -r --exclude-dir=test winchip-c6
lib/Target/X86/X86.td:def : Proc<"winchip-c6",      [FeatureX87, FeatureSlowUAMem16, FeatureMMX]>;
```

Target/X86/X86.td Target definition file for the Intel X86

X86PfmCounters.td This describes the available hardware counters for various subtargets

有基于X86的mcpu定义

首先在lib/Target/X86/X86.td中进行分析

X86.td描述了以下信息：

X86 Subtarget state（Mode64/32/16Bit）

X86 Subtarget features

Register File Description

Instruction Descriptions

X86 Processor Feature Lists

X86 processors supported.

```
foreach P = ["athlon-4", "athlon-xp", "athlon-mp"] in {
  def : Proc<P, [FeatureX87, FeatureSlowUAMem16, FeatureCMPXCHG8B, FeatureCMOV,
                 FeatureSSE1, Feature3DNowA, FeatureFXSR, FeatureNOPL,
                 FeatureSlowSHLD]>;
}
```

对于athlon-4，athlon-xp和athlon-mp这三个cpu来说，它们具有”[]“的哪些feature，也可以直接定义

def : Proc<”athlon-4”,[features]>

```
// AMD X86 Counters.
// Set basic counters for AMD cpus that we know libpfm4 supports.
def DefaultAMDPfmCounters : ProcPfmCounters {
  let CycleCounter = PfmCounter<"cpu_clk_unhalted">;
  let UopsCounter = PfmCounter<"retired_uops">;
}
def : PfmCountersBinding<"athlon", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon-tbird", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon-4", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon-xp", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon-mp", DefaultAMDPfmCounters>;
```

AMD Athlon 4处理器的hardware counters选择默认的AMD PFM counters （作用是衡量硬件的计数器）

CycleCounter = CpuCyclesPfmCounter

具体定义在X86/X86PfmCounters.td中

```
foreach P = ["amdfam10", "barcelona"] in {
  def : Proc<P, ProcessorFeatures.BarcelonaFeatures>;
}
```

AMD发布Barcelona处理器

// Barcelona

list<SubtargetFeature> BarcelonaInheritableFeatures = [FeatureX87,FeatureCMPXCHG8B,FeatureSSE4A,Feature3DNowA,...];

list<SubtargetFeature> BarcelonaFeatures = BarcelonaInheritableFeatures;

```
// AMD X86 Counters.
// Set basic counters for AMD cpus that we know libpfm4 supports.
def DefaultAMDPfmCounters : ProcPfmCounters {
  let CycleCounter = PfmCounter<"cpu_clk_unhalted">;
  let UopsCounter = PfmCounter<"retired_uops">;
}
def : PfmCountersBinding<"athlon", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon-tbird", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon-4", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon-xp", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon-mp", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"k8", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"opteron", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon64", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon-fx", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"k8-sse3", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"opteron-sse3", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"athlon64-sse3", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"amdfam10", DefaultAMDPfmCounters>;
def : PfmCountersBinding<"barcelona", DefaultAMDPfmCounters>;
```

```
def : Proc<"pentium2",          [FeatureX87, FeatureSlowUAMem16, FeatureCMPXCHG8B,
                                 FeatureMMX, FeatureCMOV, FeatureFXSR,
                                 FeatureNOPL]>;
```

```
// Intel X86 Counters.
def PentiumPfmCounters : ProcPfmCounters {
  let CycleCounter = PfmCounter<"cpu_clk_unhalted">;
  let UopsCounter = PfmCounter<"uops_retired">;
}
def : PfmCountersBinding<"pentiumpro", PentiumPfmCounters>;
def : PfmCountersBinding<"pentium2", PentiumPfmCounters>;
def : PfmCountersBinding<"pentium3", PentiumPfmCounters>;
def : PfmCountersBinding<"pentium3m", PentiumPfmCounters>;
def : PfmCountersBinding<"pentium-m", PentiumPfmCounters>;
```

我们在RISCV/RISCV.td 中定义mcpu=C910，参考pentium2在X86/X86.td中的定义

在RISCV/RISCV.td中

```
//===----------------------------------------------------------------------===//
// RISCV processors supported.
//===----------------------------------------------------------------------===//
class Proc<string Name, list<SubtargetFeature> Features>
 : ProcessorModel<Name, GenericModel, Features>;
def : Proc<"C910",          [FeatureStdExtM, FeatureStdExtA,Feature64Bit,
                             FeatureStdExtF, FeatureStdExtC,FeatureStdExtD]>;
//===----------------------------------------------------------------------===//
// Pfm Counters
//===----------------------------------------------------------------------===//
include "RISCVPfmCounters.td"
```

Based on
编号 玄铁 C910 指令集手册
版本号 1.0.0

在RISCV/RISCVPfmCounters.td中（1）

```
//===-- RISCVPfmCounters.td - RISCV Hardware Counters -------*- 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
//
//===----------------------------------------------------------------------===//
//
// This describes the available hardware counters for RISCV.
//
//===----------------------------------------------------------------------===//
```

在RISCV/RISCVPfmCounters.td中（2）定义性能计数器

```
// RISCV Hardware Counters
def CpuCyclesPfmCounter : PfmCounter<"CYCLES">;

def DefaultPfmCounters : ProcPfmCounters {
  let CycleCounter = CpuCyclesPfmCounter;
}
def : PfmCountersDefaultBinding<DefaultPfmCounters>;
```

测试

echo '1 2' | llvm-mc -disassemble -mcpu=C910

```
u@u-virtual-machine:~/tools/tmp/llvm_source_build/build$ echo '1 2' | llvm-mc -disassemble -mcpu=C910
'C910' is not a recognized processor for this target (ignoring processor)
'C910' is not a recognized processor for this target (ignoring processor)
        .text
        addl    %eax, (%rdx)
```

```
llvm/tools/clang/include/clang/Basic/X86Target.def:PROC(Pentium2, "pentium2", PROC_32_BIT)
llvm/tools/clang/lib/Basic/Targets/X86.cpp:    Builder.defineMacro("__tune_pentium2__");
llvm/tools/clang/lib/Driver/ToolChains/Darwin.cpp:    else if (Name == "pentium2")
llvm/tools/clang/lib/Driver/ToolChains/Darwin.cpp:      DAL->AddJoinedArg(nullptr, MArch, "pentium2");
llvm/tools/clang/lib/Driver/ToolChains/Darwin.cpp:      DAL->AddJoinedArg(nullptr, MArch, "pentium2");
llvm/lib/Target/X86/X86.td:def : Proc<"pentium2",        [FeatureX87, FeatureSlowUAMem16, FeatureCMPXCHG8B,
llvm/lib/Target/X86/X86PfmCounters.td:def : PfmCountersBinding<"pentium2", PentiumPfmCounters>;
llvm/lib/Target/X86/README.txt:gcc compiles this to the following when using march/mtune=pentium2/3/4/m/etc.:
```

在llvm/tools/clang/include/clang/Basic/X86Target.def中

PROC(Pentium2, "pentium2", PROC_32_BIT)

C910可以参考

建立一个RISCVarget.def来指定RISCV的cpu，核心代码

/// \name C910

PROC(C910, "c910", PROC_64_BIT)

```
/// \name i686
/// i686-generation processors, P6 / Pentium M microarchitecture based.
//@{
PROC(PentiumPro, "pentiumpro", PROC_32_BIT)
PROC(i686, "i686", PROC_32_BIT)
PROC(Pentium2, "pentium2", PROC_32_BIT)
PROC(Pentium3, "pentium3", PROC_32_BIT)
```

• 02 RISCV后端介绍(2)

2.5 选择指令

RISCVISelDAGToDAG.cpp文件

（1）定义RISCVDAGToDAGISel类，继承自SelectionDAGISel类。

```
class RISCVDAGToDAGISel final : public SelectionDAGISel {
  const RISCVSubtarget *Subtarget = nullptr;

public:
  explicit RISCVDAGToDAGISel(RISCVTargetMachine &TargetMachine)
      : SelectionDAGISel(TargetMachine) {}
```

这个类中要定义的最重要的函数是Select()，它依据机器指令返回一个SDNode对象，Select()函数用于选择DAG节点的操作码。在类中声明：

void Select(SDNode *Node) override;

```
void RISCVDAGToDAGISel::Select(SDNode *Node) {
  // If we have a custom node, we have already selected.
  if (Node->isMachineOpcode()) {
    LLVM_DEBUG(dbgs() << "== "; Node->dump(CurDAG); dbgs() << "\n");
    Node->setNodeId(-1);
    return;
  }
```

（2）另一个重要的函数是定义地址选择函数，它会计算加载、存储操作的基址和偏移。RISCVDAGISel::SelectAddrFI()则用于选择 DataDAG节点的addr类型，如果地址是全局的或者是外部的，则返回false，因为它需要在全局上下文中计算。声明如下：

bool SelectAddrFI(SDValue Addr, SDValue &Base);

```
bool RISCVDAGToDAGISel::SelectAddrFI(SDValue Addr, SDValue &Base) {
  if (auto FIN = dyn_cast<FrameIndexSDNode>(Addr)) {
    Base = CurDAG->getTargetFrameIndex(FIN->getIndex(), Subtarget->getXLenVT());
    return true;
  }
  return false;
}
```

```
// Only attempt this optimisation for I-type loads and S-type stores.
switch (N->getMachineOpcode()) {
default:
  continue;
case RISCV::LB:
case RISCV::LH:
case RISCV::LW:
case RISCV::LBU:
case RISCV::LHU:
case RISCV::LWU:
case RISCV::LD:
case RISCV::FLW:
case RISCV::FLD:
  BaseOpIdx = 0;
  OffsetOpIdx = 1;
  break;
case RISCV::SB:
case RISCV::SH:
case RISCV::SW:
case RISCV::SD:
case RISCV::FSW:
case RISCV::FSD:
  BaseOpIdx = 1;
  OffsetOpIdx = 2;
  break;
}
```

（3）最后，createRISCVISelDag 把合法的DAG转为RISCV平台的 DAG，并且为定义于同一文件中的指令调度做准备：

```
// This pass converts a legalized DAG into a RISCV-specific DAG, ready
// for instruction scheduling.
FunctionPass *llvm::createRISCVISelDag(RISCVTargetMachine &TM) {
  return new RISCVDAGToDAGISel(TM);
}
```

2.6 增加指令编码

如果指令需要进行编码，即如何用位字段表示，那么可以在.td文 件中定义指令的时候指定位字段。

（1）. 对于注册add指令的一个寄存器操作数，会有一些定义的指令编码。指令的大小是32位的，它的编码如下：

bits 0 to 6 -> 0110011

bits 7 to 11 -> rd, for destination registerall zeros

bits 12 to 14 -> 000

bits 15 to 19 -> rs1, first register operand

bit 20 to 24 -> rs2, second register operand

bit 25 to 31 -> all zeros

这可通过在.td文件中指定位模式来实现。

以上编码可通过在.td文件中指定位模式来实现

（2）在RISCVInstrFormats.td文件中定义名为Inst的变量：

```
Instruction {
  field bits<32> Inst;
  let Inst{6-0} = Opcode;
  let Namespace = "RISCV";
```

（3）在RISCVInstrInfo.td文件中定义指令编码：
add指令对应的ALU_rr

```
let hasSideEffects = 0, mayLoad = 0, mayStore = 0 in
class ALU_rr<bits<7> funct7, bits<3> funct3, string opcodestr>
    : RVInstR<funct7, funct3, OPC_OP, (outs GPR:$rd), (ins GPR:$rs1, GPR:$rs2),
              opcodestr, "$rd, $rs1, $rs2">;
```

(4) 在RISCV/MCTargetDesc/RISCVMCCodeEmitter.cpp文件中，如果机器指令的操作数是寄存器，那么就会调用编码函数

```
unsigned
RISCVMCCodeEmitter::getMachineOpValue(const MCInst &MI, const MCOperand &MO,
                                      SmallVectorImpl<MCFixup> &Fixups,
                                      const MCSubtargetInfo &STI) const {

  if (MO.isReg())
    return Ctx.getRegisterInfo()->getEncodingValue(MO.getReg());

  if (MO.isImm())
    return static_cast<unsigned>(MO.getImm());
```

（5） 在同一个文件中，指定编码指令的函数。RISCVMCCodeEmitter.cpp主要实现的功能是将RISCV代码转换为机器代码

```
void RISCVMCCodeEmitter::encodeInstruction(const MCInst &MI, raw_ostream &OS,
                                           SmallVectorImpl<MCFixup> &Fixups,
                                           const MCSubtargetInfo &STI) const {
  const MCInstrDesc &Desc = MCII.get(MI.getOpcode());
  // Get byte count of instruction.
  unsigned Size = Desc.getSize();

  if (MI.getOpcode() == RISCV::PseudoCALLReg ||
      MI.getOpcode() == RISCV::PseudoCALL ||
      MI.getOpcode() == RISCV::PseudoTAIL ||
      MI.getOpcode() == RISCV::PseudoJump) {
    expandFunctionCall(MI, OS, Fixups, STI);
    MCNumEmitted += 2;
    return;
  }
```

参考资料

llvm学习手册指导
https://github.com/llvm/llvm-project/blob/master/llvm/docs/tutorial
llvm::Triple Class Reference
https://llvm.org/doxygen/classllvm_1_1Triple.html#details
llvm学习笔记（3）
https://blog.csdn.net/wuhui_gdnt/article/details/62884600
LLVM中指令的一生
https://blog.csdn.net/wuhui_gdnt/article/details/40978431?utm_source=tuicool&utm_medium=referral

课后问题

用echo '1 2' | llvm-mc -disassemble -mcpu=help来测试现有的mcpu，发现mcpu=C910还没能加入到命令行选项中。

解决方法是

（1）在llvm/tools/clang/include/clang/Basic目录中建立RISCVTarget.def来描述新的cpu，或者在其他哪个文件中定义？

llvm/test/CodeGen暂时不用修改，对添加mcpu=C910没有帮助

“//”注释和测试文件暂时不用添加

```
u@u-virtual-machine:~/tools/tmp/llvm_source_build$ grep -r pentium2
llvm/test/CodeGen/X86/cmov-fp.ll:; RUN: llc -mtriple=i686-- -mcpu pentium2 < %s | FileCheck %s -check-prefix=NOSSE1
llvm/test/CodeGen/X86/cpus-intel.ll:; RUN: llc < %s -o /dev/null -mtriple=i686-unknown-unknown -mcpu=pentium2 2>&1 | FileC
heck %s --check-prefix=CHECK-NO-ERROR --allow-empty
llvm/test/CodeGen/X86/memset.ll:; RUN: llc < %s -mcpu=pentium2 -mtriple=i686-apple-darwin8.8.0 | FileCheck %s --check-pref
ix=X86
llvm/test/CodeGen/X86/memset-sse-stack-realignment.ll:; RUN: llc < %s -mtriple=i386-pc-mingw32 -mcpu=pentium2 | FileCheck
%s --check-prefix=NOSSE
llvm/test/CodeGen/X86/cpus-intel-no-x86_64.ll:; RUN: not llc < %s -o /dev/null -mtriple=x86_64-unknown-unknown -mcpu=penti
um2 2>&1 | FileCheck %s --check-prefix=CHECK-ERROR64
```

```
llvm/include/llvm/Support/X86TargetParser.def:X86_CPU_TYPE                    ("pentium2",    INTEL_PENTIUM_II)
llvm/tools/clang/test/Misc/target-invalid-cpu-note.c:// X86-SAME: i586, pentium, pentium-mmx, pentiumpro, i686, pentium2,
pentium3,
llvm/tools/clang/test/Preprocessor/predefined-arch-macros.c:// CHECK_C3_2_M32: #define __tune_pentium2__ 1
llvm/tools/clang/test/Preprocessor/predefined-arch-macros.c:// RUN: %clang -march=pentium2 -m32 -E -dM %s -o - 2>&1 \
llvm/tools/clang/test/Preprocessor/predefined-arch-macros.c:// CHECK_PENTIUM2_M32: #define __tune_pentium2__ 1
llvm/tools/clang/test/Preprocessor/predefined-arch-macros.c:// RUN: not %clang -march=pentium2 -m64 -E -dM %s -o - 2>&1 \
llvm/tools/clang/test/Preprocessor/predefined-arch-macros.c:// CHECK_PENTIUM3_M32: #define __tune_pentium2__ 1
```

```
llvm/tools/clang/include/clang/Basic/X86Target.def:PROC(Pentium2, "pentium2", PROC_32_BIT)
llvm/tools/clang/lib/Basic/Targets/X86.cpp:    Builder.defineMacro("__tune_pentium2__");
llvm/tools/clang/lib/Driver/ToolChains/Darwin.cpp:    else if (Name == "pentium2")
llvm/tools/clang/lib/Driver/ToolChains/Darwin.cpp:      DAL->AddJoinedArg(nullptr, MArch, "pentium2");
llvm/tools/clang/lib/Driver/ToolChains/Darwin.cpp:      DAL->AddJoinedArg(nullptr, MArch, "pentium2");
llvm/lib/Target/X86/X86.td:def : Proc<"pentium2",        [FeatureX87, FeatureSlowUAMem16, FeatureCMPXCHG8B,
llvm/lib/Target/X86/X86PfmCounters.td:def : PfmCountersBinding<"pentium2", PentiumPfmCounters>;
llvm/lib/Target/X86/README.txt:gcc compiles this to the following when using march/mtune=pentium2/3/4/m/etc.:
```

welcome to lowRISC > **Add a new mcpu=C910 option in llvm-mc** Feb 22

John 9:44 PM

Hi guys
Glad to be here 😀
Nowadays I'm going to add a new option about llvm-mc command line which is mcpu=C910, which is also a RISCV supported machine cpu, Anyone could tell me what to do? I can't find some materials related about adding a new mcpu=C910, but there did many mcpus when I type "llvm-mc --mcpu=help", I don't know how they can do that.
thank you
Best Regards

Luís Marques 11:28 PM

Hi John. Are you sure you need to add a CPU to that list? Often you can just change the -march=... and -mabi=... options passed to the compiler to match the characteristics of your RISC-V target system.

welcome to lowRISC **Add a new mcpu=C910 option in llvm-mc** Yesterday

John 12:00 AM

thank you dear Luís Marques
I have added the mcpu=C910 now， which is similar to i386,i486,i686 from X86. But I can't find mabi= option, do you know where is that? I use "grep -r mabi "command in the llvm directory.
thank you again
Best Regards 🙂👍

Luís Marques 12:02 AM

What's your overall goal?

John 3:42 PM

My overall goal is to set up one mcpu=C910 based on RISCV, then I can use " llvm-mc -mcpu=C910 " , I need to change some files, but I don't know which file to change.

Luís Marques 8:43 PM

And do you feel you need that because you don't want to type something generic like `llvm-mc -triple=riscv32 -mattr=...`, or do you really need to customise something for that CPU? Also, do notice that people typically don't use `llvm-mc` directly, but instead use a more user-friendly frontend like clang.

▲ YESTERDAY ▼ TODAY

谢 谢

欢迎交流合作

2019/02/25